FernUniversität in Hagen
Faculty of Mathematics and Computer Science

AIG
Artificial Intelligence Group

Logic Programming for a Global Logistics Problem

Master's Thesis

in partial fulfillment of the requirements for
the degree of Master of Science (M.Sc.)
in Praktische Informatik

submitted by
Olcay Altay-Kern

First examiner:	Prof. Dr. Matthias Thimm Artificial Intelligence Group
Advisor:	Dr. Emmanuelle Dietz Airbus

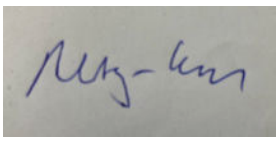
Statement

I declare that I have written the master´s thesis independently and without unauthorized use of third parties. I have only used the indicated resources and I have clearly marked the passages taken verbatim or in the sense of these resources as such. The assurance of independent work also applies to any drawings, sketches or graphical representations. The work has not previously been submitted in the same or similar form to the same or another examination authority and has not been published. By submitting the electronic version of the final version of the master´s thesis, I acknowledge that it will be checked by a plagiarism detection service to check for plagiarism and that it will be stored exclusively for examination purposes.

I explicitly agree to have this thesis published on the webpage of the artificial intelligence group and endorse its public availability.

Software created for this work has been made available as open source; a corresponding link to the sources is included in this work. The same applies to any research data.

Signed by:Olcay Altay Kern
Signed at:2025-07-14 12:09:00 +02:00
Reason:I Approve this document



AIRBUS

. . . . . .

(Place, Date) (Signature)

Zusammenfassung

Abstract

Logic programming has demonstrated to be a compelling alternative to imperative methodologies for addressing combinatorial optimization problems. Within an industrial setting, an Airbus R&T (Research and Technology) team has adopted ASP (Answer Set Programming) as the programming paradigm to find valid solutions for the global manufacturing setup of a future commercial aircraft. Initial research efforts have provided promising outcomes, however the obtained answer sets do not meet the expectations regarding the resulting KPIs (Key Performance Indicators) and variability of solutions. In this thesis additional capabilities from the used framework were implemented to improve answer sets significantly with regards to the objective functions. The use of domain-heuristics and weak constraints combined with an active design of optimization statements led to answer sets which are almost optimal despite a vast solution space. The industrial systems engineers can explore diverse solutions from a range of near-optimal answer sets and find desirable configurations with relatively low runtime.

Contents

1. Introduction **2**
- 1.1. Motivation and Background . . . 2
- 1.2. Problem Statement . . . 3
  - 1.2.1. Generalization of the Problem and Solution Space . . . 4
  - 1.2.2. Related Problems . . . 8
  - 1.2.3. Objective Functions and Optimization Statements . . . 9
  - 1.2.4. Search Tree Traversal . . . 10
  - 1.2.5. Hypotheses . . . 11
- 1.3. Industrial System Scenarios and Constraints . . . 12
- 1.4. Objectives . . . 13
- 1.5. Current State of Research . . . 17
- 1.6. Outline . . . 18

2. Theoretical Background **20**
- 2.1. Logic Programming . . . 20
- 2.2. Answer Set Programming (ASP) and Stable Model Semantics . . . 23
- 2.3. *Potassco* Toolset . . . 25
- 2.4. Input Language of *clingo* . . . 26
  - 2.4.1. The Grounder *gringo* . . . 29
  - 2.4.2. The Solver *clasp* . . . 29

3. Representation of the Logistics System and the Optimization Problem **32**
- 3.1. Representation of the Logistics System . . . 32
- 3.2. Major Issues of the initial Research Work . . . 34
  - 3.2.1. Optimization Statements . . . 34
  - 3.2.2. Usage of Aggregates . . . 36
- 3.3. Overview of New Features and Expected Impact . . . 38
  - 3.3.1. Main Objective Functions . . . 38
  - 3.3.2. Search Tree Traversal . . . 38

4. Methodology **41**
- 4.1. Design of Experiments . . . 41
- 4.2. Evaluation Metrics . . . 42
  - 4.2.1. Key Performance Indicators of Objective Functions . . . 42
  - 4.2.2. Variability Measurement . . . 43
  - 4.2.3. Search Performance . . . 44

5. Results of Main Phase **45**
- 5.1. Baseline Reference . . . 45
- 5.2. Workshare Constraint Implementations . . . 49
  - 5.2.1. KPI evolution . . . 50
  - 5.2.2. Variability . . . 51

5.2.3. Search Performance . . . 52
5.3. Domain Heuristics . . . 54
5.3.1. KPI evolution . . . 54
5.3.2. Variability . . . 54
5.3.3. Search Performance . . . 56
5.4. Weak Constraints . . . 56
5.4.1. KPI evolution . . . 56
5.4.2. Variability . . . 58
5.4.3. Search Performance . . . 58
5.5. Optimization Statements . . . 58
5.5.1. KPI evolution . . . 59
5.5.2. Variability . . . 60
5.5.3. Search Performance . . . 61
5.6. Consolidated Result . . . 61

6. Discussion of Main Phase **64**

7. Results of Final Phase **68**
7.1. KPI evolution . . . 68
7.2. Variability . . . 69
7.3. Search Performance . . . 69

8. Discussion of Final Phase **71**

9. Conclusion **72**
9.1. Summary of Contributions . . . 72
9.2. Future Work . . . 73

A. Appendix **78**

List of Figures

1. Exemplary aircraft breakdown and associated production and transportation sequence . . . 4
2. Principle of a Multi-Echelon Production Network for an aircraft . . . 7
3. Impact of warehouses on the routing . . . 9
4. Multi-scatter plot of answer sets evolution showing all KPI combinations. Source: [1] . . . 10
5. Pareto optimal answer sets in red forming a Pareto Front. Source: [2] 11
6. Three answer sets with different values for the two observed objective functions $function\ 1$ and $function\ 2$. Source: [2] . . . 12
7. Part of a full Multi-Echelon Production Network . . . 14
8. One selected solution with four different paths within the Multi-Echelon Production Network . . . 15
9. Design parameters and outputs . . . 16
10. Typical two-step architecture of ASP solutions like *clingo*, Source: [3] 24
11. Single-Sourcing vs Multi-Sourcing: Multiple paths . . . 37
12. Evolution of the Production Costs with respect to the Transportation Costs within the (baseline) run . . . 47
13. Evolution of the Production Costs with respect to the CO_2 costs within the (baseline) run . . . 48
14. Evolution of the Transportation Costs with respect to the CO_2 costs within the (baseline) run . . . 49
15. Evolution of the similarity values within the baseline run. . . . 50
16. Evolution of the KPIs for the different Workshare implementations . 50
17. Evolution of the Production Costs with respect to the Transportation Costs for the different Workshare implementations . . . 52
18. Similarity evolution of two single configurations . . . 52
19. Similarity evolution of two combined configurations . . . 53
20. Overview of the resulting KPIs from the domain heuristics runs . . . 55
21. Overview of the resulting KPIs from the Weak Constraint runs . . . 57
22. Overview of the resulting KPIs from the optimization statement runs 59
23. Evolution of the Production Costs with respect to the Transportation Costs for the tested optimization statements . . . 60
24. Evolution of the Production Costs with respect to the Transportation Costs for all tested features . . . 62
25. Overview of the resulting KPIs from the final configuration runs . . . 69

List of Tables

1. Overview of all facts of the logic program. 33
2. Relevant rules and optimization statements of the (baseline) program excluding facts. 35
3. Overview of all new features to be tested 40
5. The number of models found for the (baseline) run 45
6. The reference KPIs of the (baseline) run. 45
4. The reference model last (baseline) printed as a table. 46
7. KPI percentages of the baseline comparing the first and the last model 47
8. The KPI percentages of each heuristic in relation to the last (baseline) model. The best results are highlighted in gray. 51
9. Similarity comparison for all Workshare implementations for production sites (p) and transportation similarities (t). 53
10. The number of models found for the different Workshare implementations 53
11. The KPI percentages of each heuristic in relation to the last (baseline) model. The best results are highlighted in gray. 55
12. Similarity comparison for all heuristics, with respect to the production sites (p) and transportation means (t). 55
13. The number of models found for the different heuristics 56
14. The KPI percentages of each Weak Constraint in relation to the last (baseline) model. The best results are highlighted in gray. 57
15. Similarity comparison for all Weak Constraints, with respect to the production sites (p) and transportation means (t). 58
16. The number of models found for the different Weak Constraints . . 58
17. The KPI percentages of each optimization statement in relation to the last (baseline) model. The best results are highlighted in gray. 60
18. Similarity comparison for all optimization statements with respect to the production sites (p) and transportation means (t). 61
19. The number of models found for the different optimization statements 61
20. The KPI percentages of each final configuration in relation to the last (baseline) model. 68
21. Similarity comparison for all final configurations, with respect to the production sites (p) and transportation means (t). 69
22. The number of models found for the different final configurations . 70

Acronyms

ASP Answer Set Programming.

CDCL Conflict-Driven Clause Learning.

CDNL Conflict-Driven Nogood Learning.

DAG Directed Acyclic Graph.

DPLL Davies-Puttnam-Logemann-Loveland algorithm.

KPI Key Performance Indicator.

R&T Research & Technology.

SAT Boolean Satisfiability Problem.

SMC Set Multicover Problem.

TSP Traveling Salesperson Problem.

VSIDS Variable State Independent Decaying Sum.

1. Introduction

This section outlines the motivation behind the research and provides necessary background information. It includes the following content:

- *Motivation and Background*: A description of the relevance of the research area and the broader context of this work
- *Problem Statement*: A formulation of the general and specific problems to be addressed in this thesis
- *Related Problems*: A mentioning of similar or related problems
- *Objectives*: The research goals and expected outcomes
- *Current State of Research*: A short overview of previous and related works, the current state and possible gaps in the literature

1.1. Motivation and Background

Imagine being within an immense labyrinth of galactic scale, with only a few known honey pots. Given such vast dimensions, the likelihood of locating one within a lifetime, even with knowledge of a direct route and moving with the speed of light, is close to zero. This analogy serves as a representation of real-world challenges encountered in industrial applications, where navigating immense solution spaces with numerous constraints is commonplace [4]. These combinatorial optimization problems are symbolized as a massive labyrinth, representing the solution space, where the honey pots symbolize the local optima and the moving speed the computational power.

In any combinatorial optimization problem characterized by such a vast solution space, guidance is essential to even approach the targets. One would benefit from multiple starting points scattered across different regions of the labyrinth, ideally in promising areas. Additionally, indicators confirming whether one's directional choices are correct would be helpful. Without such aids, achieving the objective – or finding a way in the labyrinth in this metaphorical scenario – is almost impossible.

This thesis focuses on a use case within the industrial sector, specifically concerning the development and manufacturing of a next-generation civil passenger aircraft.

In the conceptual-design stage of a commercial aircraft and its manufacturing system, industrial architects must generate and compare multiple production architectures before any landmark investment decisions are taken by the senior management. Multiple scenarios need to be analyzed and optimized with regards to some predefined Key Performance Indicator (KPI) [5, 6], typically recurring production and transportation costs or non-recurring costs of investments.

The difficulty of manufacturing such a product necessitates a multitude of resources. The product design defines the manufacturing build process, which includes workloads and technologies, requiring diverse sites equipped with various expertise and resources. Moreover, in the politically charged atmosphere of the aerospace industry, numerous commitments to partners must be upheld, involving some dozens of major manufacturing sites across the world, each responsible for producing or assembling components [7, 8]. The main challenge is to identify the optimal configuration for a manufacturing setup that minimizes logistics efforts while keeping production cost low.

Given – for instance – the limitations of transportation means, such as the necessity of a harbor for ship use, this task represents a common combinatorial optimization problems in industrial applications. A trivial solution – manufacturing everything at one site centrally – is unfeasible due to the substantial number of constraints as indicated above.

To address this combinatorial challenge, an Airbus central Research & Technology (R&T) team has adopted Answer Set Programming (ASP). ASP is a subclass of logic programming which belongs to the declarative programming paradigms. ASP is based on the *Stable Model Semantics* and has demonstrated significant efficacy in resolving such problems. It is regarded as one of the most successful domains of artificial intelligence [9]. In this case the solver is supposed to identify valid solutions and optimize with regards to a number of different figures of merit, such as production or transportation costs. A valid solution is a solution that ensures that all required manufacturing steps are passed in the right order and that transportation - internal or external - is performed between every manufacturing step while holding the constraints. Figure 1 shows the basic steps when defining the industrial system. The product breakdown determines the manufacturing steps to be performed. The associated order of production steps is then defined. A typical manufacturing and transportation path consists of repeating sequences of productions step, transport step, warehouse, transportation and the next production step. These will determine the according objective functions.

Although valid solutions have been generated, the variability of these solutions remains low, and they appear to be distant from any optimum. It is concluded that only a very small fraction of the solution space has been explored. This work is the continuation of former research work of the central R&T team of Airbus with the title *A Logic Programming approach to Global Logistics in a Co-Design Environment* [1] using the *clingo* ASP System to solve the optimization problem.

1.2. Problem Statement

The problem statement will be split into three subsections, the first one generalizing the specific problem of the thesis to known problems, the second one tackling the problem of selecting the right objective functions and according formulation of the

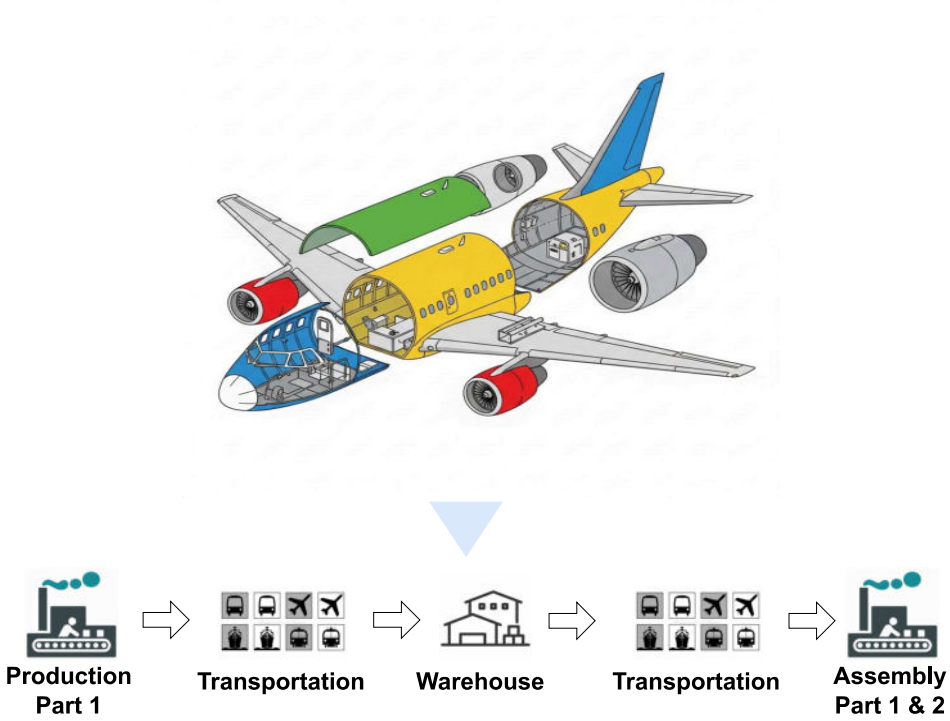


Figure 1: Exemplary aircraft breakdown and associated production and transportation sequence

optimization statements and the third one dealing with issues that arise with the traversal of the search space.
The detailed concepts behind the ASP features mentioned in this section will be subject to Section 2, the Theoretical Background.

1.2.1. Generalization of the Problem and Solution Space

The main problem faced in this thesis is the optimization of a *Multi-Echelon Production Network* with regards to some cost functions. A Multi-Echelon Production Network is organized in stages that must be passed in a predefined order [10].
To be more precise, this problem consists of two subproblems: First, to match parts to production sites in order to guarantee that each part is produced by at least two sites and each site produces at least one part (or the required share by site or country), and second, to decide on the transportation routes and according means among the production sites. For better visualization of the problems, the representation as a Multi-Echelon Production Network graph is helpful, which we will do below.

The first subproblem, where parts are matched with production locations, can be regarded as a *Set Multicover Problem (SMC)*. The production locations assigned with parts to be manufactured correspond to the nodes of the Multi-Echelon Production Network. The second subproblem is about selecting transportation means among the production sites, which completes the network with its edges. Optimization inside this network, e. g. minimizing transportation costs, corresponds to the *Shortest Path in a Directed Acyclic Graph Problem*. The details are described below.

Optimal Allocation of Part Production Across Sites
A set of manufacturing sites across the world is involved in the full manufacturing chain, all with their individual capacities to produce specific components. As sites can produce multiple parts and each part need to be produced by two sites for our specific Scenario, this leads to a many-to-many assignment problem. In this case, we deal with a special case of a Set Multicover Problem. A Set Multicover Cover Problem is to cover all elements of a universe N containing n elements at least a predefined number of times i with a minimum number of sets. It is a generalization of the *Set Cover Problem* with the main difference that elements of the universe may be covered multiple times [11].

To formalize the above problem, let there be a set $P = \{p_1, p_2, ..., p_n\}$ of parts to be produced and a set $S = \{s_1, s_2, ..., s_m\}$ of (production) sites. Each site $s \in S$ can produce a subset $P_s \subseteq P$ of all parts. Depending on the Sourcing Strategy, each part $p \in P$ must be produced by i_p sites (e. g. $i = 1$ for single sourcing, $i = 2$ for double sourcing etc). Each part p has a fixed workload w_p independent from the site where it is produced. And finally each site $s \in S$ has a dedicated cost factor c_s reflecting the specific manufacturing cost level of the location. Let x_{ps} be the decision variable with

$$x_{ps} = \begin{cases} 1 & \text{if part } p \text{ is produced at site } s \\ 0 & \text{otherwise} \end{cases}$$

The objective function is

$$min \sum_{s \in S} \sum_{p \in P} c_s \cdot w_p \cdot x_{ps}$$

accordingly and we have the following constraints:

- *Workshare*: each site must produce at least one part

$$\sum_{p \in P_s} x_{ps} \geq 1 \; \forall s \in S$$

- *Sourcing Strategy/Completeness*: each part p must be produced by at least i_p sites

$$\sum_{s \in S : p \in P_s} x_{ps} \geq i_p \; \forall p \in P$$

- *Eligibility*: parts can only be assigned to sites when they are able to produce the part

$$x_{ps} = 0 \text{ if } p \notin P_s$$

The size of this subproblem is determined by the given instances of production sites that can produce the number of parts to be produced. There are $n_{max} = 13$ production sites available for 34 parts to be produced. In average, every part can only be produced by $n = 10$ production sites. From these, $k = 2$ need to be selected for every part to comply with the Double Sourcing Strategy (will be explained in Section 1.3). This can be calculated with the binomial coefficient with the following number of possible combinations for every part [12]:

$$\binom{n}{k} = \frac{n!}{k!(n-k)!}$$

As we have overall $P = 34$ parts to be produced, the overall number of possible combinations is

$$\left(\frac{n!}{k!(n-k)!}\right)^P = 1.6E56$$

Optimal Routing Across Production Sites The second problem is finding the shortest path in a logistics network which can be regarded as a Multi-Echelon Production Network. It represents an industrial setup that is structured in manufacturing stages. Every stage must be passed downstream and for all stages there might be multiple sites to fulfill the task. It can typically be represented as a *Directed Acyclic Graph (DAG)* [13]. See an example of a Multi-Echelon Production Network with four echelons (Elementary Part Suppliers, Sub-Assembly Sites, Assembly Sites and Final Assembly Sites) in Figure 2. All need to be passed downstream in this order to produce a product.

Common challenges in industrial applications are the optimization of such a network with regard to some KPIs [2], e. g. production or transportation costs, or analyzing the robustness of the network against disruptions [14].

Let G be a DAG with $G = (V, E)$ with $u \in V$ and V is partitioned into l layers

$$V = V_1 \cup V_2 \cup ... \cup V_l$$

where each V_i (with $i \in \{1, \ldots, l\}$) is the set of nodes representing the possible production sites at echelon i. Edges $E \subseteq V \times V$ are directed and can only go from one level i to the next level $i + 1$. Each edge $(u_i, v_{i+1}) \in E$ has a cost $c_{tm}(u_i, v_{i+1})$ where tm is any transportation means that is available between the nodes u_i and v_{i+1}.

As there are multiple possibilities to select production sites per echelon, binary decision variables must be introduced for any edge:

$$x_{uv} = \begin{cases} 1 & \text{if } u \text{ and } v \text{ and } edge(u, v) \text{ is selected in the path} \\ 0 & \text{otherwise} \end{cases}$$

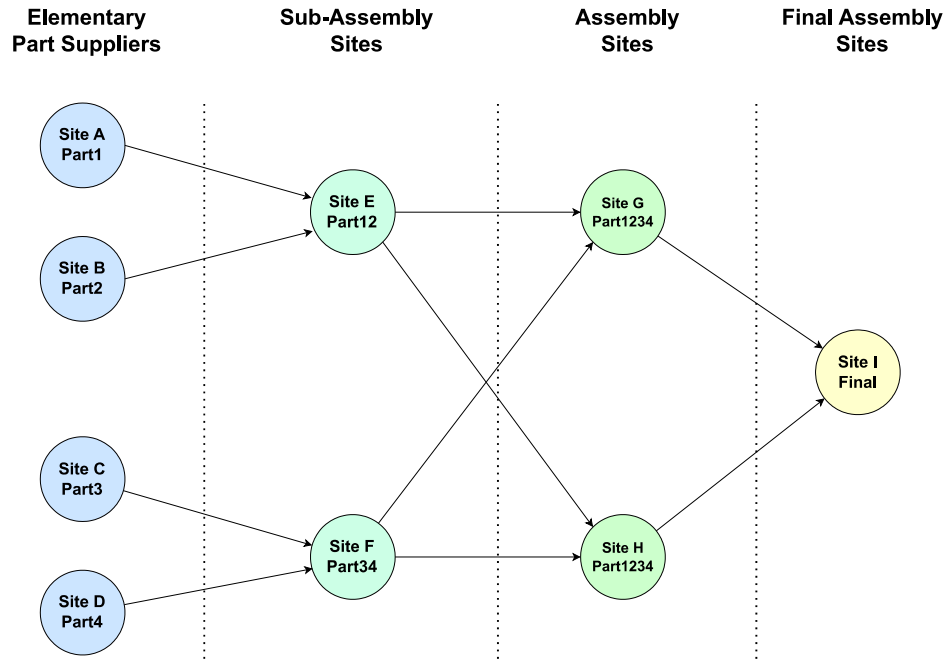


Figure 2: Principle of a Multi-Echelon Production Network for an aircraft

The optimization problem with regards to any cost c_{tm} of a transportation mean tm at the edge (u_i, v_{i+1}) can then be expressed by:

$$min \sum_{(u_i,v_{i+1})\in E} c_{tm}(u_i, v_{i+1})$$

Problem formulations that are expressed as shortest path problems in a DAG with vertices V and edges E can be solved in linear time with $\Theta(V+E)$, for instance with a topological sort method [12].
The size of this subproblem is determined by the transport routes which are depending on the selected production sites from the first subproblem, the available transport means for these routes, the part to be transported and potential warehouses that are used in between. There are $R = 132$ core routes (the connection between two production sites that need perform consecutive manufacturing steps) to be served for a Double Sourcing Strategy (see Section 1.3 for explanation of the Sourcing Strategy and see Section 3.2.2 for the explanation why a double Sourcing Strategy requires four times the routes compared to a Single Sourcing Strategy), $n_1 = 17$ warehouses and 30 transportation means available. In average, every route

and part can only be served by $n_2 = 10$ transport means. Between every route there is the option to have none, $k_1 = 0$ (which we call *Direct*), one, $k_1 = 1$ (called *Via 1*) or two , $k_1 = 2$ warehouses (called *Via 2*). See the Warehouse Scenario explained in Section 1.3 for detailed description. Selecting a warehouse to be used between two production sites will make two routes necessary instead of a single direct route. And two warehouse will lead to three routes instead of one. See Figure 3 for visualization. As a consequence the number of routes can increase up to three times of the 132 routes mentioned earlier resulting in $R_{max} = 394$ possible routes maximum for the Double Sourcing Strategy.

As for the first subproblem, this can be calculated with the binomial coefficient with the following number of possible combinations split into two key decisions, i.e. the selection of the number of warehouses for a route between two production sites and the selection of a transportation mean for every resulting route to be served.

For the selection of warehouses the solution space or number of possible combinations can be calculated for all 132 core routes with:

$$\left(\binom{n_1}{k_1} + \binom{n_1}{k_2} + \binom{n_1}{k_2}\right)^R = (1 + 17 + 136)^{132} = 5.6E288$$

And for the selection of one transportation mean for each of the worst case number of routes R_{max}:

$$\binom{n_2}{1}^{R_{max}} = 10^{394} = 10E394$$

Regarding the KPIs, the production costs are determined by the assignment of parts to production sites and therefore subject to the Set Multicover Problem. Transportation costs and CO_2 emissions on the other hand are subject to the routing problem as they can be represented and computed from the costs of the edges.

1.2.2. Related Problems

Typically the problem above comes with another well-known problem, the *Multi-Commodity Flow Problem*, which appears when multiple part types flow through a logistics network. This problem additionally considers capacity and time factors of the network to be optimized making it a *Minimum-Cost-Flow-Problem* [15] which is not subject to this thesis.

Due to the obligation that every partner must hold a share in the manufacturing process, a *Traveling Salesperson Problem (TSP)* component is also a characteristic of the overall problem. The main problems and constraints addressed in the TSP, finding the shortest path with passing every node, are already tackled in the problem

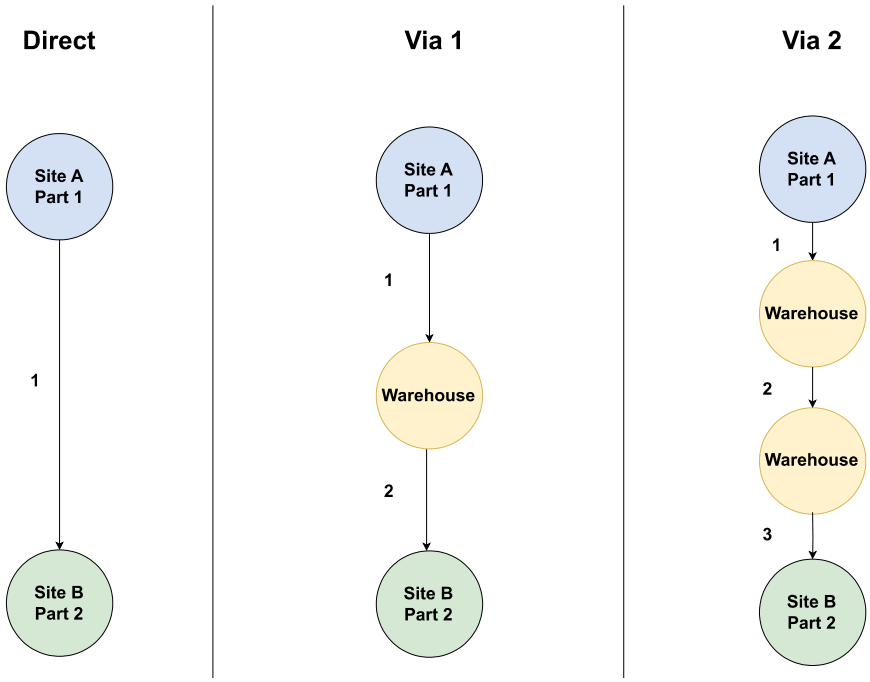


Figure 3: Impact of warehouses on the routing

generalization by the *Set Multicover Problem* with the constraint that every site must produce a part - ensuring that every node of the DAG is passed.

1.2.3. Objective Functions and Optimization Statements

The implementation of the initial research activity and the according answer sets are far away from an optimum with regards to a multi-objective function. An optimum is an answer set which optimizes the predefined KPIs in a multi-objective optimization formula. Figure 4 is visualizing the effects observed for a *Pareto* optimization indicating that solutions do not hit the Pareto frontier. A Pareto optimization aims at finding the best solution between two conflicting objective functions [2]. See the evolution of the KPIs in the scatter plot in Figure 4. It is not important to see the details of the axes, but rather the general evolution of the models and according KPIs.

Each dot represents a solution - answer set - of the logistics system. Each subplot is showing the according value for every answer set with regard to any combination of two KPIs out of the predefined ones. The iterations evolve over time from the top right corner of the plot to improved solutions in the direction of the bottom left cor-

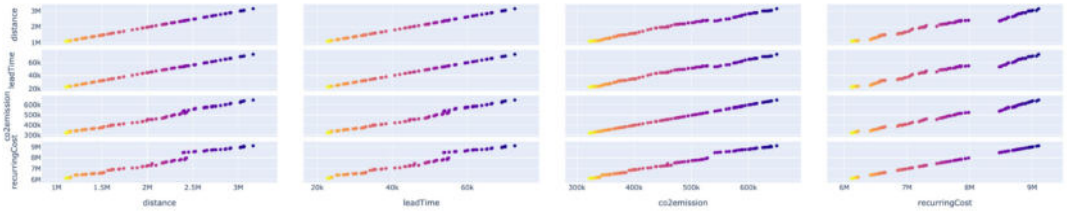


Figure 4: Multi-scatter plot of answer sets evolution showing all KPI combinations. Source: [1]

ner for every subplot without finding an optimum. The single dots representing answer sets are linearly moving until timeout. The process does not provide an optimum for any combination. An optimum in a Pareto diagram is normally indicated by the so-called *Pareto Front*. The set of Pareto optimal solutions in Pareto diagrams is supposed to form a front around the bottom left corner as both KPIs can not be improved simultaneously anymore [2]. The principle is shown in Figure 5.

A Pareto optimal solution *dominates* another answer set if the two resulting objective function values are both better than for the compared answer set. An example is shown in Figure 6. Answer set A dominates answer set B, but does not dominate answer set C. Pareto optimal solutions are not dominated by any other solution.

Apparently the Pareto Front is not reached for any combination of KPIs and the given predefined maximum number of models for the optimization.
It is acknowledged by the research team and confirmed by the industrial architects, who are the customers of the research work, that part of the chosen KPIs are not the right ones to be taken into account for architectonic trades and according decisions. The analysis of the solutions shows that variability of the selected predicates or atoms which build an answer set is low. The generated solutions cover a very small portion of the search space, which leads to the next problem: the limited traversal of the search space.

1.2.4. Search Tree Traversal

The problem encoding consists of a large number of facts and rules without negations. This leads to a positive propositional program which is *stratified* (see Section 2.1) once the program is grounded, which only takes a few seconds. This prevents from single atoms cyclic dependencies. All rules are *safe* (see Section 2.1). All together, this has a positive impact on the complexity of the program and the associated effort and run-times of the solver.
On the other hand, the absence of negations will result in the inference processes to not encounter many conflicts during solving while solvers like *clingo*, which use *Conflict-Driven Nogood Learning (CDNL)* techniques, may not be able to prune the search space effectively. As the search space is not pruned sufficiently, the optimizer does not succeed in converging for any combination of KPIs.

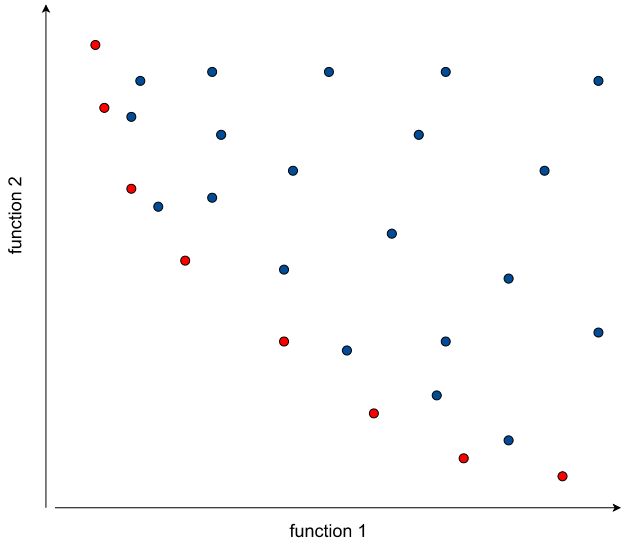


Figure 5: Pareto optimal answer sets in red forming a Pareto Front. Source: [2]

The static weighting of the optimization statements does lead to a unique search space traversal, missing opportunities to actively explore promising spaces.
The conclusion of the observations above is that partially the choice of the KPIs is not suitable for a Pareto optimization due to their linear dependencies. Additionally, the selected KPIs do not serve to properly perform the trades requested to enable decisions on architectural alternatives.
It is assumed that only a small amount of the search space is explored, which is suggested by the low variability of answer sets and the linear direction of the answer set evolution towards the theoretical optimum. The search space is not pruned sufficiently to enable convergence of the optimizer.

1.2.5. Hypotheses

As a consequence of the observations made and described in the previous sections, the following hypotheses are established:

H1 The static and independent formulation of the optimization statements does

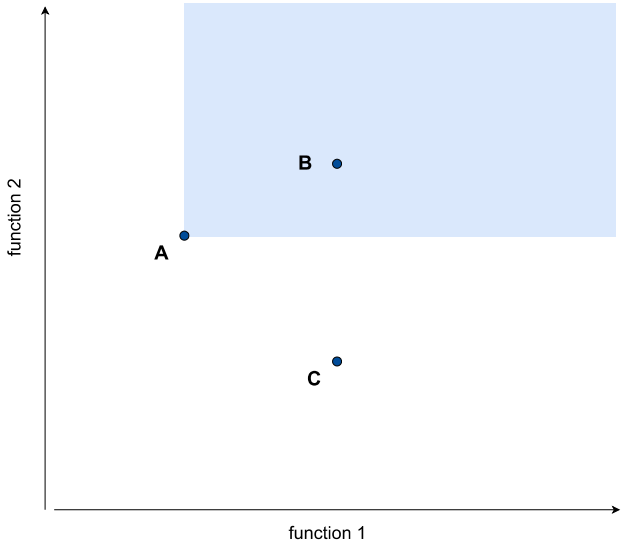


Figure 6: Three answer sets with different values for the two observed objective functions $function\ 1$ and $function\ 2$. Source: [2]

not enable a proper Pareto optimization and results in a unique convergence behavior of the optimizer missing diversity in the traversal of the search space

H2 The problem encoding does not support main mechanism of the solver, i.e. CDNL, and the search space is therefore not properly pruned to enable to find an optimum

H3 The solver is not actively guided through the search space and promising areas are not actively searched to increase both, variability of answer sets and the probability to find an optimum

1.3. Industrial System Scenarios and Constraints

The character of the industrial system to be optimized is strongly determined by some general strategies or assumptions concerning the supply chain and manufacturing setup which can be regarded as possible standard *Scenarios*. These Scenarios do have an impact on the topology of the industrial system that can significantly

change the systems properties. They might act as additional constraints or choices, potentially having an impact on the complexity of the system and the according problem formulation. The Scenarios to be considered in this work are *Sourcing Strategy* and *Warehouse Strategy*.
The Sourcing Strategy defines whether a production step has to be performed by minimum one or multiple sites to ensure redundancy in the system which might lead to more robustness of the industrial system. The strategy might be defined and applied depending on the production stage of the Multi-Echelon Production Network.
Warehouses contribute to the robustness of the network, too, but can also have an impact on reachability of production sites with certain transportation means, because these warehouses provide an opportunity to switch transportation means. It can be chosen to have no warehouse, one warehouse or two warehouses between two production steps.
Two important constraints need to be implemented: one to guarantee each site produces at least one part (*Completeness*) and one to make sure that every part is produced by at least one site (*Workshare*). The Workshare constraint might be refined with an interval of expected shares per site or country.
Batching considerations will not be tackled in this thesis. Batching refers to the possibility to transport several numbers of a part with one transportation mean depending on the available space. An even more advanced approach which is applied often in practice is the concept of *mixed batching*, i.e. the possibility to transport different sorts of parts in multiple numbers with one transportation mean [16]. Practically, batching will directly influence the number of necessary transports between two sites. In this thesis, a single-part batching is assumed, meaning that every transportation mean transports exactly one part.

1.4. Objectives

The objective of this thesis is to find solutions to the problem of identifying valid industrial and logistics setups with ASP for the next generation Airbus civil aircraft manufacturing system with Pareto optimal KPIs regarding the multi-objective functions.
A valid solution in our context is an answer set that ensures that all required manufacturing steps are passed in the right order and that transportation is performed between every manufacturing step. This finally results in the representation of a Multi-Echelon Production Network, which describes a manufacturing system that is arranged in stages or multiple layers with materials and products flowing downstream [10]. Figure 7 aims at visualizing such Multi-Echelon Production Network with different possibilities to produce specific parts (represented as nodes) and transportation links among them (represented as edges).
For the specific problem tackled in this thesis, 30 locations around the globe can act as production or warehouse sites with 34 parts to be produced. For the trans-

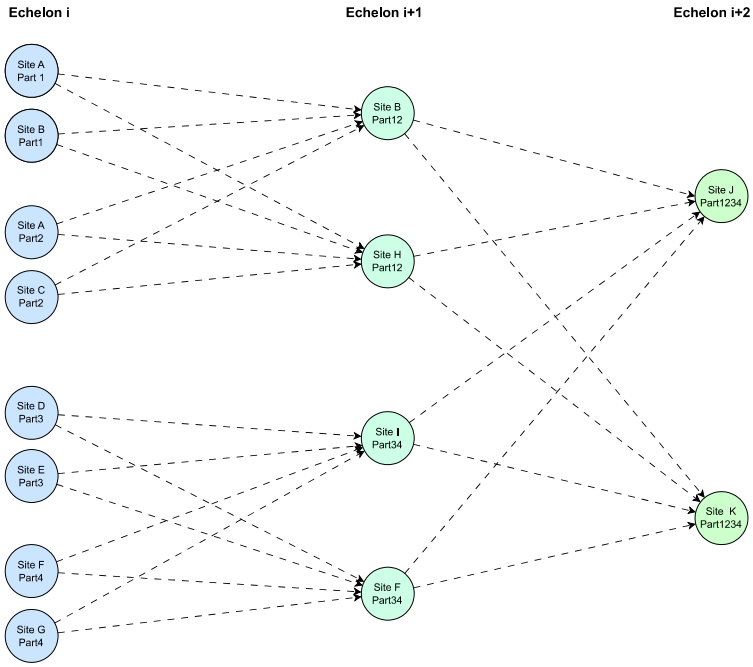


Figure 7: Part of a full Multi-Echelon Production Network

portation among the selected production and warehouse sites 34 different transport means are available. The valid solutions are constrained by the facts that not every production site can produce every part, not every transportation mean is available between every site and not every part can be transported with every transportation mean.

There must be locations that produce the parts needed for a full aircraft. For our problem we need two locations to produce a specific part, which is called *Double-Sourcing*. This is a significant element of the Scenario to be considered, i.e. the Sourcing Strategy. The Sourcing Strategy is defined per production stage of the *Multi-Echelon Production Network*.

The next major Scenario to be considered is the Warehouse Scenario, which requests that between two manufacturing steps there should be none, one or two warehouses.

Figure 7 visualizes a part of a full Multi-Echelon Production Network with three stages from echelon i to echelon i+2, which would be *Elementary Part*, *Sub-Assembly* and *Assembly* translated to our specific problem with only the last stage – *Final Assembly* – not shown in this Figure.

It represents all possible solutions to produce a part and potential transportation

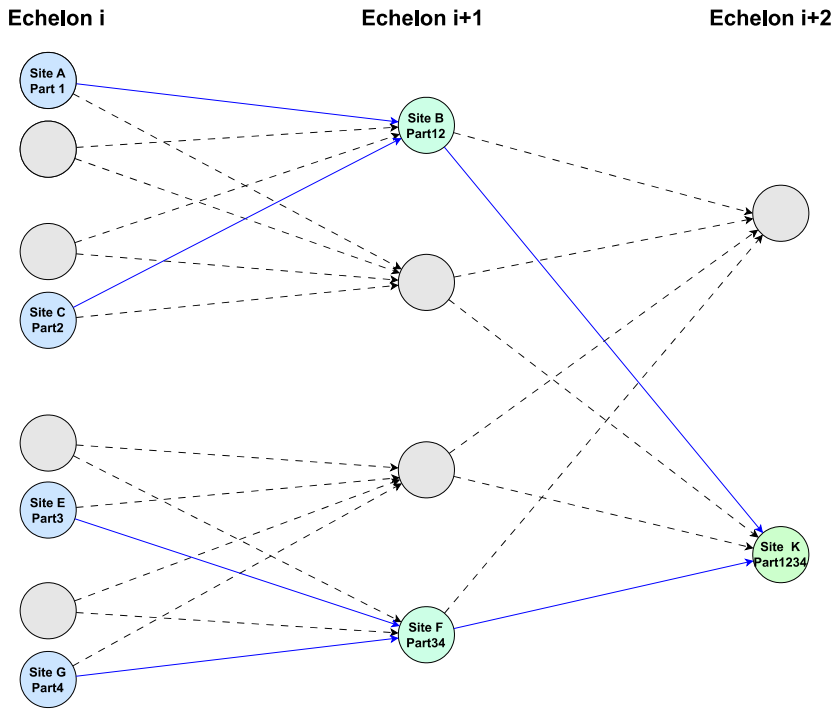


Figure 8: One selected solution with four different paths within the Multi-Echelon Production Network

in between production sites. In this case two suppliers are available per part which could produce the according part. The directed edges between the production sites indicate transportation, but are not visualizing all available transportation links and means in this figure. There are no warehouses between two production sites in this Scenario.
Figure 8 on the other hand shows one selected solution out of the possibilities seen earlier. The solution includes the choice of one production site per part in case of a *Single Sourcing Strategy* (one production site per part) and the selection of a transportation link and according mean in between two selected production sites of consecutive activities.

A simple single batching is assumed, meaning that only one single piece of a specific part is transported by the according transportation mean.
All together, the Double Sourcing Strategy, the Warehouse Strategy allowing to select zero to two warehouses and the single batching as defined above set the baseline configuration for this work.
The predefined KPIs are determining the objective functions which need to be opti-

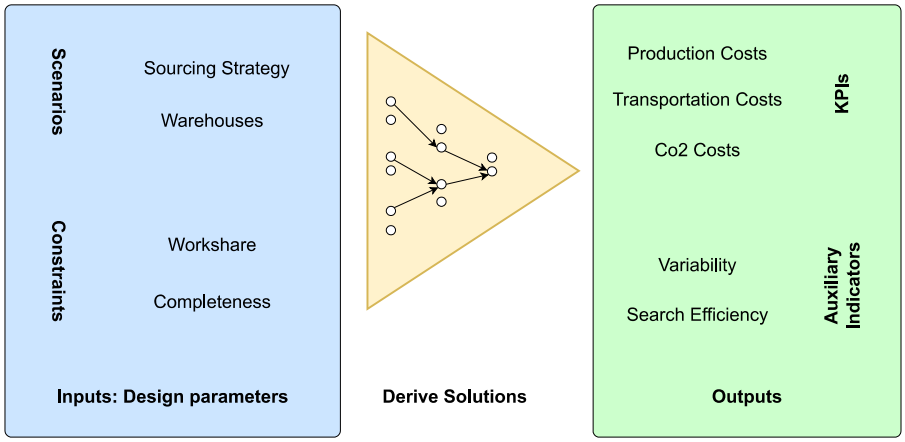


Figure 9: Design parameters and outputs

mized leading to a multi-objective optimization need.
The KPIs are *Production Costs*, *Transportation Costs* and CO_2 *emissions*. All are supposed to be (at least partially) conflicting and therefore subject to a Pareto optimization. The objective with the highest priority of this thesis is to find Pareto optimal solutions for some combinations of the defined KPIs as stated above.
Moreover, having high variability among answer sets is required for two reasons: first, to increase confidence that a large amount of the search space is explored and no better local optimum is missed, and second, to provide a variety of solutions that an industrial architect can actively explore to learn from unknown solutions and gain additional knowledge which can support an iterative and incremental product development.
Figure 9 gives an overview of the inputs and outputs of the experiments to be performed in this thesis. The inputs are considered design variables of the industrial system and the outputs are KPIs or auxiliary indicators to measure the quality of the solution system.

The optimization of the industrial system as above is a task performed in the context of systems engineering activities. Modeling and simulation capabilities as

developed and provided in this thesis aim at providing decision support for design trades [17]. In order to take decisions, it is not only required to obtain results for the KPIs, but also to have means to interpret the results properly. Therefore a visualization of the solutions and results is a desired, but not mandatory function to be delivered with this thesis.

The *Research Questions* derived from the problem statements and objective are:

Research Questions

1. Does a weighting of KPIs regarding the objective functions lead to more variability of the answer sets and support finding Pareto optimal solutions?
2. Do heuristic programming and *Weak Constraints* help find solutions in more promising areas as expected by domain knowledge and therefore faster generate improved results with regards to the optimization objectives?
3. Can a combined use of all above-mentioned methods lead to significantly better answer sets with regards to the optimization objectives, with more variability of answer sets and lower runtime?

1.5. Current State of Research

ASP has proven to deliver very good results when it comes to solving search problems in an industrial domain . There are successful industrial implementations mainly in the application areas of configuration, scheduling and planning. The declarative problem specifications are easy to understand and provide high expressiveness, making it a suitable paradigm [9]. Erdem, Gelfond and Leone [18] provide an overview of successful implementations, for instance in the optimization of the planning of Robot activities, where the work plan is considered optimal when the total cost of actions is minimal. Ostrowski, Schaub, Toletti and Wanko have presented an ASP based solution for the problem of train scheduling involving routing and scheduling optimization for a dense railway system.
As with imperative programs, ASP often faces a huge search space with many nondeterministic decisions. The *clingo* solution offers non-domain and domain heuristics to improve the performance. Non-domain heuristics have not demonstrated a significant increase of performance yet, therefore research is focusing more on introducing domain-specific heuristics into ASP solvers [9].
Combinatorial optimization problems are usually solved in a single shot [3] which leads to limitations in the exploration of the search space. Very few multi-shot approaches exist, most of them focus on dealing with changing logic programs. They tackle the grounding and solving steps with a flexible approach that enables the manipulation of logic programs with certain operations [19]. This approach can also help in reducing the ground programs to a minimal size which is very beneficial as

the grounding of very large problem instances is one of the major challenges in the use of ASP for industrial decision problems.
The use of multi-shot approaches is not officially published yet for the *Potassco* framework.
To the best of our knowledge there are no references found that investigate the capabilities of a system that combines active optimization statement weighting and domain-heuristics.

1.6. Outline

This subsection provides an overview and the structure of this thesis and how the sections interconnect.
The thesis starts with an *Abstract* which serves as a brief preview. It summarizes the motivation, key problems, methodologies, major findings and conclusions.

The *Theoretical Background* will review the theoretical foundations necessary to understand this work. It is divided as follows:

- *Logic Programming*: An introduction to the fundamental principles of logic programming
- *Answer Set Programming and Stable Model Semantics*: An explanation of answer set programming as a paradigm including the stable model semantics as an essential element of it
- *Potassco Toolset*: A description of the software tools and frameworks utilized for this work
- *Input Language*: An overview of the input language and the according constructs used for modeling and representation of the problem in this thesis

The section *Representation of the Industrial System and Optimization Problem* is essential to understand the logic program created in this work, which is one of the core elements of this thesis and translating the real-world problem into computational one. It consists of the following topics:

- *Issues of the Initial Program*: Explaining some major issues that caused the initial encoding to not produce the expected results or even errors
- *Representation of the Logistics System in a Logic Program*: Presenting the Problem Encoding with facts and rules
- *New Features and Expected Impact*: Introducing the new features to be tested and their expected impact on the objective functions and search tree traversal behavior

The *Methodology* section outlines the scientific approach adopted in this work. It explains:

- the specific research method and experimental design employed
- the evaluation criteria and procedures for analyzing the results

The outcomes of the experiments of the main and final phase are presented in the *Results* sections. They are supported by providing figures, tables and statistical data. Both Results sections are followed by their associated Discussions which will interpret the results in the context of the research questions and objectives. It critically reviews the findings and highlights the limitations of the work.
The thesis will close with the *Conclusion* which will summarize the main findings and reflect the main contributions. It will also include a suggestion for *Future Work*.

2. Theoretical Background

From an imperative programming perspective and in a simplified way, the industrial optimization problem described above could be characterized as finding the shortest path with constraints in a DAG where the nodes represent manufacturing sites and the edges transportation among them. The nodes hold the information about Production Costs while the edges provide the Transportation and CO_2 Costs. In this work we look at the problem from a declarative programming perspective using ASP as the programming methodology. It is rooted in the field of artificial intelligence and computational logic [20].

2.1. Logic Programming

Declarative programming characterizes the expected solution of a problem instead of precisely defining the steps to get there [20]. One kind of declarative programming is *Logic Programming* which is the foundation of the ASP solution used in this thesis. In Logic Programming an algorithm can be regarded as consisting of two components, the logic and the control. The logic expresses what the problem is while the control specifies the control flow. This separation does bring the advantage that the programmer can focus on specifying the logic by describing the problem. The control flow can be left to the Logic Programming system [21]. The implementation of the logic is based on *First-Order Logic* and uses logical formulas to represent knowledge in the form of rules. A Logic Program is a finite set of *rules*. Given a signature Σ, a *rule* r has the following form:

$$A \leftarrow A_1, ..., A_m, notA_{m+1}, ..., notA_n \tag{1}$$

Let $head(r) = A$ be the *head* and $body(r) = A_0, .., A_m, notA_{m+1}, ..., notA_n$ be the *body* of the *rule*. Negative atoms are denoted *not* which refers to *Negation as a Failure*, which will be explained later. Therefore the understanding of the formula is that the head of the rule must hold if any positive atom in the body is provably *true* and any negative atom can not be proven *true* and therefore is assumed *false* [22]. Van Gelder [23] defines the *body* of such *rule* as a *sub-goal*. Rules without a body or sub-goal are called *facts* and the '$\leftarrow$' can be omitted.

We define the set of positive atoms in the body of the rule above as

$$B(r)^+ = \{A_0, ..., A_m\}$$

and the set of negative atoms as

$$B(r)^- = \{not\ A_{m+1}, ..., not\ A_n\}$$

The handling of negation can be treated differently and it will be subject to Section 2.2.

Alternatively a rule as in Equation 1 can be written without negation in literal form. A literal can be a positive or negative atom. The rule looks as following accordingly:

$$A \leftarrow L_1, ..., L_n \tag{2}$$

When comparing Equation 2 with the previous style in Equation 1, L_0 corresponds with the head atom A, the literals $L_1, .., L_m$ with the positive atoms $A_1, ..., A_m$ and $L_{m+1}, ..., L_n$ with the negative atoms $A_{m+1}, ..., A_n$.

When a rule does not contain any variables, but only constants of the signature Σ, it is called a *ground rule*. The *ground program* of a program that includes variables can be obtained by replacing all variables in the program by all combinations of constants provided with the signature of the program. The *universe* of such kind of program is consequently built by the *ground terms* of the given signature, the *Herbrand-Universe*.
Take the following signature $\Sigma = \{Hamburg/0, Toulouse/0, Part1/0, Part2/0\}$. All elements are functions of arity 0, i.e. constants. The first two represent production locations while the latter two represent parts to be manufactured. Then the according Herbrand-Universe is given with $U = \{Hamburg, Toulouse, Part1, Part2\}$.
For the following program

Example 1.

$ispart(Part1)$
$isPart(Part2).$
$isProduction(Hamburg).$
$isProduction(Toulouse).$
$canProduce(Hamburg, Part1).$
$canProduce(Toulouse, Part2).$
$produces(X, Y) \leftarrow canProduce(X, Y), isProduction(X), isPart(Y).$

the according ground rules are

$isPart(Part1).$
$isPart(Part2).$
$isProduction(Hamburg).$
$isProduction(Toulouse).$
$canProduce(Hamburg, Part1).$
$canProduce(Hamburg, Part2).$
$canProduce(Toulouse, Part1).$
$canProduce(Toulouse, Part2).$
$produces(Hamburg, Part1) \leftarrow canProduce(Hamburg, Part1),$
$isProduction(Hamburg), isPart(Part1).$
$produces(Hamburg, Part2) \leftarrow canProduce(Hamburg, Part2),$
$isProduction(Hamburg), isPart(Part2).$
$produces(Toulouse, Part1) \leftarrow canProduce(Toulouse, Part1),$
$isProduction(Toulouse), isPart(Part1).$
$produces(Toulouse, Part2) \leftarrow canProduce(Toulouse, Part2),$
$isProduction(Toulouse), isPart(Part2).$

See in Example 1 that the variables in the last rule of the program in Example 1 are replaced by some - but not all - combination of the elements of the Herbrand-Universe. Note that the process of grounding already mentions the possible domains of the variables which is defined by the facts. It is obvious that only the constants which are defined as production sites can produce parts. The consideration of the domain of a variable while grounding can reduce the effort to generate the ground programs. This aspect of the definition of the domain of a variable is especially critical if the body of a rule contains negation. This issue will be addressed later with the concept of *safe* and *unsafe negation*.
Herbrand-Interpretations I are assignments of truth values $\{\bot, \top\}$ to a set X of ground atoms indicating valid solutions, also called *Herbrand-Models*, so that for each rule in P $I(r) = \top$ holds [19, 1]. For the definition of a valid solution or model, there are varying concepts depending on the selected semantics. In general, different interpretations I may be models of program P, but some solutions might be more favorable. One common problem that requires attention is the handling of incomplete information. These topics will be addressed in the Section 2.2 for further details.
A program consisting only of positive atoms and ground formulas is called a *Positive Propositional Program* [20].
Positive Propositional Programs are *stratified*, which ensures that an atom never depends on its own negation. *Non-stratified* programs on the other hand lead to cyclic dependencies putting additional complexity to the programs, valid interpretations

and the possible models [23].
Furthermore the concept of safe and unsafe negation in the body or sub-goal of a rule is important to avoid an undefined domain of a variable. Unsafe negation occurs when a negative sub-goal contains free variables. In order to avoid unsafe negation it is necessary that any variable in a negative sub-goal must appear in the head of the rule or in a positive sub-goal. Otherwise the undefined domain will lead to a free variable for the negative atom and can increase the search space significantly for grounding and proof of negation, practically reducing the performance of the solver [23].

Example 2.

$$unproducablePartAt(X,Y) \leftarrow part(X), not\ partProducableAt(X,Y).$$
$$unproducablePartAt(X,Y) \leftarrow part(X), productionSite(Y),$$
$$not\ partProducableAt(X,Y).$$

Example 2 gives two examples of formula - one safe and one unsafe. The first formula contains the free variable Y in a negative atom of its body without defining it with any positive atom while the second formula defines the variable Y with the preceding positive atom *productionSite(Y)* and is therefore safe.

2.2. Answer Set Programming (ASP) and Stable Model Semantics

ASP is a special paradigm of logic programming without a specific query that needs to be answered. Instead, ASP returns an *Answer Set* that includes all facts that can be derived from the rules of the program. This Answer Set is the *Stable Model* of the program [20]. The Stable Model semantics is one main characteristic of ASP. Stable Model semantics handles incomplete information under the *Closed World Assumption (CWA)* applying *Negation as a Failure*. This means that any ground atom that can not be derived from the program P is assumed false by default. When applied to a positive propositional program, the Stable Model is a *Minimal Model*, meaning that there exists no subset of the Answer Set which is also a Stable Model of the program [20]. Positive propositional programs have a unique *Minimal Herbrand Model* [24].
Most ASP solvers use a two-step approach as shown in Figure 10. First, a ground program of the existing program is created. This step is called *grounding*. Then the ASP solver computes the Stable Models of this propositional program - the Answers Sets [19, 3].

The grounding process requires the substitution of variables by constants which can lead to a combinatorial challenge with a complexity depending on the specific problem encoding. This can lead to the so-called *Grounding Bottleneck* [3]. Drivers of complexity can be the number of variables and elements of the universe, but also the use of certain input language constructs that reflect the combinatorial challenges of the problem. Possible constructs of the used input language leading to possibly

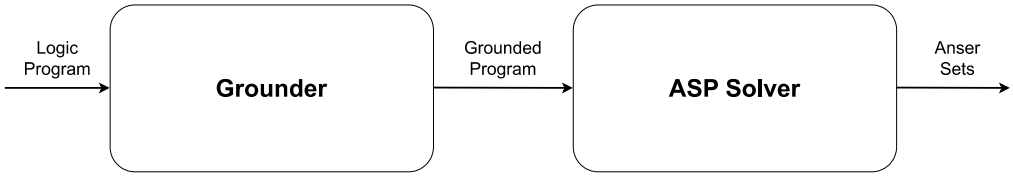


Figure 10: Typical two-step architecture of ASP solutions like *clingo*, Source: [3]

increased complexity will be presented later.
To tackle the possible Grounding Bottleneck, ASP systems might apply strategies to delay grounding of some parts of the program by incremental grounding of program segments or even preventing from grounding parts that are not needed at all for solving. The latter is presented by Faber and Friedrich in [3] and named *Lazy Grounding*.

For solving, *Boolean Satisfiability Problem (SAT)* solving techniques are applicable on propositional programs, looking for an assignment of the variables making the entire formula true. These SAT solving techniques are often applied as well for ASP solutions [22].
Common algorithms to solve the Boolean Satisfiability Problem are *Davies-Puttnam-Logemann-Loveland algorithm (DPLL)* and *Conflict-Driven Clause Learning (CDCL)*. Both algorithms use *Backtracking*, *Unit Propagation* and *Restarts* as key mechanisms to find solutions. Backtracking is a method in an algorithm to return to an earlier decision level when the recent search was unsuccessful. It is a mean to traverse the search tree. Unit propagation is a deterministic rule which looks for unit clauses - clauses with a single unassigned literal inside the logic program which can be assigned correctly with little effort. Restarts is a technique that can help improve the efficiency of a search by resetting the assignment of variables, but keeping learned clauses or *Nogoods*, which will be explained later. As a result the search tree will be traversed differently when starting from the beginning [25]. The foundations and further explanations of DPPL and the according use of Backtracking and Unit Propagation can be found in [26] and [27]. The concept of *conflicts*, i.e. assignments that can not lead to solutions, is important to both algorithms. The main difference between both algorithms is that for CDCL, after finding a conflict, not only Backtracking is performed, but adding a new learned clause to the program which represents the assignment which led to the conflict and therefore can not lead to a valid solution. Also, there is a difference in the application of Backtracking. DPLL performs chronological Backtracking while CDCL uses non-chronological Backtracking by recording the causes of conflict and using it later instead of immediately returning to the preceding decision level. This can prune the search space significantly [28, 29].
ASP is able to solve decision problems that are NP-hard and harder [3].

2.3. *Potassco* Toolset

The toolset used in this work is the *Potassco Collection* which includes *clingo*, a combination of a solver, which is called *clasp*, and a grounder named *gringo*. As a consequence, *clingo* follows a two-step approach to generate Answer Sets as described earlier. First, a grounding step is performed replacing all variables in the logic program with given instances and generating a finite propositional representation of the program [30]. Then the solver computes the stable models using *CDNL (Conflict-Driven Nogood Learning)* techniques which can be regarded as an evolution of the

CDCL algorithm described in the previous section [19].

2.4. Input Language of *clingo*

A logic program in *clingo* allows the use of facts and rules as described in Section 2.1. Additionally, so called **Integrity Constraints** are admitted. Integrity Constraints are rules without a head of the form:

$$:- A_1, ..., A_m, notA_{m+1}, ..., notA_n \tag{3}$$

An Integrity Constraint as in Equation 3 states that solution candidates which satisfy the body are not desired solutions, potentially describing constraints to the solution space.

clingo incorporates the utilization of *Weak Constraints* to provide guidance throughout the search process. Weak Constraints diverge from the binary nature of Integrity Constraints as they are not restricted to absolute fulfillment or negation. Instead, they operate with optimization statements that enable the evaluation of constraint adherence with regard to specific criteria. By this, the task of finding valid solutions - or Answer Sets - can be extended to finding optimal solutions [31].

Weak Constraints have the form:

$$:\sim \ L_1, ..., L_n \ [w@p, t_1, ...t_n] \tag{4}$$

The Weak Constraint is associated with a term tuple which can be seen in the squared brackets of Equation 4. The first element of the tuple represents the weight that is associated with the satisfied body. The $@p$ is an optional prioritization factor. Whenever the body of a Weak Constraint is satisfied, it contributes its term tuple $t_1, ..., t_n$ to a cost function. This cost function accumulates the weights w of all contributing tuples. Each tuple is only included once with its dedicated weight which refers to the **aggregate** property of the Weak Constraint. Note that when generating term tuples which contribute to some function, this is normally done to perform some operations on the results. This is demonstrated later when explaining optimization statements.

To properly understand the underlying concepts behind Weak Constraints it is important to see how aggregates are defined and expressed in *clingo*.

Before looking at some examples of aggregates and arithmetic functions applied on them, it is important to mention that typically the set of tuples in aggregates as described above are subject to conditions. Therefore *conditions* and *conditional literals* will be presented upfront.

A conditional literal has the form:

$$L_0 : L_1, ..., L_n \tag{5}$$

The ":" expresses a set notation and the literals behind it - $L_1, ...L_n$ - is the condition. L_0 holds whenever the condition holds. Conditions can have a meaning of a nested implication if we look at the next example:

$$A :- L_0 : L_1, L_2 \tag{6}$$

In the body of Equation 6, L_0 and L_1, L_2 practically act as head and body. L_0 holds if L_1 and L_2 hold.

Aggregates are expressive modeling constructs that allow forming values from groups of elements and conditions over terms. Together with arithmetic functions and comparisons available from build-in light theory solving capabilities, truth values can be derived from the aggregate's evaluation [31]. Aggregates can appear in the head and body of rules.
An aggregate in the body of a rule has the form:

$$s_1 \prec \alpha\{t_1 : L_1; ...t_n : L_n\} \prec s_2 \tag{7}$$

In Equation 7 the main part of the aggregate is inside the brackets including the term tuples $t_1, ..., t_n$ and the corresponding conditional literal tuples $L_1, ..., L_n$. α is some predefined built-in arithmetic function that can be applied to the term-tuples. *clingo* provides the functions *#count* (counts the number of term tuples), *#sum* (sums the weights of the term tuples), *#sum+* (sums only the positive weighted elements of the term tuples), *#min* (the minimum value weight of the term tuples) and *#max* (the maximum value of the term tuples).
The result of applying the function α to the term tuples can be compared to the terms s_1 and s_2 which act as lower and upper bounds. The comparison finally can be evaluated to *true* or *false* values.
The bounds as well as the function α can be omitted. In that case the default is the *#count* function, which is resulting in a so called *Cardinality Constraint*. There are no weights needed for the elements of the term tuples of a Cardinality Constraint. For the $\#count$ function a default value of 0 is set as a term in case no term is provided. It is important to remind that the term tuples are treated as set elements, so that the same term tuple is only counted once. Taking into account the said, the shortcut of the aggregate above would be:

$$\{t_1 : L_1; ...; t_n : L_n\}2 \tag{8}$$

The aggregate in Equation 8 can be understood as selecting maximum two elements t_i and t_j which fulfill the corresponding literal condition L_i.

Example 3.

$$1\{partProducedAt(Part, Location) : partProducableAt(Part, Location)\}2$$

Example 3 shows an example of a Cardinality Constraint with the $\#count$ function. The term tuple as well as the function declaration is therefore omitted. This aggregate will select minimum one or maximum two production locations for every part (*partProducedAt(Part,Location)* which fulfills the condition that the according location is capable of producing it (*partProducableAt(Part,Location)*).
Cardinality Constraints are also called *Choice Rules* as they enable to select a predefined number of elements out of a set. In case that no boundaries are set, all possible combination of the set can be selected.

When it comes to aggregates in the head of a rule, the literal to be derived needs to be added. Therefore a head aggregate has the form:

$$s_1 \prec \alpha\{t_1 : L_1 : L_1; ...; t_n : L_n : L_n\} \prec s_2$$

Optimization statements are a way to express several Weak Constraints. There are two optimization statements available, *#minimize* and *#maximize*.

A minimize statements for example has the form:

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

It represents the following n Weak Constraints:

$$\begin{aligned} :\sim\ & L_1.[w_1@p_1, t_1], ..., \\ :\sim\ & L_n.[w_n@p_n, t_n] \end{aligned}$$

A $\#maximize$ statement is treated the same way as a $\#minimize$ statement with inverse weights.
With the use of an Optimization statement with several Weak Constraints, a weighting or order of prioritization of term tuples can be performed. The definition of an *optimal Answer Set* can be adjusted to a specific preference this way.

The following example should showcase the use of a $\#minimize$ optimization statement applied on a $\#sum$ aggregate.

Take a transportation path between two production locations expressed this way as a fact: $path(Part, Location1, Location2, Distance)$. It tells that the Part can be transported form Location 1 to Location 2 with the according distance. If we aim to minimize the transportation distance of all necessary transportation between two consecutive production steps performed at the according locations, it can be expressed with the following statement:

Example 4.

$$\begin{aligned} &\#minimize\{(Distance, Part, Location1, Location2 : \\ &path(Part, Location1, Location2, Distance)\}. \end{aligned}$$

This statement will sum all the single distances reflected in the term *Distance* of ground atoms that hold the condition *path(Part,Location1,Location2,Distance)* and find the model that minimizes this value. Note, that the value to be minimized is the first element of the term tuple - in this case the term *Distance* from the term tuple *Distance, Part, Location1, Location2*.
The other elements of the term tuple are needed as some kind of identifier to address the aggregate property of the expression and to prevent from counting elements only once when it is not intended. The following example will demonstrate the potential issue:

Example 5.

$$\#minimize\{(Distance, Location1, Location2 : \\ path(Part, Location1, Location2, Distance)\}.$$

Imagine, the term *Part* is left out from the term tuple compared to Example 4. Then the two different ground atoms *path(Part1,Hamburg,Toulouse,2000)* and *path(Part5,Hamburg,Toulouse,2000)* do both contribute with the identical term tuple *2000,Hamburg,Toulouse* to the $\#sum$ function and therefore are only counted once due to the aggregate property of the construct. Assuming that these two are the only term tuples contributing to the $\#sum$ function, it evaluates to 2000 while in the previous example it evaluates to 4000. In this specific case, the additional term *Part* ensures that despite the fact that it is a transport between the same production locations it distinguishes them, because both represent two different transportation activities for different parts at possibly different stages of the production process.

2.4.1. The Grounder *gringo*

The Grounder used in the *Potassco* framework is called *gringo*. It is applying incremental grounding by gradually increasing the atom base. Facts and rules that can be derived independently can be grounded immediately while others are on hold until their conditions are met. This can have a positive impact on grounding efficiency and runtime overall.

2.4.2. The Solver *clasp*

The SAT solver used in the *Potassco* framework is *clasp*. The solving mechanisms in *clasp* are based on algorithms such as DPLL and CDCL including Unit Propagation, Backtracking and conflict-driven learning techniques which guide the search tree traversal and prune the search space. *clasp* is an evolution of the previously mentioned algorithms with the concept that all inferences in ASP solving can be regarded as Unit Propagation of so-called *Nogoods* [22]. Nogoods are local assignments that can not be extended to a solution. Formally, a Nogood is a set of literals $\{\delta_1, ..., \delta n\}$ defining an assignment that is unintended as it can not lead to a solution. A Nogood is *violated* by an assignment A if $\{\delta_1, ..., \delta n\} \subseteq A$. Otherwise, if

$\{\delta_1, ..., \delta n\} \not\subseteq A$, A is a solution for a set of Nogoods [22].
When it comes to heuristic decisions to be made, clasp uses the *Variable State Independent Decaying Sum (VSIDS)* heuristic as its default [31]. VSIDS is a heuristic that selects the ground literals depending on their values inside the (local) program. The value is resulting from the literal's appearance in recent conflicts. Furthermore it uses a decaying sum mechanism to adjust the scores or values dynamically over time. For further details consult [32].

One important feature of *clasp* is the possibility to use *Domain-specific Heuristics* into ASP solving which complements the solver's default heuristics. This enables to guide the search through non-determinism with experts knowledge and explore promising zones of the search space first, potentially leading to a more efficient search [31].

Heuristics are represented the following way:

$$\#heuristic\ A_0 : L_1, ..., L_n\ [w@p, m]$$

The # symbol is used to signal meta-statements in *clingo*. A_0 is an atom, $L_0, ..., L_n$ is the body of a rule, and w, p and m inside the squared brackets are terms. Similar as for Weak Constraints, w represents a weight, p is an optional prioritization and m is a modifier representing different types of heuristic information. Six different types for modifier m are available in *clingo*: *sign*, *level*, *true*, *init* and *factor*.
The heuristic modifier *sign* gives guidance for the assignment of the truth value when the solver has to decide. By default the sign value is 0. If the sign value - represented by w above - is set greater than 0, then the solver will assign the value of the according atom to *true*. If the value is less than 0, it will be set to *false*. These decisions will not impact the number of Answer Sets, but only the order in which they will be identified and listed. To understand this concept, the modifier *level* will be introduced as well and an example will demonstrate the impact on according Answer Sets. The heuristic modifier *level* puts some kind of importance to the atoms. In the Example 6, the solver is asked to assign two atoms with the truth value *true* first. The Integrity Constraint disallows to have both true at the same time. So, the modifier *level* determines which atom to set *true* first, i.e. *transportMean(truck)* in this case:

Example 6.

$$\begin{aligned}
&\#heuristic\, transportMean(truck).\ [1, sign]\\
&\#heuristic\, transportMean(aircraft).\ [1, sign]\\
&\#heuristic\, transportMean(truck).\ [10, level]\\
&\{transportMean(truck); transportMean(aircraft)\}\\
&:- transportMean(truck), transportMean(aircraft).
\end{aligned} \tag{9}$$

Example 6 will result in providing the Answer Set $\{transportMean(truck)\}$ first as it is asked to be set to *true* due to its *sign* value of 1 and its higher *level* value than

transportMean(aircraft). The next Answer Set is $\{transportMean(aircraft)\}$ and the final Answer Set is the empty set. The Answer Set with both atoms is disallowed due to the Integrity Constraint.
The modofier *true* combines the modifiers *sign* and *level*. The detailed explanations of the other modifiers will be left out here as they will not be used for this thesis. The interested reader is referred to [31].

clingo offers - among others - a vast set of additional features, including advanced constraint and theory solving capabilities, optimization and hybrid solving techniques. Some of these capabilities are still in the research stage [31].

With the Input Language explained, the next section will describe, how a Logistics System can be modeled with ASP.

3. Representation of the Logistics System and the Optimization Problem

This section will present how an industrial and logistics system – including the optimization problem and constraints – can be represented in a logic program with its according facts and rules. This program is considered to be the *baseline* program.
Significant issues that are resolved with the new baseline and the according changes of the encoding will be highlighted next.
This chapter will end with the features to be tested and their according formulations in ASP.

3.1. Representation of the Logistics System

As common in logic programming and ASP, the program can be split into facts and rules and structured in a *Guess-and-Check* architecture [33]. Guess-and-Check is a structure which aims at finding solutions by first generating solution candidates (*Guess*) and then filtering these by eliminating those that do not meet constraints or conditions (*Check*).
The baseline program including facts and rules is defined as the baseline configuration labeled with **(baseline)**.
Find an overview of the facts, i.e. 15 predicates, their descriptions and the according labels for further referencing in Table 1. Overall the logic program consists of more than 30.000 instances of facts.

The predicates *productionLocation/1* and *warehouseLocation/1* simply define the domain of existing production and warehouse locations.
costLocation/2 provides the relative cost of working units per location and is required to calculate the Production Costs per part at a specific production location.
country/1 defines the domain of countries where the production and warehouse locations are belonging, while *locatedIn/2* establishes the relation between a location and the country.
part/1 defines the domain of parts to be produced in a dedicated production step and *valueAdded/2* the according value or required workload to be manufactured independent from the production location. *partProduceableAt/2* determines which part can be produced at which production site.
productionPlan/2 provides the production sequence, telling which part is required for another part as an input.
transportMean/1 sets the domain of available transportation means.
canBeTransportedFromTo/8 is the most complex fact providing all available transportation links including the according properties of the link. There are more than 29.0000 facts of this predicate.

The rules to describe a logistics system are explained next.

Fact	Description	Label
$productionLocation(aProduction)$	aProduction is a Location that can produce a part	**(prodLoc)**
$warehouseLoc(aWarehouse)$	aWarehouse is a Warehouse that can store and repack parts	**(wareLoc)**
$costLocation(aProduction, 120)$	aProduction has a cost factor of 120 per unit	**(costLoc)**
$country(germany)$	Germany is a country	**(country)**
$locatedIn(aProduction, italy)$	aProduction is located in Italy	**(locIn)**
$part(part1)$	part1 is a part to be produced	**(part)**
$valueAdded(part1, 30)$	part1 has a value or workload of 30 units	**(valAdd)**
$partProduceableAt(part1, aProduction)$	part1 can be produced at production location aProduction	**(pProdceableAt)**
$productionPlan(part12, part1)$	part1 is needed for production of part 12 and needs to be produced before part12	**(prodPlan)**
$transportMean(aircraft)$	an aircraft is a tranportation mean	**(transpMean)**
$canBeTransportedFromTo(Hamburg, Toulouse, part2, truck, 120, 70, 23, 1200)$	part2 can be transported from Hamburg to Toulouse by truck with distance 120, lead-time 70, Co2 emission 23 and transport cost 1200	**(canBeTransp)**
$elmentaryPart(part1)$	part1 is an elementary part and therefore belonging to the first stage of the multi echelon network	**(isElementary)**
$subAssembly(part12)$	part12 is a sub-assembly part and therefore belonging to the second stage of the multi echelon network	**(isSubAss)**
$assembly(part1234)$	part1234 is an assembly part and therefore belonging belonging to the third stage of the multi echelon network	**(isAss)**
$finalProduct(part12345678)$	part12345678 is an elementary part and therefore belonging to the last stage of the multi echelon network	**(isFinal)**

Table 1: Overview of all facts of the logic program.

Find the overview of the baseline program rules – referred to as the baseline configuration with the label **(baseline)** in Table 2. The following explanations are providing the according label of the rules which are shown in the *label* column in the table.
At every production location there is an *intrasite* transportation available that can transport any part within a location with distance zero. As a consequence, every part can be transported intrasite: **(intraSite)**.
It is always possible to transport a part directly from one production location to another if it can be transported at all: **(ruleDirect)**.
Another option is that the part goes via a Hub, which is a warehouse: **(ruleVia1)**. Or the part can even be transported via two hubs, which significantly increases accessibility: **(ruleVia2)**.

Finally, the solution candidates, i.e. the valid logistics system to be derived, are represented by two predicates, *path/8* and *partProducedAt/2*. Therefore these rules reflect the *guess* part of the baseline program as described earlier.
partProducedAt/2 describes which part is produced at which site. Depending on the Sourcing Strategy to be selected for the according echelon, it is expressed as a Choice Rule/Cardinality Constraint. It is the implementation of the logical element which represents the Set Multicover Problem as described in Section 1.2.1.
It must be possible to define the Sourcing Strategy per production stage considering the multi-echelon property of the production and logistic system. Therefore, the Choice Rules are implemented for every level of the Multi-Echelon Production

Network: **(choiceLv1)**, **(choiceLv2)**, **(choiceLv3)** and **(choiceLv4)**. This ensures that every part is produced at at least at the number of sites that was set as the lower bound. The baseline value is 2 for all lower and upper bounds resulting in a Double Sourcing request for all production stages.
One important constraint is the implementation of the Workshare constraint ensuring that every production location must at least produce one part. In the *(baseline)* configuration, this is implemented as an Integrity Constraint: **(workshare-IC)**.

The second solution candidate predicate, *path/8*, expresses a transportation path between two production sites that needs to be performed with a specific transportation mean and the according attributes. As this is part of the solution to be derived, it is expressed as a Choice Rule/Cardinality Constraint in the head of the rule. A valid path can be either direct, via one hub or via two hubs. One out of of several possible paths taking into account the correct production sequence, including the choice of an available transportation mean, has to be selected: **(choicePath)**.
Finally, the optimization statements used for the objective functions - or KPIs - can be found in the group of rules **(optim-baseline)**.
The first optimization statement aims at minimizing the Production Costs, the second the Transportation Costs and the last one the CO_2 emissions. As no priorities are set, *clingo* does prioritize in declining order of appearance. This topic will be detailed later in section 3.2.1.

3.2. Major Issues of the initial Research Work

This thesis builds upon the initial research work done by the central R&T team and it is worth to specifically look at some major shortcomings or issues in order to understand the rationales behind some modified or new features implemented for the baseline or to be tested in the experiments.

3.2.1. Optimization Statements

The initial problem encoding which is part of the bassline configuration is facing general issues with the formulation of the optimization problem, making it unsuitable for a Pareto optimization and resulting in wrong results for some KPIs. This needed to be resolved together with logical mistakes encountered.
The attempt for a Pareto optimization for the KPIs as mentioned above was realized by three independent $\#minimize$ statements as visible in **(optim-baseline)**.

clingo will prioritize by the order of appearance by default, meaning that the first optimization statement will be optimized as first priority. Once this term is optimized, the following optimization statements will be taken into account to break the tie of the previous optimization [31]. As a conclusion, this might lead to a Pareto optimization with a behavior that prioritizes accordingly and potentially generates answer sets with unbalanced KPIs, meaning that the KPI with the highest priority will have good values while the lower priority KPIs might be weak. 1.2.3.

Program excluding facts	Description	Label
transportMeanAtSite(P, $intrasiteTransport$) $\leftarrow$ $productionLocation$(P)	intrasite transport available at every site	**(intraSite)**
canTransport($intrasiteTransport$, Part) $\leftarrow$ $part$(Part)	intrasite can transport every part	
direct(Part, From, To, TM, D, LT, CO2, TC) $\leftarrow$ canTransport(From, To, Part, TM, D, LT, CO2, TC)	transport part directly	**(ruleDirect)**
via1(Part, From, (Via1, To), (TM1, TM2), D, LT, CO2, TC) $\leftarrow$ canTransport(From, Via1, Part, TM1, D1, LT1, CO21, TC1), From! = To, productionLoc(From), productionLoc(To), warehouseLoc(Via1), canTransport(Via1, To, Part, TM2, D2, LT2, CO22, TC2), D = D1 + D2, LT = LT1 + LT2, CO2 = CO21 + CO22, TC = TC1 + TC2	via one warehouse or	(**ruleVia1**)
via2(Part, From, ((Via1, Via2), To), (TM1, TM2, TM3), D, LT, CO2, TC) $\leftarrow$ canTransport(From, Via1, Part, TM1, D1), From! = To, productionLoc(From), warehouseLoc(Via1), canTransport(Via1, Via2, Part, TM2, D2), Via1! = Via2, productionLoc(To), warehouseLoc(Via2), canTransport(Via2, To, Part, TM3, D3), D = D1 + D2 + D3, LT = LT1 + LT2 + LT3, CO2 = CO21 + CO22 + CO23, TC = TC1 + TC2 + TC3	via two warehouses	(**ruleVia2**)
$\leftarrow$ productionLoc(P), not produced(P)	at least one part per site	**(workshare-IC)**
produced(P) $\leftarrow$ producedAt(_, P)	true if P produces a part	
1{path(Part, From, To, TM, D, LT, CO2, TC) : *direct*(From, To, Part, TM, D, LT, CO2, TC); path(From, Part, (Via1, To), (TM1, TM2), D, LT, CO2, TC) : via1(Part, From, (Via1, To), (TM1, TM2), D, LT, CO2, TC), path(Part, From, ((Via1, Via2), To), (TM1, TM2, TM3), D, LT, CO2, TC),) : via2(Part, From, ((Via1, Via2), To), (TM1, TM2, TM3), D, LT, CO2, TC),)}1 $\leftarrow$ producedAt(Part, From), productionPlan(Super, Part), producedAt(Super, To)	direct path, or path via one site, or path via two sites	(**choicePath**)
e{producedAt(Part, P) : produceableAt(Part, P)}e $\leftarrow$ elementary(Part)	depending on its echelon	**(choiceLv1)**
s{producedAt(Part, P) : produceableAt(Part, P)}s $\leftarrow$ subassembly(Part)	each part needs to be	**(choiceLv2)**
a{producedAt(Part, P) : produceableAt(Part, P)}a $\leftarrow$ assembly(Part)	produced at e, s, a or f	**(choiceLv3)**
f{producedAt(Part, P) : produceableAt(Part, P)}f $\leftarrow$ finalProduct(Part)	sites, with $e, s, a, f \in \mathbb{N}$	**(choiceLv4)**
#**minimize**{(Value $\cdot$ Cost)/$e_1 \cdot e_2$, Part, P : producedAt(Part, P), valAdded(Part, Value), costLoc(P, Cost)}	minimize Production Costs	**(optim-baseline)**
#**minimize**{TC, Part, From, To, TM : path(Part, From, To, TM, D, LT, CO2, TC)}	minimize Transportation Costs	
#**minimize**{CO2, Part, From, To, TM : path(Part, From, To, TM, D, LT, CO2, TC)}	minimize CO_2 emissions	

Table 2: Relevant rules and optimization statements of the (baseline) program excluding facts.

3.2.2. Usage of Aggregates

A wrong usage of aggregates in the optimization statements as described in Example 4 for Transportation and CO_2 Costs was detected resulting in values that are too high for Multiple Sourcing Strategies. The starting point of the issue is the body B of rule **(choicePath)** in Table 2.

While for a Single Sourcing Strategy there is only one production location selected per part and therefore *partProducedAt(Part,From)* would only evaluate to *True* for one combination of part and production location, it will be *True* for multiple production locations in the case of a Multiple Sourcing Strategy. The same applies to *partProducedAt(Part,To)*. This results in several combinations per part evaluating to *True* in the body $B(r)$ of this rule and ultimately in selecting multiple valid paths as requested by the head of the rule $H($**(choicePath)**$)$.

All these valid paths are taken into account for the $\#minimize$ statements, which are aggregates and therefore sum all elements and their contributing terms as if all parts are taking every single path (see section 2.4). In fact, the overall number of a specific part would be distributed among all existing possible paths. The number of possible paths n_{paths} can be determined by the number of production sites for a dedicated predecessor part $nbSites_{pred}$ and the number of sites for the successor part $nbSites_{succ}$ with:

$$n_{paths} = nbSites_{pred} \cdot nbSites_{succ}$$

See the following example for two parts A and B with A needed for production of B: In the case of single sourcing, the body of the rule **(choicePath)** only evaluates to *True* for

Example 7.

$$\leftarrow partProducedAt(Part1, SiteA), productionPlan(Part2, Part1), partProducedAt(Part2, SiteC)$$

while for the example of a Double Sourcing for *part1* and Triple sourcing for *part2* all of the following bodies would evaluate to *True* and thus derive the Choice Rule **(choicePath)** to select solution candidates with *path/8*.

$$\leftarrow partProducedAt(Part1, SiteA), productionPlan(Part2, Part1), partProducedAt(Part2, SiteC).$$
$$\leftarrow partProducedAt(Part1, SiteA), productionPlan(Part2, Part1), partProducedAt(Part2, SiteD)$$
$$\leftarrow partProducedAt(Part1, SiteA), productionPlan(Part2, Part1), partProducedAt(Part2, SiteE).$$
$$\leftarrow partProducedAt(Part1, SiteB), productionPlan(Part2, Part1), partProducedAt(Part2, SiteC).$$
$$\leftarrow partProducedAt(Part1, SiteB), productionPlan(Part2, Part1), partProducedAt(Part2, SiteD).$$
$$\leftarrow partProducedAt(Part1, SiteB), productionPlan(Part2, Part1), partProducedAt(Part2, SiteE).$$

This example is reflected in Figure 11. For Single Sourcing, only one path can exist through which the full flow of parts is guided. For the Multiple Sourcing with n paths, the fraction of parts flow is only $\frac{1}{n}$ for each existing path assuming that the distribution among the paths is equal.

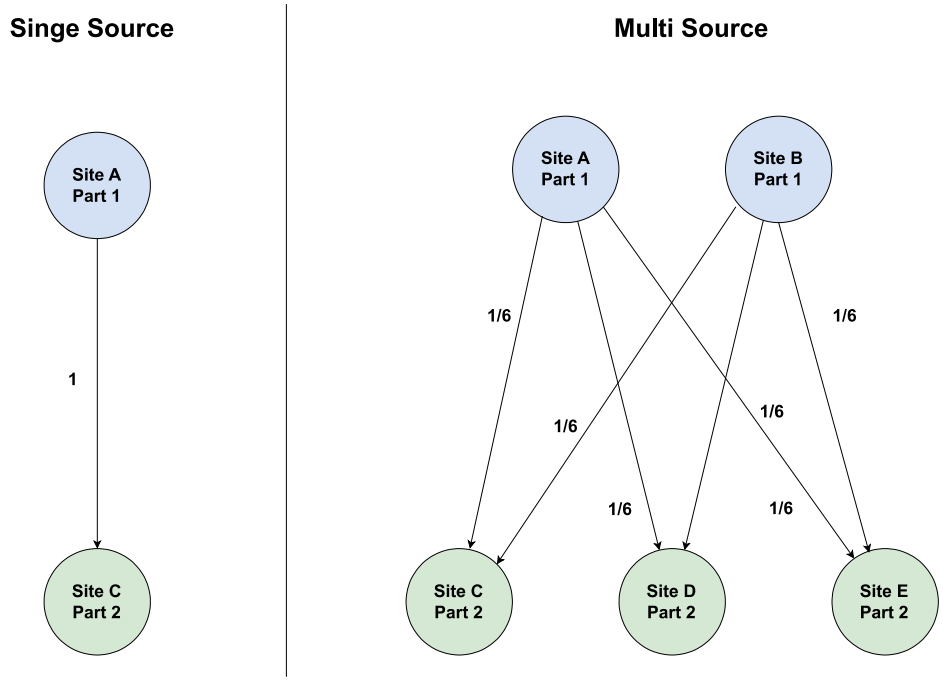


Figure 11: Single-Sourcing vs Multi-Sourcing: Multiple paths

3.3. Overview of New Features and Expected Impact

Remember that the complexity of the problem is depending on the Scenarios. The features to be tested will be tested on the **(baseline)** Scenarios. The baseline Scenario is the Double Sourcing Strategy and the Warehouse option, which gives the choice between no, one or two warehouse. The baseline configuration are all facts and rules as presented in Tables 1 and 2. An overview of all new features to be tested is shown in Table 3 with the according references of each rule or group of rules in the *label* column.
The following description of the new features to be tested is split into two groups depending on the expected main impact, i.e. on the KPIs or the Search Tree Traversal.

3.3.1. Main Objective Functions

Different order of weightings/priorities of the main objective functions will be tested as in rules **(optim-1P-2C-3T)**, **(optim-1T-2P-3C)**, **(optim-1T-2C-3P)**, **(optim-1C-2P-3T)**, **(optim-1C-2T-3P)** and **(optim-comb-Obj)**. Except for the last one, which will replace the two first $\#minimize$ statements, all optimization statements are part of the according configuration, but only with changing order of priorities as indicated by the naming. This is expected to have a direct quantitative impact on the KPIs. The active weighting of the optimization statements should also increase *Variability* of the answer sets by optimizing in different directions. A combined multi-objective function as in rule **(optim-comb-Obj)** will be tested to provoke different traversal behavior of the optimizer on its convergence paths tackling hypothesis H1. A combined function is an indirect mean to optimize two KPIs with the same priority as long as both values are normalized. This is expected to result in a different behavior of the solver than with independent optimization statements.

3.3.2. Search Tree Traversal

The active exploration of the search space is targeting to enable to prune the search space effectively and efficiently and by that increasing the probability to converge to an optimum. Only by steadily increasing the number of identified Nogoods, the search space can be reduced to a manageable size [19]. Active traversal management can also increase variability of the solutions and reduce runtime to find an optimum.
For the purpose of search space pruning, the use of Integrity and Weak Constraints to identify undesired solutions can be useful means. These are expected to increase the number of detected Nogoods and actively trigger the main mechanism of the solver and tackle hypothesis H2. Undesired solutions might refer to practical constraints concerning the industrial setup, for instance, that some work packages should be certainly assigned to some production sites.

On the other hand, implementation of domain knowledge by the means of Integrity or Weak Constraints might have negative impacts by excluding or penalizing solutions that could be mathematically optima, but are not desired solutions from domain experts view. Runtime until convergence may increase as well when mathematically better solutions are excluded or penalized.
The following group of Integrity Constraints will be tested as a set: **(minVal-IC)**. These Integrity Constraints ensure that minimum workshares are allocated to certain production sites making results more valid for the end-user and expected to prune the search-space. But it can also lead to excluding mathematically better or even optimal solutions and therefore produce worse KPIs and longer runtime.
Weak Constraints will be tested in two ways. One Weak Constraint will be replacing the **(workshare–IC)** rule from the **(baseline)** configuration: **(nb–Sites–WC)**. The weight w will be set higher than for all other optimization statements, so that over time every site will be allocated with a part.
Second approach is to test Weak Constraints as domain knowledge representations and compare them with domain heuristics implementations.
The strategic use of Weak Constraints as described in Section 2.4 may guide the solver and reduce the runtime. As Weak Constraints are light versions of Integrity Constraints, this might partially ease hypothesis H3 and prune the search space while also guiding the solver actively. The guided search into more favorable areas and rejecting solutions that are not valid or optimal is supposed to prune the search space and reduce runtime required to find a (local) optimum.
The Weak Constraints to be tested are **(min-hubs-on-locations)**, **(min-prefer-trucks)**, **(min-prefer-certain-locations)** and **(min-back-forth)**.

The implementation of domain heuristics is an alternative way of domain knowledge implementation. When applying a domain heuristic, it replaces the default VSIDS heuristics for the concerned predicates and will set according atoms to *True* when deciding on the atom. Therefore models including the preset domain heuristics will be found earlier.
The domain heuristics to be tested are supposed to be counterparts of the Weak Constraints mentioned above: **(heu-hubs-on-locations)**, **(heu-prefer-trucks)**, **(heu-prefer-certain-locations)** and **(heu-back-forth)**.
The Weak Constraints and domain heuristics aim to represent secured or assumed knowledge about the mathematical optimum with regards to a specific objective and are expected to find better KPIs faster for the addressed KPI, but can have adverse effects on another KPI.
(heu-prefer-certain-locations) and **(min-prefer-certain-locations)** are targeting to find better solutions quicker for the Production Costs. The optimization of Production Costs can be facilitated by guiding the search through preferred production sites, i. e. sites where the Production Costs are the lowest. Therefore this is also expected to have a positive effect on the runtime. It can lead to lower performance regarding the other KPIs though.

Program excluding facts	Description	Label
#**heuristic** producedAt(Part, P) : produceableAt(Part, P). $[w, true]$	prefer to produce part at site P	**(heu-prefer-certain-locs)**
#**heuristic** producedAt(Part, P) : producedAt(Part, P), productionPlan(Super, Part), subassembly(Part), produceableAt(Part, P). $[w, true]$	prefer to produce subassemblies at same site as assemblies P	**(heu-hubs-on-location)**
#**heuristic** path(Part, From, To, TM, D) : truck(TM). $[w, true]$	prefer trucks as transportation mean	**(heu-prefer-trucks)**
#**heuristic** producedAt(Super, P2) : productionPlan(SuperSuper, Super), productionPlan(Super, Part), producedAt(Part, P), producedAt(Super, P), $PL! = PL2$. $[w, false]$	avoid transporting a part when it needs to be transported back	**(heu-back-forth)**
#**maximize**{w, Part, Super, P : producedAt(Part, P), (**choiceLvL2**)(Part)}.	maximize number of sites where sub- and assembly are done	**(min-hubs-on-locations)**
#**maximize**{w, Part, P : productionLocation(P)}	maximize production at certain sites	**(min-prefer-certain-locs)**
#**maximize**{1, Part, From, To, TM, _, _, _, _), truck(TM)}.	maximize usage of trucks as transportation mean	**(min-prefer-trucks)**
backAndForth(Part, SuperSuper) ← productionPlan(SuperSuper, Super), productionPlan(Super, Part), producedAt(Super, P2), producedAt(SuperSuper, P), P2! = P #**minimize**{w, Part, Super, P : backAndForth(Part, SuperSuper)}	minimize transporting a part to be returned	**(min-back-forth)**
#**maximize**{1@1w, P : producedAt(Part, P)}.	maximize number of production sites	**(nb-Sites-WC)**
← #**sum**{Value, Part : producedAt(Part, P), valAdded(Part, Value), locatedIn(P,)} ≤ n	work share for France,	**(minVal-IC)**
← #**sum**{Value, Part : producedAt(Part, P), valAdded(Part, Value), locatedIn(P,)} ≤ n	Germany,	
← #**sum**{Value, Part : producedAt(Part, hh), valAdded(Part, Value)} ≤ m	Hamburg and	
← #**sum**{Value, Part : producedAt(Part, tls), valAdded(Part, Value)} ≤ m	Toulouse with $m, n \in \mathbb{N}$	
#**minimize**{(Value ∗ Cost)/e@w, Part, P : producedAt(Part, P), valAdded(Part, Value), costLoc(P, Cost)}	minimize Production Costs	**(optim-1P-2C-3T)**
#**minimize**{TC@w, Part, From, To, TM : path(Part, From, To, TM, D, LT, CO2, TC)}	minimize Transportation Costs	**(optim-1T-2P-3C)** **(optim-1T-2C-3P)**
#**minimize**{CO2@w, Part, From, To, TM : path(Part, From, To, TM, D, LT, CO2, TC)}	minimize CO_2 emissions	**(optim-1C-2P-3T)** **(optim-1C-2T-3P)**
#**minimize**{((Value · Cost) + TC)/$e_1 \cdot e_2$, Part, P : producedAt(Part, P), valAdded(Part, Value), costLoc(P, Cost), path(Part, From, To, TM, D, LT, CO2, TC)}	minimize sum of production and transportation	**(optim-comb-Obj)**

Table 3: Overview of all new features to be tested .

All other Weak Constraints and heuristics are addressing to improve the Transportation and CO_2 Costs. **(min–prefer–trucks)** and **(heu–prefer–trucks)** take advantage of the fact that some transportation means are considered "standard" transportation means and therefore have the lowest Transportation Costs.
Considering the production plan, it seems reasonable that parts of the same level are transported to a possibly close site or to the same site for further assembly, the so-called hubs. Hubs can occur at different levels, but given the constraint that each site needs to produce (at least) one part in one industrial system, they should probably not be on all levels. Therefore it is chosen that sub-assembly and assembly should be produced at the same location: **(min–hubs–on–location)** and **(heu–hubs–on–location)**.
Finally the rules **(min–back-forth)** and **(min–back–forth)** are trying to avoid solutions where a part is produced in location A (previous), then sub-assembled in location B (intermediate), and then again assembled in location A (last).

4. Methodology

The methodology chosen and applied in this thesis is an *Experimental Quantitative Methodology*. It suits to quantify the influence of additional features and their according impact on the objectives systematically. Quantitative experimental methods enable to find patterns or associations among design parameters and the outputs and the according weighted dependencies [34].
This section will present how experiments will be conducted, the according data gathered and the metrics applied to measure the performance of the tested features. Three metrics will be used to determine the impacts of the dedicated rules on the overall performance, i.e. the KPIs, the *Variability* and the *Search Performance*.

4.1. Design of Experiments

The research will be performed in two phases. In the *Main Phase* an independent analysis of each feature will be conducted to have clear visibility of the single impacts on Answer Sets and their associated measures, which will be presented in the next sections. After presentation and discussion of the Main Phase results, proposals will be made for the second phase, the *Final Phase* configurations to assess the conclusions made from the Main Phase.
The reference for all experiments are the results of the **(baseline)** configuration run, including all facts, all rules and optimization statements as shown in Table 2.
An overview of all new features to be tested is shown in Table 3.
Each heuristic and Weak Constraint will be added individually to the **(baseline)** configuration for simulation.
The rules **(nb-Sites-WC)** and **(minVal-IC)** and any combination of them will each replace the **(workshare-IC)** rule as they are alternative implementations of the Workshare constraint. The constraints **(minVal-IC)** and **(nb-Sites-WC)** can be combined and will be tested as well with the label **(comb. nbSites-minVal)**. So will the combination **(workshare-IC)** and **(minVal-IC)** which is labeled **(comb. baseline-minVal)** accordingly.
The optimization statements at the bottom of Table 3 are representing the same optimization statements of the **(baseline)** configuration, only with alternative weightings of each KPIs, except for the rule **(optim-comb-Obj)**, which will replace the first two optimization statements concerning Production and Transportation Costs.
The timeout for every run is 36.000 seconds.

Every optimization run will generate the KPIs as described in Section 4.2.1. The Answer Sets will be analyzed regarding their Variability with the method as described in Section 4.2.2. And finally, the number of models will be logged for every run to compare the Search Performance for the according configuration. The results of the KPIs as well as for the Search Performance will be compared in percentages against the reference results.

The features and configurations to be tested in the Final Phase will be defined with the discussion of the Main Phase results in Section 6.

4.2. Evaluation Metrics

The Evaluation Metrics described in this section are the parameters to measure the impact of each feature on the performance of the program. There will be two kinds of evaluation metrics:

- the first group of metrics will refer to the main objective functions of the industrial and logistics system itself and act as a direct indicator of the impact on the performance i.e. the KPIs
- the second group of metrics are auxiliary performance indicators measuring the behavior of the solver and optimizer, especially with regards to its search tree traversal and act as indirect indicators, i.e. Variability and Search Performance

4.2.1. Key Performance Indicators of Objective Functions

Key Performance Indicators are all objective functions of the industrial and logistics system. They refer to the two problems of *Optimal Allocation of Part Production Across Sites* and *Optimal Routing Across Production Sites* as described in Section 1.2.1. These are:

- Production Costs: Cost for production of all parts that need to be produced. Production Costs are driven by the workload per part w_p, which is fixed, and the cost factor of the dedicated production location c_s that is chosen to produce the part, which is location dependent. The specific formula for the Production Costs $ProdCost$ of a part is

  $$ProdCost = w_p \cdot c_s$$

  which are implemented by the facts **(valAdd)** and **(costLoc)**. See the according formal optimization statements and implementation of the sum of all Production Costs as an aggregate in the logic program in the first rule of the group of rules **(optim-baseline)**. It cumulates in the first term of the term tuple as described in the aggregate part of section 2.4 and Equation 7.

- Transportation Costs: Cost of transportation activities are driven by the distance D of the transport activity and the transport cost per distance factor for the transportation mean chosen. The according cost for a transportation link between two production sites that perform consecutive activities are given by facts as represented in the eighth argument in **(canBeTransp)**. See the according formal optimization statements and implementation of the sum of all transportation costs as an aggregate in the logic program in the second rule of the group of rules **(optim-baseline)**.

- CO_2 emissions: Cost of CO_2 emissions is driven by the distance D of the transport activity and the transportation mean chosen similarly as for the Transportation Costs. Nevertheless, these two are not fully linear as CO_2 emissions and operational cost of a transportation mean are independent from each others. The according CO_2 cost for a transportation link between two production sites that perform consecutive activities is given by facts, represented as the seventh argument in **(canBeTransp)**. This optimization statement of the sum of all CO_2 costs is implemented as an aggregate in the logic program in the third rule of the group of rules **(optim-baseline)**.

The general reference to compare the KPIs as defined above will be the according results from the **(baseline)** run. All runs need to be confirmed by a validation run.
For the consolidated summary of all tested features, two additional trivial validation values will be determined, i.e. the best achievable Production Costs – labeled **(trivial-production)** – and Transportation Costs – labeled **(trivial-transportation)**. Both are resulting from a **(baseline)** configuration run with 360.000 seconds timeout, except that for the optimization statements **(optim-baseline)** only a single *#minimize* statement each for Production and Transportation Costs only will be implemented. For the Production Costs also the Workshare constraint **(workshare-IC)** is removed, so that parts can be assigned to the lowest cost production sites where possible without being forced to consider locations with higher costs.
These two values should help to better evaluate the quality of the potential Pareto Front in a consolidated scatter plot of the two concerned KPIs.
As the CO_2 emissions have the lowest priority in the **(baseline)** configuration, they will not be analyzed for Pareto optimal Answer Sets. The according figures of the KPI will be available in the appendix.

4.2.2. Variability Measurement

The Variability of the Answer Sets is an indicator of how extensively the search space is explored. The more the Answer Sets are diverse, the more confidence can be gained that no global or local optimum is missed.
To measure Variability of the Answer Sets, a similarity unit of measurement is selected that bases on the *Jaccard-Coefficient*. Originally developed by Salton & McGill in 1983 [35] in the context of electronic information retrieval, it is generally speaking a unit that measures commonality of two objects.

$$Jaccard(a,b) = \frac{|\Gamma(a) \cap \Gamma(b)|}{|\Gamma(a) \cup \Gamma(b)|} \tag{10}$$

See in Equation 10 how a *Jaccard coefficient* can be formalized. Following this, the similarity is defined by the ratio of elements that two objects a and b share versus the sum of all elements they relate to individually.
In this context it aims at comparing two Answer Sets that are represented as tables including the columns *Part*, *Produced At*, *Transported To* and *Transport Mean*. The row

elements of the table are all transportation steps derived from the set of *path/8* predicates of the solution. As the path is determined by two decisions, i.e the allocation of parts to production sites and the choice of transportation means between two sites, the Jaccard coefficient will be split into two components. This could be regarded as the split into node-similarity and edge-similarity when each solution is represented as a graph as we did earlier.
First, the set with respect to the allocation of a part $p \in P$ to sites $s \in S$ for a model is defined as

$$S_{nodes}(p) = \{S_{nodes_1}(p), \ldots, S_{nodes_n}(p)\}$$

with $S_{nodes_i}(p) = \{(p, s_i) \mid p \in P, 1 \leq i \leq n\}$ with n being the total number of sites $s \in S$ producing part $p \in P$. The corresponding set of all such sets is denoted as $S_{nodes} = \{S_{nodes}(p) \mid p \in P\}$. Likewise, the set of all sets with respect to all choices of transportation means between sites for a model (or Answer Set) is defined as

$$S_{edges} = \{S_{edges_1}, \ldots, S_{edges_m}\}$$

with m being the number of all performed transportation activities. Accordingly, we compute the Jaccard similarity between two Answer Sets m_1 and m_2 following Equation 10:

$$J_{production} = \frac{|S_{nodes_{m_1}} \cap S_{nodes_{m_2}}|}{|S_{nodes_{m_1}} \cup S_{nodes_{m_2}}|}$$
$$J_{transport} = \frac{|S_{edges_{m_1}} \cap S_{edges_{m_2}}|}{|S_{edges_{m_1}} \cup S_{nodes_{m_2}}|}$$

The similarity measure is applied in three ways. First, we compare the models of the first and last model of some runs to determine the internal Variability within a single run. Second, we aim to see the Variability against a reference model, which is the last model of the **(baseline)** run to see how much Variability can be achieved overall among different runs with different encoding. Finally, the configurations are compared among each other to check whether they produce diverse Answer Sets.

4.2.3. Search Performance

The *Search Performance* is a mean to measure how good the solver can find models for the given configuration. It will be measured by logging the number of models found until standard timeout, i.e. 36.000 seconds. The figures will be presented in percentages versus the models found by the (baseline) configuration.
Another aspect of *Search Performance* is whether an optimum was found before timeout or not.
Finally, the grounding time is the last element of *Search Performance*, which will be tracked for every optimization run.

5. Results of Main Phase

This section presents the results of the conducted experiments of the Main Phase as described in Section 4.
Experiments were carried out on an AMD EPYC 7443P 24-Core Processor, 64GB RAM type DDR4-3200. The timeout was set to 36.000 seconds. Note that in all figures, only every 100th model is plotted. The best performing rules regarding the KPIs and Search Performance will be highlighted in grey in the tables.

5.1. Baseline Reference

Grounding took about half a minute and the first model was found after a few seconds.
The reference for all further evaluations is the **(baseline)** run with a timeout of 36.000 seconds. It found more than 13.300 models. No optimum was found. See the output of the model´s *path/8* atoms with 132 lines, each representing one of the core routes in Table 4. It contains all the arguments reflecting the decisions taken, i.e. parts allocated to production locations (columns *producedAt* and *part*), the routing (columns *transportedTo* and *Transport Mean*) and the associated costs (columns *Distance, TranspRC, Co2* and *Lead-Time*).

	number of models found after	
configuration	optimum found	36.000sec
(**baseline**)	no	13.300 (100%)

Table 5: The number of models found for the (baseline) run

The respective KPIs for the last model are shown in Table 6 and defined as the reference with 100% each. The figures refer to the definitions from Section 1.2.3, but are all factorized, so that real figures are hidden due to confidentiality.

	Costs after 36000sec timeout		
model	Production	Transport	CO_2
last (**baseline**)	1.498.375 (100%)	1.322.027 (100%)	145.597 (100%)

Table 6: The reference KPIs of the (baseline) run.

Table 7 shows the percentages of the KPIs (Production Costs, Transportation Costs and CO_2 costs) for the first and the last model of the baseline run. As the last model

Unnamed: 0	part	producedAt	transportedTo	Transport Mean	Distance	TranspRC	Co2	LeadTime
0	s6Tpart	locationC	locationD	truckOversizedLowBed	2899	13044	2261	58
1	s4Upperpart	locationC	locationD	truckOversizedLowBed	2899	13044	2261	58
2	s5UpperTpart	locationF	locationD	truckOversizedLowBed	2311	10399	1802	46
3	s4LowerTpart	locationC	locationD	truckGeneralCargo	2899	10146	2055	53
4	s6Upperpart	locationC	locationD	truckGeneralCargo	2899	10146	2055	53
5	s3Fpart	locationJ	locationD	truckOversizedLowBed	780	3511	608	16
6	part7	locationI	locationHdeBretagne	oversizedBelugaXL	463	14824	22916	1
7	s4LowerTpart	locationC	locationB	truckLongDistance	2841	9944	2015	52
8	part4	locationD	locationB	oversizedUnderDeckRoRoShip	153	978	24	6
9	part3	locationD	locationB	oversizedUnderDeckRoRoShip	153	978	24	6
10	part7	locationD	locationB	oversizedUnderDeckRoRoShip	153	978	24	6
11	s4LowerTpart	locationC	locationB	truckGeneralCargo	2841	9944	2015	52
12	s5LowerTpart	locationC	locationB	truckGeneralCargo	2841	9944	2015	52
13	s2UpperTpart	locationF	locationB	truckOversizedLowBed	2368	10658	1847	47
14	part7	locationI	locationB	oversizedBelugaXL	631	20207	31239	1
15	part67	locationHdeBretagne	locationE	oversizedBelugaXL	492	15745	24341	1
16	part1	locationHdeBretagne	locationE	oversizedBelugaXL	492	15745	24341	1
17	part2345	locationHdeBretagne	locationE	oversizedBelugaXL	492	15745	24341	1
18	part67	locationB	locationE	oversizedBelugaXL	1266	40507	62621	2
19	part1	locationB	locationE	oversizedBelugaXL	1266	40507	62621	2
20	part2345	locationB	locationE	oversizedBelugaXL	1266	40507	62621	2
21	part5	locationB	locationB	intrasiteTransport	0	0	0	0
22	part2345	locationB	locationB	intrasiteTransport	0	0	0	0
23	part3	locationB	locationB	intrasiteTransport	0	0	0	0
24	part1	locationB	locationB	intrasiteTransport	0	0	0	0
25	part6	locationB	locationB	intrasiteTransport	0	0	0	0
26	s3Tpart	locationB	locationB	intrasiteTransport	0	0	0	0
27	part4	locationB	locationB	intrasiteTransport	0	0	0	0
28	part2	locationB	locationB	intrasiteTransport	0	0	0	0
29	part67	locationB	locationB	intrasiteTransport	0	0	0	0
30	s3Tpart	locationD	locationD	intrasiteTransport	0	0	0	0
31	s6Upperpart	locationG	(naplesHarbor,locationD)	(truckGeneralCargo,truckLongDistance)	1902	6656	1349	35
32	s2LowerTpart	locationG	(naplesHarbor,locationD)	(truckOversizedStandardBed,truckGeneralCargo)	1902	6684	1351	35
33	s3Fpart	locationJ	(marseilleHarbor,locationB)	(truckOversizedLowBed,truckOversizedLowBed)	2339	10523	1824	47
34	part2	locationD	(locationHHarbor,locationHdeBretagne)	(oversizedUnderDeckRoRoShip,largeOversizedRoadTransport)	2599	17623	481	104
35	part3	locationD	(locationHHarbor,locationHdeBretagne)	(oversizedUnderDeckRoRoShip,largeOversizedRoadTransport)	2599	17623	481	104
36	part7	locationD	(locationHHarbor,locationHdeBretagne)	(oversizedUnderDeckRoRoShip,largeOversizedRoadTransport)	2599	17623	481	104
37	s6Fpart	locationG	(locationHHarbor,locationD)	(truckOversizedLowBed,truckOversizedLowBed)	3093	13920	2413	62
38	part67	locationHdeBretagne	(locationHHarbor,locationB)	(largeOversizedRoadTransport,oversizedUnderDeckRoRoShip)	2678	18111	492	107
39	part2345	locationHdeBretagne	(locationHHarbor,locationB)	(largeOversizedRoadTransport,oversizedUnderDeckRoRoShip)	2678	18111	492	107
40	part67	locationB	(locationHHarbor,locationK)	(oversizedUnderDeckRoRoShip,largeOversizedRoadTransport)	9219	60882	1599	369
41	s4UpperTpart	locationC	(dunkerqueHarbor,locationD)	(truckGeneralCargo,truckLongDistance)	3804	13313	2697	69
42	s3Fpart	rochefort	(dunkerqueHarbor,locationD)	(truckOversizedLowBed,truckOversizedLowBed)	1398	6293	1091	28
43	s3Upperpart	locationC	(dunkerqueHarbor,locationD)	(truckLongDistance,seaLargeContainerShip)	3763	11926	2278	81
44	s3Fpart	rochefort	(gruenendeichHarbor,locationB)	(truckOversizedLowBed,truckOversizedLowBed)	1351	6083	1055	27
45	s4Upperpart	locationC	(gruenendeichHarbor,locationB)	(truckOversizedStandardBed,truckOversizedStandardBed)	2865	12897	2236	57
46	s4UpperTpart	locationC	(gruenendeichHarbor,locationD)	(truckLongDistance,truckLongDistance)	2952	10333	2094	54
47	s4UpperTpart	locationC	(gruenendeichHarbor,locationB)	(truckLongDistance,truckShortDistance)	2865	10013	2032	52
48	s2UpperTpart	locationC	(gruenendeichHarbor,locationB)	(truckLongDistance,truckLongDistance)	2865	10030	2032	52
49	s2LowerTpart	locationC	(gruenendeichHarbor,locationB)	(truckLongDistance,truckLongDistance)	2865	10030	2032	52
50	s2LowerTpart	locationF	(gruenendeichHarbor,locationD)	(truckGeneralCargo,truckGeneralCargo)	2470	8645	1752	45
51	s2LowerTpart	locationF	(gruenendeichHarbor,locationB)	(truckGeneralCargo,truckGeneralCargo)	2383	8342	1690	43
52	s6Fpart	locationG	(gruenendeichHarbor,locationB)	(truckOversizedLowBed,truckOversizedLowBed)	1872	8428	1461	37
53	s6Tpart	locationC	(gruenendeichHarbor,locationB)	(truckOversizedLowBed,truckOversizedLowBed)	2865	12897	2236	57
54	s5LowerTpart	locationC	(gruenendeichHarbor,locationD)	(truckOversizedLowBed,truckOversizedStandardBed)	2952	13287	2303	59
55	s4UpperTpart	locationC	(gruenendeichHarbor,locationD)	(truckGeneralCargo,truckGeneralCargo)	2952	10333	2094	54
56	s3LowerTpart	locationC	(gruenendeichHarbor,locationD)	(truckGeneralCargo,truckShortDistance)	2952	10247	2094	54
57	s3Tpart	locationD	(locationBHarbor,locationB)	(oversizedUnderDeckRoRoShip,kugelbake)	167	1272	42	7
58	part6	locationD	(locationBHarbor,locationB)	(oversizedUnderDeckRoRoShip,kugelbake)	167	1272	42	7
59	part5	locationD	(locationBHarbor,locationB)	(oversizedUnderDeckRoRoShip,kugelbake)	167	1272	42	7
60	s3Tpart	locationB	(locationBHarbor,locationD)	(oversizedUnderDeckRoRoShip,kugelbake)	167	4249	266	13
61	s5UpperTpart	locationC	(locationBHarbor,locationD)	(truckOversizedLowBed,oversizedUnderDeckRoRoShip)	3000	13799	2242	63
62	s5UpperTpart	locationC	(locationBHarbor,locationB)	(truckOversizedLowBed,kugelbake)	2853	13059	2234	58
63	s6Lowerpart	locationC	(locationBHarbor,locationD)	(truckOversizedLowBed,truckOversizedLowBed)	2979	13406	2323	60
64	s6Lowerpart	locationC	(locationBHarbor,locationB)	(truckOversizedLowBed,oversizedUnderDeckRoRoShip)	2853	12857	2219	57
65	s6Fpart	locationC	(locationBHarbor,locationD)	(truckOversizedLowBed,truckOversizedLowBed)	2979	13406	2323	60
66	s6Fpart	locationC	(locationBHarbor,locationB)	(truckOversizedLowBed,truckOversizedLowBed)	2853	12839	2225	57
67	part2	locationD	(locationBHarbor,locationB)	(kugelbake,kugelbake)	167	4451	281	14
68	s5UpperTpart	locationC	(locationBHarbor,locationD)	(truckLongDistance,truckGeneralCargo)	2979	10426	2113	54
69	s5LowerTpart	locationC	(locationBHarbor,locationD)	(truckLongDistance,seaLargeContainerShip)	3000	10176	2026	58
70	s5Upperpart	locationC	(locationBHarbor,locationD)	(truckLongDistance,truckGeneralCargo)	2979	10426	2113	54
71	s2Upperpart	locationC	(locationBHarbor,locationD)	(truckLongDistance,truckShortDistance)	2980	10320	2113	54
72	s5UpperTpart	locationF	(locationBHarbor,locationB)	(truckOversizedLowBed,oversizedUnderDeckRoRoShip)	2387	10762	1856	48
73	s2UpperTpart	locationF	(locationBHarbor,locationD)	(truckOversizedLowBed,truckOversizedLowBed)	2513	11311	1960	51
74	s4Lowerpart	locationC	(locationBHarbor,locationB)	(truckGeneralCargo,kugelbake)	2853	10216	2033	53
75	s5LowerTpart	locationC	(locationBHarbor,locationB)	(truckGeneralCargo,truckLongDistance)	2853	9986	2023	52
76	s3Upperpart	locationC	(locationBHarbor,locationB)	(truckGeneralCargo,seaLargeContainerShip)	2853	9964	2017	52
77	s5Upperpart	locationC	(locationBHarbor,locationB)	(truckGeneralCargo,truckShortDistance)	2853	9977	2023	52
78	s2LowerTpart	locationC	(locationBHarbor,locationD)	(truckGeneralCargo,truckLongDistance)	2980	10430	2113	54
79	s3Upperpart	locationA	(locationAHarbor,locationB)	(truckShortDistance,seaLargeContainerShip)	27800	40057	1747	1111
80	s4Lowerpart	locationC	((naplesHarbor,marseilleHarbor),locationD)	(truckGeneralCargo,seaLargeContainerShip,truckLongDistance)	4176	12840	2404	96
81	s6Upperpart	locationG	((naplesHarbor,marseilleHarbor),locationB)	(truckLongDistance,seaLargeContainerShip,truckLongDistance)	2301	6276	1074	62
82	s6Upperpart	locationC	((portoHarbor,marseilleHarbor),locationB)	(truckGeneralCargo,seaLargeContainerShip,truckLongDistance)	7793	22789	4118	189
83	s4LowerTpart	locationA	((locationAHarbor,marseilleHarbor),locationD)	(truckShortDistance,seaLargeContainerShip,seaLargeContainerShip)	30908	44531	1941	1236
84	s4UpperTpart	locationA	((locationAHarbor,marseilleHarbor),locationB)	(truckShortDistance,seaLargeContainerShip,truckGeneralCargo)	27020	41837	2610	1050
85	s4UpperTpart	locationA	((locationAHarbor,marseilleHarbor),locationD)	(truckGeneralCargo,seaLargeContainerShip,truckGeneralCargo)	26962	41649	2569	1049
86	s4UpperTpart	locationA	((locationAHarbor,locationHHarbor),locationB)	(truckShortDistance,seaLargeContainerShip,seaLargeContainerShip)	28257	40715	1775	1130
87	s2UpperTpart	locationA	((locationAHarbor,shanghaiHarbor),locationB)	(truckGeneralCargo,seaLargeContainerShip,seaLargeContainerShip)	27996	40351	1759	1119
88	s3LowerTpart	locationA	((locationAHarbor,shanghaiHarbor),locationD)	(truckGeneralCargo,seaLargeContainerShip,seaLargeContainerShip)	27916	40237	1754	1116
89	s5Upperpart	locationE	((portoHarbor,locationBHarbor),locationB)	(truckLongDistance,seaLargeContainerShip,truckLongDistance)	4321	8473	975	149
90	s4UpperTpart	locationC	((dunkerqueHarbor,locationBHarbor),locationB)	(truckLongDistance,seaLargeContainerShip,truckLongDistance)	3865	12095	2290	85
91	s5UpperTpart	locationC	((dunkerqueHarbor,locationBHarbor),locationB)	(truckLongDistance,seaLargeContainerShip,truckGeneralCargo)	3865	12095	2290	85
92	s5Upperpart	locationE	((dunkerqueHarbor,locationBHarbor),locationD)	(truckLongDistance,seaLargeContainerShip,seaLargeContainerShip)	1818	4607	738	52
93	s3LowerTpart	locationC	((dunkerqueHarbor,locationBHarbor),locationB)	(truckGeneralCargo,seaLargeContainerShip,truckLongDistance)	3865	12095	2290	85
94	s4LowerTpart	locationA	((locationAHarbor,locationBHarbor),locationB)	(truckShortDistance,seaLargeContainerShip,kugelbake)	27815	40329	1764	1112
95	s4LowerTpart	locationA	((locationAHarbor,locationBHarbor),locationB)	(truckShortDistance,seaLargeContainerShip,truckGeneralCargo)	27815	40099	1754	1111
96	s5UpperTpart	locationA	((locationAHarbor,locationBHarbor),locationD)	(truckShortDistance,seaLargeContainerShip,truckLongDistance)	27942	40543	1844	1113
97	s5UpperTpart	locationA	((locationAHarbor,locationBHarbor),locationB)	(truckGeneralCargo,seaLargeContainerShip,truckShortDistance)	27815	40104	1754	1111
98	s5LowerTpart	locationA	((locationAHarbor,locationBHarbor),locationD)	(truckShortDistance,seaLargeContainerShip,truckShortDistance)	27942	40433	1844	1113
99	s2LowerTpart	locationA	((locationAHarbor,locationBHarbor),locationB)	(truckGeneralCargo,seaLargeContainerShip,oversizedUnderDeckRoRoShip)	27815	40141	1749	1111
100	s2Upperpart	locationA	((locationAHarbor,locationBHarbor),locationD)	(truckGeneralCargo,seaLargeContainerShip,truckShortDistance)	27942	40447	1844	1113
101	s2Upperpart	locationA	((locationAHarbor,locationBHarbor),locationB)	(truckGeneralCargo,seaLargeContainerShip,oversizedUnderDeckRoRoShip)	27815	40141	1749	1111
102	s3LowerTpart	locationA	((locationAHarbor,locationBHarbor),locationB)	(truckGeneralCargo,seaLargeContainerShip,truckGeneralCargo)	27815	40113	1754	1111
103	s4Lowerpart	locationA	((locationAHarbor,gibraltarHarbor),locationB)	(truckGeneralCargo,seaLargeContainerShip,seaLargeContainerShip)	28423	40967	1786	1136
104	s2UpperTpart	locationA	((locationAHarbor,gibraltarHarbor),locationD)	(truckShortDistance,seaLargeContainerShip,seaLargeContainerShip)	28344	40839	1781	1133
105	s4Lowerpart	locationA	((locationAHarbor,portoHarbor),locationD)	(truckGeneralCargo,seaLargeContainerShip,seaLargeContainerShip)	27803	40074	1747	1112
106	s5LowerTpart	locationA	((locationAHarbor,portoHarbor),locationB)	(truckGeneralCargo,seaLargeContainerShip,truckLongDistance)	27061	43954	3255	1030
107	s3Upperpart	locationA	((locationAHarbor,portoHarbor),locationD)	(truckGeneralCargo,seaLargeContainerShip,seaLargeContainerShip)	27803	40074	1747	1112
108	s2LowerTpart	locationA	((locationAHarbor,portoHarbor),locationD)	(truckOversizedLowBed,seaLargeContainerShip,seaLargeContainerShip)	27803	40093	1748	1112
109	s4UpperTpart	locationA	((locationAHarbor,tianjinHarbor),locationD)	(truckGeneralCargo,seaLargeContainerShip,seaLargeContainerShip)	29472	42478	1850	1178
110	s4LowerTpart	locationA	((locationAHarbor,casablancaHarbor),locationD)	(truckGeneralCargo,seaLargeContainerShip,seaLargeContainerShip)	28065	40452	1764	1122
111	s6Tpart	locationA	((locationAHarbor,marseilleHarbor),locationB)	(truckOversizedLowBed,oversizedUnderDeckRoRoShip,truckOversizedLowBed)	27020	170214	5209	1052
112	part2	locationB	((locationBHarbor,locationHHarbor),locationHdeBretagne)	(kugelbake,oversizedUnderDeckRoRoShip,largeOversizedRoadTransport)	2693	18424	512	108
113	part4	locationD	((locationBHarbor,locationHHarbor),locationHdeBretagne)	(oversizedUnderDeckRoRoShip,oversizedUnderDeckRoRoShip,largeOversizedRoadTransport)	2840	19164	520	113
114	part4	locationB	((locationBHarbor,locationHHarbor),locationHdeBretagne)	(oversizedUnderDeckRoRoShip,oversizedUnderDeckRoRoShip,largeOversizedRoadTransport)	2693	18222	497	107
115	part6	locationB	((locationBHarbor,locationHHarbor),locationHdeBretagne)	(oversizedUnderDeckRoRoShip,oversizedUnderDeckRoRoShip,largeOversizedRoadTransport)	2693	18222	497	107
116	part3	locationB	((locationBHarbor,locationHHarbor),locationHdeBretagne)	(oversizedUnderDeckRoRoShip,oversizedUnderDeckRoRoShip,largeOversizedRoadTransport)	2693	18222	497	107
117	part5	locationD	((locationBHarbor,locationHHarbor),locationHdeBretagne)	(kugelbake,oversizedUnderDeckRoRoShip,largeOversizedRoadTransport)	2840	22343	759	120
118	s4Upperpart	locationA	((locationAHarbor,locationHHarbor),locationB)	(truckGeneralCargo,oversizedUnderDeckRoRoShip,truckGeneralCargo)	26882	168228	5024	1047
119	part6	locationD	((dunkerqueHarbor,locationHHarbor),locationHdeBretagne)	(oversizedUnderDeckRoRoShip,oversizedUnderDeckRoRoShip,largeOversizedRoadTransport)	3203	21489	577	128
120	part5	locationB	((dunkerqueHarbor,locationHHarbor),locationHdeBretagne)	(oversizedUnderDeckRoRoShip,oversizedUnderDeckRoRoShip,largeOversizedRoadTransport)	3291	22052	592	132
121	part1	locationB	((locationBHarbor,locationKHarbor),locationK)	(kugelbake,oversizedUnderDeckRoRoShip,largeOversizedRoadTransport)	9233	61176	1617	371
122	part2345	locationB	((locationBHarbor,locationKHarbor),locationK)	(oversizedUnderDeckRoRoShip,oversizedUnderDeckRoRoShip,largeOversizedRoadTransport)	9233	60974	1602	370
123	part67	locationHdeBretagne	((locationHHarbor,locationKHarbor),locationK)	(largeOversizedRoadTransport,oversizedUnderDeckRoRoShip,largeOversizedRoadTransport)	8399	56605	1531	337
124	part1	locationHdeBretagne	((locationHHarbor,locationKHarbor),locationK)	(largeOversizedRoadTransport,oversizedUnderDeckRoRoShip,largeOversizedRoadTransport)	8399	56605	1531	337
125	part2345	locationHdeBretagne	((locationHHarbor,locationKHarbor),locationK)	(largeOversizedRoadTransport,oversizedUnderDeckRoRoShip,largeOversizedRoadTransport)	8399	56605	1531	337
126	s2LowerTpart	locationG	((dunkerqueHarbor,locationBHarbor),locationB)	(truckGeneralCargo,oversizedUnderDeckRoRoShip,truckGeneralCargo)	2469	10661	1369	60
127	s2UpperTpart	locationC	((dunkerqueHarbor,locationBHarbor),locationD)	(truckGeneralCargo,oversizedUnderDeckRoRoShip,seaLargeContainerShip)	4012	15733	2361	91
128	part1	locationHdeBretagne	((locationHHarbor,locationBHarbor),locationB)	(largeOversizedRoadTransport,oversizedUnderDeckRoRoShip,kugelbake)	2693	18404	510	108
129	s6Tpart	locationA	((locationAHarbor,gibraltarHarbor),locationD)	(truckOversizedLowBed,oversizedUnderDeckRoRoShip,truckOversizedLowBed)	27011	167738	5996	1026
130	s2UpperTpart	locationA	((locationAHarbor,gibraltarHarbor),locationB)	(truckOversizedLowBed,oversizedUnderDeckRoRoShip,truckOversizedLowBed)	27068	167997	6041	1027
131	s4UpperTpart	locationA	((locationAHarbor,gibraltarHarbor),locationD)	(truckOversizedStandardBed,oversizedUnderDeckRoRoShip,truckOversizedStandardBed)	27011	167738	5996	1026
132	s5LowerTpart	locationA	((locationAHarbor,gibraltarHarbor),locationB)	(truckOversizedLowBed,oversizedUnderDeckRoRoShip,truckGeneralCargo)	27068	165257	5847	1022

Table 4: The reference model last (baseline) printed as a table.

	% of models cost to last (**baseline**)		
model	Production	Transport	CO_2
last (**baseline**)	100	100	100
first (**baseline**)	115	138	131

Table 7: KPI percentages of the baseline comparing the first and the last model

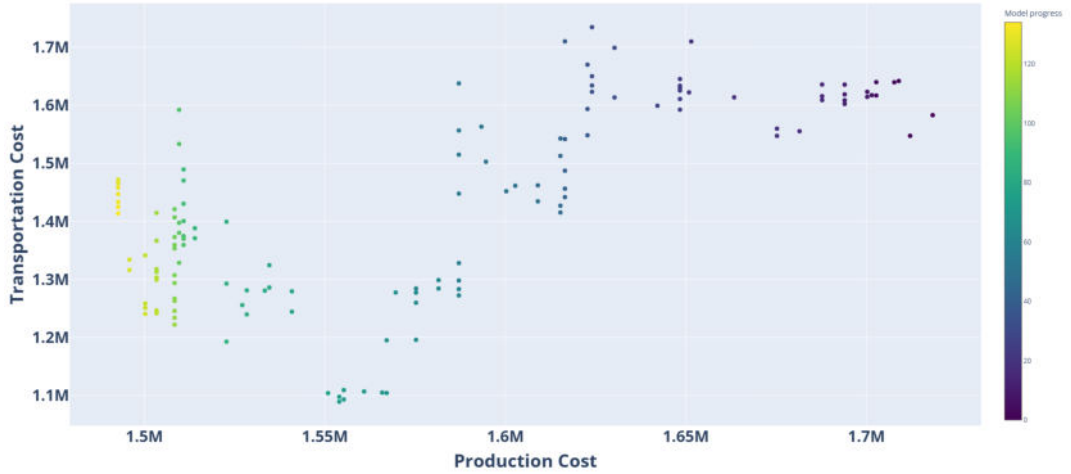


Figure 12: Evolution of the Production Costs with respect to the Transportation Costs within the (baseline) run

of the baseline run does act as the main reference for all evaluations it is set to 100% as described above. The table shows that the values at the start of the run were higher with 115% for Production Costs, 138% for Transportation Costs and 131% for the CO_2 emissions.

Figure 12 visualizes the evolution of the Production Costs with respect to the Transportation Costs. Each dot represents a model and the progress is shown by the colors of the dots, moving from dark purple for the first model to bright yellow for the last. A Pareto Front is indicated as described in Section 1.2.3 and Figure 5. The last models are those with the lowest Production Costs from the supposed Pareto Front, but not the lowest Transportation Costs that were found in earlier iterations.

Figure 12 on the other hand shows the evolution of the Production Costs with respect to the CO_2 costs with a similar behavior. The models move from the upper

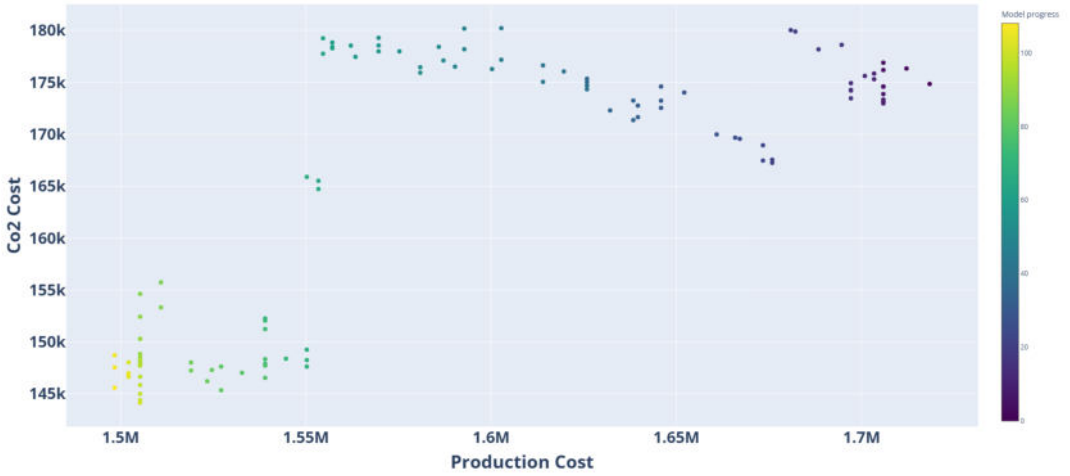


Figure 13: Evolution of the Production Costs with respect to the CO_2 costs within the (baseline) run

right corner to the lower left and a Pareto Front is indicated, but less obvious for both KPIs.

The last scatter plot in Figure 14 visualizes the evolution of Transportation Costs with respect to the CO_2 costs. While there is still a tendency of dots moving from the upper right corner to the lower left, there is no apparent Pareto Front and the last models are clearly not close to a potential optimum with regards to the two KPIs.

Finally, Figure 15 presents how the Jaccard similarity values evolve from the first to the last model within the (baseline) run. The x-axis is the index of the model (every 100th printed) and the y-axis is the Jaccard similarity value, which is split into production (orange) and transportation (green) similarity. The figure shows the according similarity values of each model compared with the last (best) model.

The first model of the baseline run has similarities with the last model between 0.16 for transportation and 0.23 for production similarity. Similarity values steadily go up until finally reaching 1.0 for both values. It can be observed that there is an exponential looking increase of the transportation similarity at the end of the run. This is an effect resulting from the prioritization of the optimization statements and the way the solver manages these. With the (baseline) configuration, first the Production Costs are optimized, then the Transportation Costs. Therefore, once the Production Costs value has improved (one iteration), several iterations of improving the Transportation Costs follow. This reflects in the similarity values for transportation. Secondly, transportation similarity is production similarity-dependent, meaning that a transportation link can only be similar, when both nodes (production sites) are already similar. Together this explains the increase at the end and is

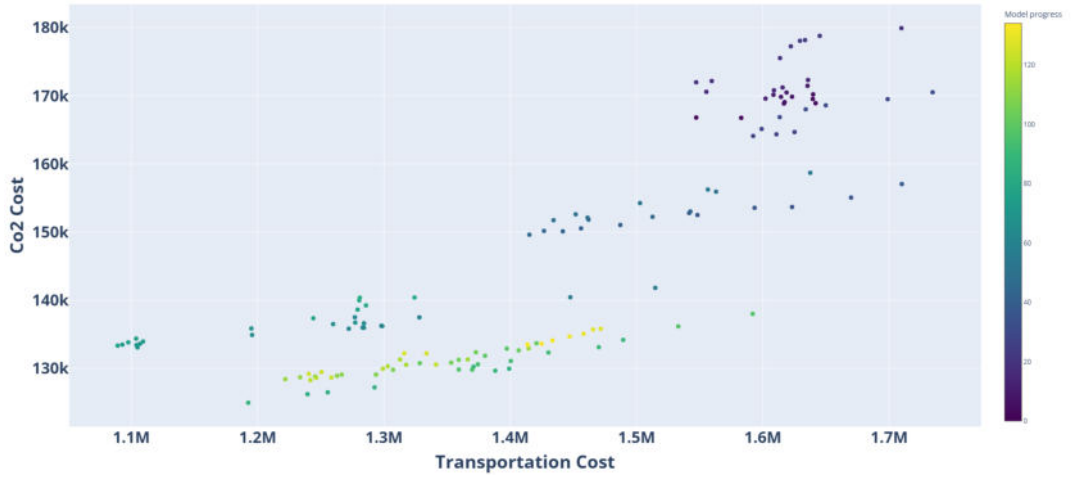


Figure 14: Evolution of the Transportation Costs with respect to the CO_2 costs within the (baseline) run

also observed for all other runs (compare Figures 18 and 19).

The models are represented by the *path/8* predicate as described in rule **(choicePath)**.

5.2. Workshare Constraint Implementations

The Workshare Constraint and its implementation is one of the elements to test features as described in Section 3.3. Find the results below split by KPI evolution, Variability and Search Performance.

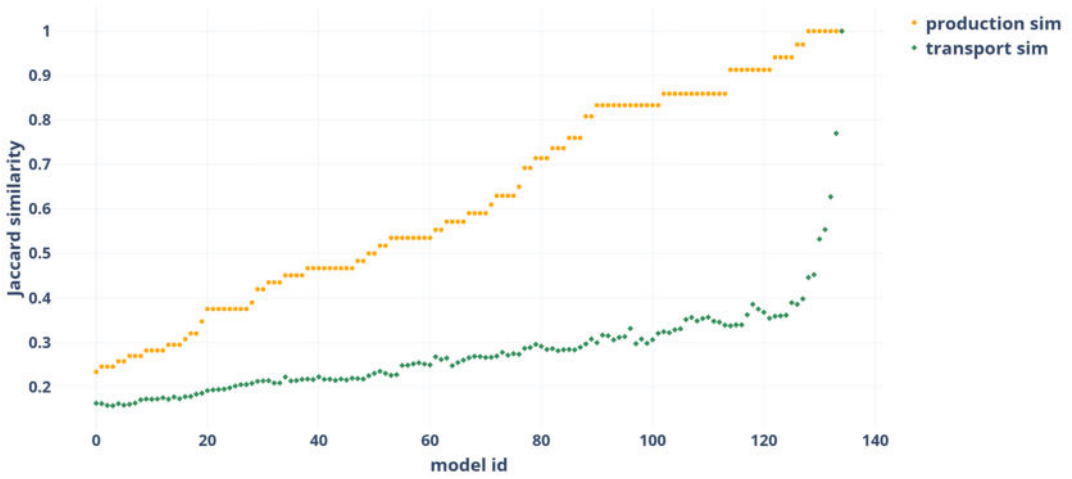
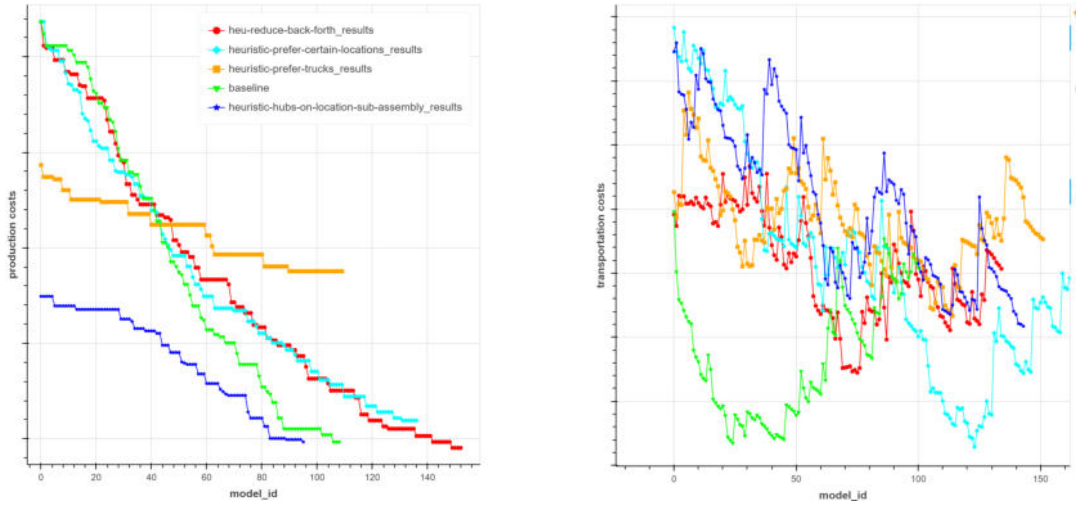


Figure 15: Evolution of the similarity values within the baseline run.

5.2.1. KPI evolution



(a) Evolution of the Production Costs for the different Workshare implementations

(b) Evolution of the Transportation Costs for the different Workshare implementations

Figure 16: Evolution of the KPIs for the different Workshare implementations

Figure 16 shows how the KPIs evolved over time. The x-axis refers to the model id (every 100th model) and the y-axis refers to the respective KPI. Regarding the Production Costs, the baseline run, the (min-Val-IC) run and the (comb. nbSites-WC/minVal-IC) run progress very closely and produce nearly the same number of models while the (comb. baseline/minVal-IC) starts with the far best Production

(workshare) implementation	% of models cost to last (**baseline**) Production	Transport	CO_2
last (minVal-IC)	99%	107%	118%
last (nbSites-WC)	102%	106%	110%
last (comb. baseline/minVal-IC)	100%	100%	205%
last (comb. nbSites-WC/minVal-IC)	100%	87%	112%

Table 8: The KPI percentages of each heuristic in relation to the last (baseline) model. The best results are highlighted in gray.

Costs and remains steadily below all other runs, but produces less models (see Figure 10. All configurations produce continuously decreasing Production Costs and end up with similar Production Costs values.
The behavior concerning the Transportation Costs differs strongly from the relatively homogeneous evolutions of the Production Costs. All configurations have ups and downs throughout the runs, but finally reach similar Transportation Costs values. The (comb. baseline/minVal-IC) rule does produce lower values compared with all other configurations from the beginning before arriving in the same area as the others towards the end of the run.
The CO_2 costs evolve similarly for all configurations, except for (comb. baseline/minVal) with steadily higher values, ending up at 205% above the baseline configuration (see Table 8).

Figure 17 is exemplarily visualizing all Workshare constraints on a scatter plot showing the Production Costs with regards to the Transportation Costs. This view emphasizes the supposed Pareto Front from Figure 12 by producing diverse models with diverse KPIs supporting the visibility of a Pareto Front. The variability of the KPIs among the different Workshare constraints indicates variability of answers sets as only differences in the models can lead to differences of the KPIs.

The direct measurement of the variability is subject to the next subsection.

5.2.2. Variability

The similarities within the runs are shown in four graphs, Figures 18a, 18b, 19a and 19b. Most of them are evolving similarly, with the production similarity starting slightly above the transportation similarity around 0.2, is steadily remaining above the transportation similarity until finally reaching 1. The transportation similarities are increasing slower for most of the run, but having a steep increase for the last models until reaching 1, too. In Figure 19, 19b the behavior of the (comb. baseline-

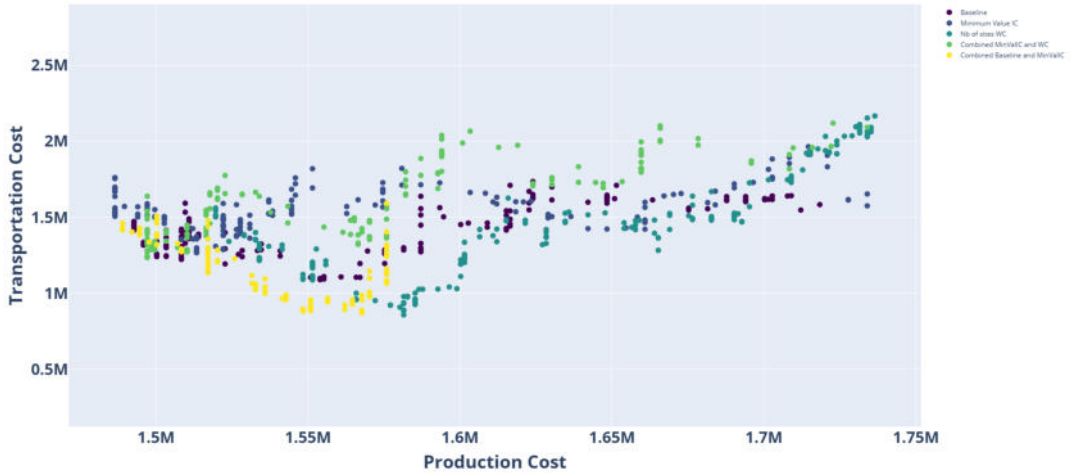
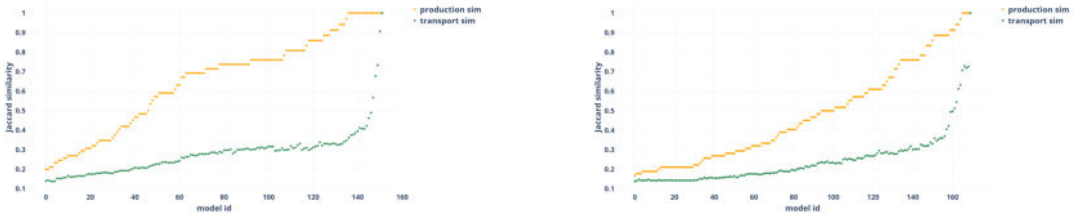


Figure 17: Evolution of the Production Costs with respect to the Transportation Costs for the different Workshare implementations



(a) Evolution of the similarities within the (minVal) configuration run

(b) Evolution of the similarities within the (nbSites-WC) configuration run

Figure 18: Similarity evolution of two single configurations

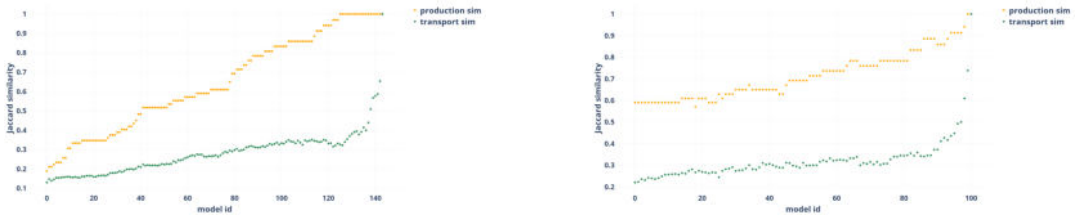
minVal-IC) run differs from the others. It starts with a higher production similarity value of 0.6 already.

Note, that the general evolution of similarities as presented here also applies to the following features and runs, so they will not be presented anymore, but are accessible in the Appendix.

Figure 9 is providing an overview of all Jaccard similarities among the applied Workshare configurations for both production (p) and transportation (t) similarities. Production similarities vary between 0.375 to 0.69 while transportation similarities range between 0.16 and 0.3.

5.2.3. Search Performance

Table 10 shows the number of models found until timeout for all different Workshare representations. The **(nbSites-WC)** configuration was able to find 26% more



(a) Evolution of the similarities within the combined (nbSites-WC) and (minVal-IC) configuration run

(b) Evolution of the similarities within the combined (baseline) and (minVal-IC) configuration run

Figure 19: Similarity evolution of two combined configurations

Heuristic	(minVal-IC)		(nbSites-WC)		(baseline/minVal-IC)		(nbSites-WC/minVal-IC)	
	p	t	p	t	p	t	p	t
(**baseline**)	0.69	0.29	0.5	0.24	0.53	0.2	0.63	0.3
(minVal-IC)	1	1	0.55	0.24	0.51	0.18	0.71	0.29
(nbSites-WC)			1	1	0.375	0.16	0.55	0.25
(comb. baseline/minVal-IC)					1	1	0.48	0.17
(comb. nbSites-WC/minVal-IC)							1	1

Table 9: Similarity comparison for all Workshare implementations for production sites (p) and transportation similarities (t).

	number of models found after	
(workshare) implementation	optimum found	36.000sec
(minVal-IC)	no	113%
(nbSites-WC)	no	126%
(comb. baseline/minVal-IC)	no	75%
(comb. minVal-IC/nbSites-WC)	no	107%

Table 10: The number of models found for the different Workshare implementations

models compared to the **(baseline)** while the **(minVal-IC)** and **(comb. minVal-IC/nbSites-WC)** configurations still found 13% or 7% respectively more models. Only **(comb. baseline/minVal-IC)** found less, namely 25% less models than the **(baseline)**.

5.3. Domain Heuristics

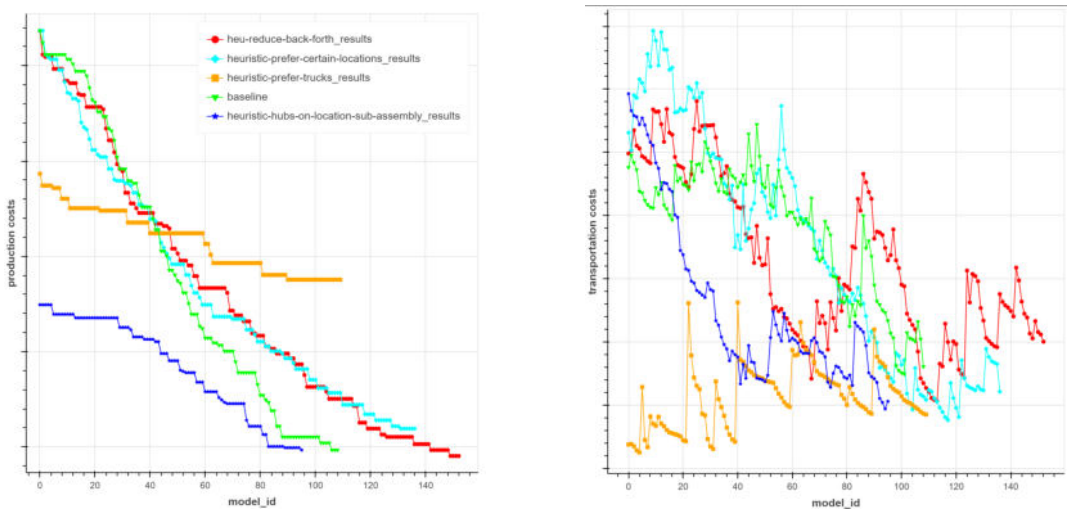
The results of the tested heuristics will be presented in this section. Find the results below split again by KPI evolution, Variability and Search Performance.

5.3.1. KPI evolution

Figure 20 gives an overview of the evolution of the KPIs of the heuristics runs.
For the Production Costs, the **(baseline)**, the **(heu-reduce-back-forth)** and the **(heu-prefer-certain-locations)** configurations move closely with the **(heu-prefer-certain-locations)** heuristic achieving better results shortly after start and the **(baseline)** finding better solutions quicker towards the end of the run. The **(heu-hubs-on-locations-sub-assembly)** starts with significantly better values and remains with better values until it finally ends up with similar values as all other configurations, except the **(heu-prefer-trucks)** heuristics which started with relatively good values, but finishes worst with just minor improvements compared with the first model. All heuristics end with the same result as the **(baseline)** run or worse (see Figure 11).
The Transportation Costs on the other hand are all better compared to the **(baseline)** in the end for the applied heuristics, except for the **(heu-reduce-back-forth)**rule. All heuristics improve the Transportation Costs values over time, but only the **(heu-prefer-trucks)** rule ends up slightly worse than at the start, which is by far the lowest Transportation Costs at the beginning. The **(heu-hubs-on-location-sub-assembly)** heuristics achieves the best improvements right away from the start and finally ends at 92% compared with the baseline, only outperformed by the **(heu-prefer-trucks)** rule with 88% versus the baseline (compare Table 11). The CO_2 Costs evolution leads to three observations: First, almost all heuristics have different starting points. Second, except for the **(heu-prefer-trucks)** rule (174%, see Table11) all end in the same area, around 100% of the **(baseline)**. And third, while starting worst, the **(heu-hubs-on-location-sub-assembly)** heuristic outpaces all others with steep improvements before reaching 3.000 models.

5.3.2. Variability

Concerning the Jaccard similarities among all heuristics, diversity is higher than for the Workshare implementations, especially for the production similarities with values between 0.2 and 0.5. The transportation variability is again higher with values of 0.15 to 0.23. See Table 12 for the precise values.



(a) Evolution of the Production Costs for the tested heuristics

(b) Evolution of the Transportation Costs for the tested heuristics

Figure 20: Overview of the resulting KPIs from the domain heuristics runs

	% of models cost to last (**baseline**)		
Heuristic	Production	Transport	CO_2
last (heu-reduce-back-forth)	100%	106%	102%
last (heu-prefer-certain-locations)	100%	94%	98%
last (heu-prefer-trucks)	106%	88%	174%
last (heu-hubs-on-location)	100%	92%	105%

Table 11: The KPI percentages of each heuristic in relation to the last (baseline) model. The best results are highlighted in gray.

Heuristic	(heu-reduce-back-forth)		(heu-hubs-on-locations)		(heu-prefer-certain-locations)		(heu-prefer-trucks)	
	p	t	p	t	p	t	p	t
(**baseline**)	0.65	0.27	0.52	0.2	0.61	0.3	0.2	0.15
(heu-reduce-back-forth)	1	1	0.55	0.24	0.61	0.27	0.22	0.16
(heu-hubs-on-location)			1	1	0.43	0.19	0.15	0.11
(heu-prefer-certain-locations)					1	1	0.2	0.15
(heu-prefer-trucks)							1	1

Table 12: Similarity comparison for all heuristics, with respect to the production sites (p) and transportation means (t).

heuristic	number of models found after optimum found	36.000sec
(heu-reduce-back-forth)	no	114%
(heu-hubs-on-location)	no	71%
(heu-prefer-certain-locations)	no	102%
(heu-prefer-trucks)	no	82%

Table 13: The number of models found for the different heuristics

5.3.3. Search Performance

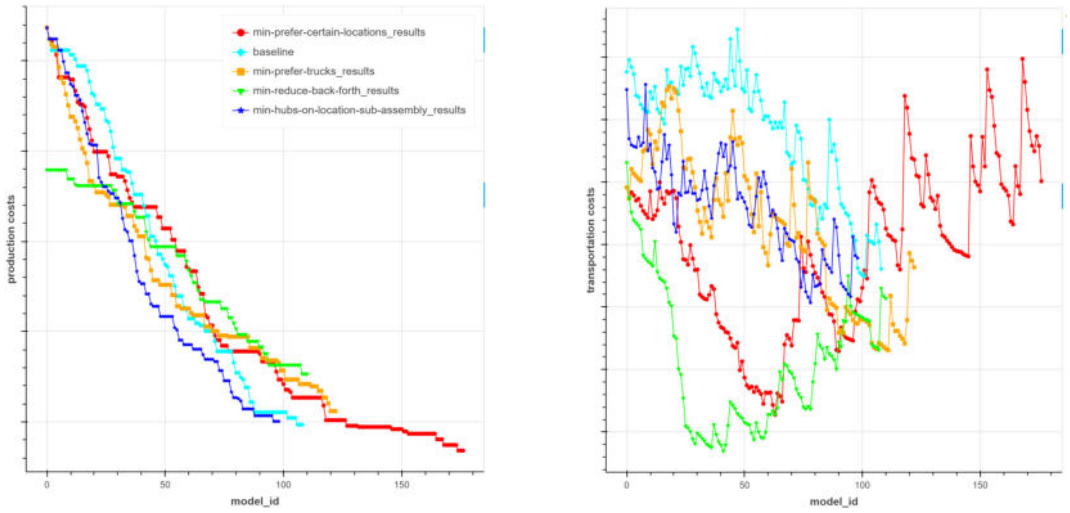
Grounding took between 0.37 and 0.38 seconds. The Search Performance is reflected in Table 13. The **(heu-hubs-on-locations)** and **(heu-prefer-trucks)** rules only find 71% and 82% of the number of models of the baseline run. **(heu-prefer-certain locations)** finds slightly more and **(heu-reduce-back-forth)** find even 14% more than the baseline. No optimum was found by any configuration.

5.4. Weak Constraints

Find the results for the tested Weak Constraints below with the usual split by KPI evolution, Variability and Search Performance.

5.4.1. KPI evolution

The evolution of the KPIs can be seen in Figure 21.
The Production Costs evolution is similar for all Weak Constraints and they achieve values between 99% and 102% compared with the **(baseline)** with **(min-prefer-certain-locations)** being the one configuration performing better than the **(baseline)** and finding more models before timeout (+32% versus (baseline)) while all other configurations find less (see Table 16). **(min-reduce-back-forth)** has a lower starting point than the others, but quickly moves in the same areas as the other configurations.
The Transportation Costs start with similar values for all configurations and end close to the baseline for **(min-prefer-trucks)** and **(min-hubs-on-locations)**. **(min-reduce-back-forth)** quickly improves and finishes best with 93% versus **(baseline)**. **(min-prefer-certain-locations)** performs worst, ending at 121% versus **(baseline)**.
CO_2 values start and evolve similarly for all configurations, except for **(min-reduce-back-forth)** which starts with higher values, but improves quickly outperforming the others before the 5000th model. **(min-prefer-certain-locations)** and **(min-prefer-trucks)** have slightly better results than the baseline, while **(min-hubs-on-location)** and **(min-reduce–back-forth)** have same or worse results (see Table 14).



(a) Evolution of the Production Costs for the tested Weak Constraints

(b) Evolution of the Transportation Costs for the tested Weak Constraints

Figure 21: Overview of the resulting KPIs from the Weak Constraint runs

	% of models cost to last (**baseline**)		
Weak Constraint	Production	Transport	CO_2
last (min-reduce-back-forth)	102%	93%	105%
last (min-prefer-certain-locations)	99%	121%	96%
last (min-prefer-trucks)	101%	100%	99%
last (min-hubs-on-location)	100%	103%	100%

Table 14: The KPI percentages of each Weak Constraint in relation to the last (baseline) model. The best results are highlighted in gray.

Heuristic	(min-red.-back-forth)		(min-hubs-on-loc.)		(min-prefer-certain-loc.)		(min-prefer-trucks)	
	p	t	p	t	p	t	p	t
(**baseline**)	0.5	0.21	0.69	0.32	0.69	0.26	0.63	0.27
(min-red.-back-forth)	1	1	0.5	0.25	0.5	0.23	0.47	0.23
(min-hubs-on-loc.)			1	1	0.67	0.27	0.59	0.28
(min-prefer-certain-loc.)					1	1	0.67	0.27
(min-prefer-trucks)							1	1

Table 15: Similarity comparison for all Weak Constraints, with respect to the production sites (p) and transportation means (t).

	number of models found after	
Weak Constraint	optimum found	36.000sec
(min-reduce-back-forth)	no	83%
(min-hubs-on-location)	no	74%
(min-prefer-certain-locations)	no	132%
(min-prefer-trucks)	no	92%

Table 16: The number of models found for the different Weak Constraints

5.4.2. Variability

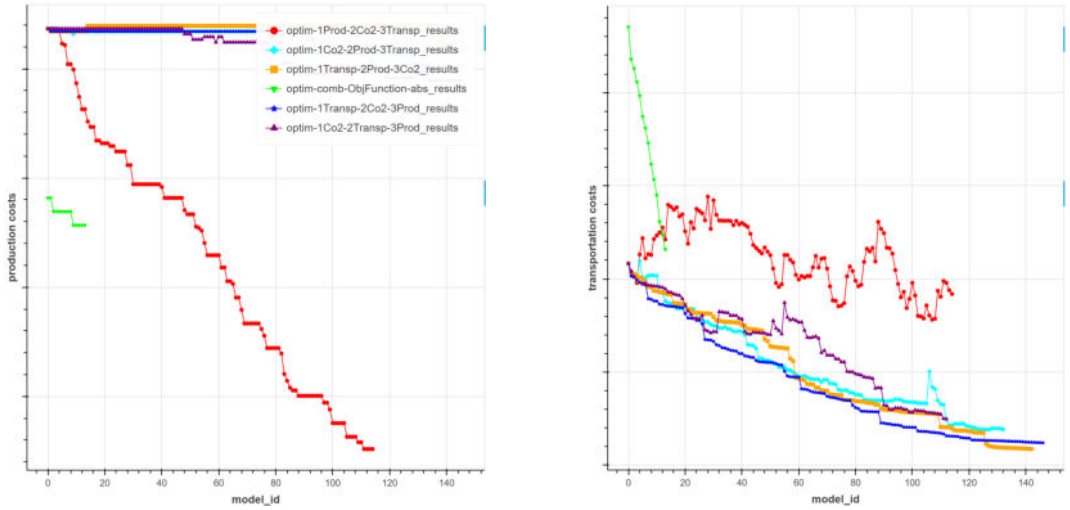
Jaccard similarities of the applied Weak Constraints move between 0.5 and 0.69 versus last (baseline) for the production and between 0.21 and 0.32 for the transportation.
The similarities among the Weak Constraint configurations are 0.47 to 0.67 for production and 0.23 to 0.27 for transportation which are relatively narrow intervals compared with the previous features, but confirming the tendency of higher variability of transportation compared with production. All figures can be seen in Table 9.

5.4.3. Search Performance

Grounding took between 0.36 and 0.4 seconds for all configurations. Table 16 shows the percentage of models found versus the **(baseline)** run. As already mentioned in the KPI evolution analysis, **(min-prefer-certain-locations)** has significantly higher number of models found while the other Weak Constraints are significantly below the (baseline) performance. No optimum was found for any run.

5.5. Optimization Statements

This section presents the Main Phase results of the different optimization statement configurations.



(a) Evolution of the Production Costs for the tested optimization statements

(b) Evolution of the Transportation Costs for the tested optimization statements

Figure 22: Overview of the resulting KPIs from the optimization statement runs

5.5.1. KPI evolution

The evolution of the KPIs as visible in Figure 22 does strongly deviate from the results of the previous trials.
The Production Costs evolution of the **(optim-1-2C-3T)** configuration is the only one with a similar behavior as the (baseline) and other configurations tested before ending up with 102% compared with it (see Table 17 for all values). The configuration **(optim-comb-Obj)** starts with lower values and is improving slowly, but there is no valid tendency to be observed as the number of models found is very low compared with all tested configurations and stops at 109% above **(baseline)**. All other configurations move almost flat regarding the Production Costs finishing at 113% or 114% of **(baseline)** costs.
The Transportation Costs again evolve similarly with continuously improving values for all configurations prioritizing Transportation or CO_2 Costs ending up with significantly lower costs for both (see Table 17) compared to the **(baseline)**. The **(optim-comb-Obj)** starts with the highest values, but shows a very steep improvement of values stopping at 109% with the timeout without showing hints of saturation. The **(optim-1P-2C-3T)** configuration show minor improvements with ups-and-downs finishing at 107% of the **(baseline)**. The CO_2 costs show similar evolutions as the Transportation Costs, but behavior of all configurations is closer among each others. All configurations end up with better values than the **(baseline)** or at least equal, i.e. for **(optim-comb-Obj)**.

	% of models cost to last (**baseline**)		
Optimization statement	Production	Transport	CO_2
last (optim-1P-2C-3T)	102%	107%	84%
last (optim-1T-2P-3C)	114%	44%	58%
last (optim-1T-2C-3P)	113%	47%	62%
last (optim-1C-2P-3T)	114%	52%	59%
last (optim-1C-2T-3P)	113%	57%	60%
last (optim-comb-Obj)	109%	125%	100%

Table 17: The KPI percentages of each optimization statement in relation to the last (baseline) model. The best results are highlighted in gray.

5.5.2. Variability

The production similarities versus the **(baseline)** are between 0.2 and 0.25 for all configurations, except the **(optim-1P-2C-3T)** configuration with 0.61. The transportation values versus **(baseline)** range between 0.14 and 0.21.
Among the optimization statements the values are between 0.21 and 0.65 for production and between 0.14 and 0.43 for transportation. Find all values in Figure 18.
To visualize the impact of the alternative optimization statements on the KPIS as well as on the variability, see Figure 23. Again a Pareto Front is implied, this time approached from different directions than previously observed.

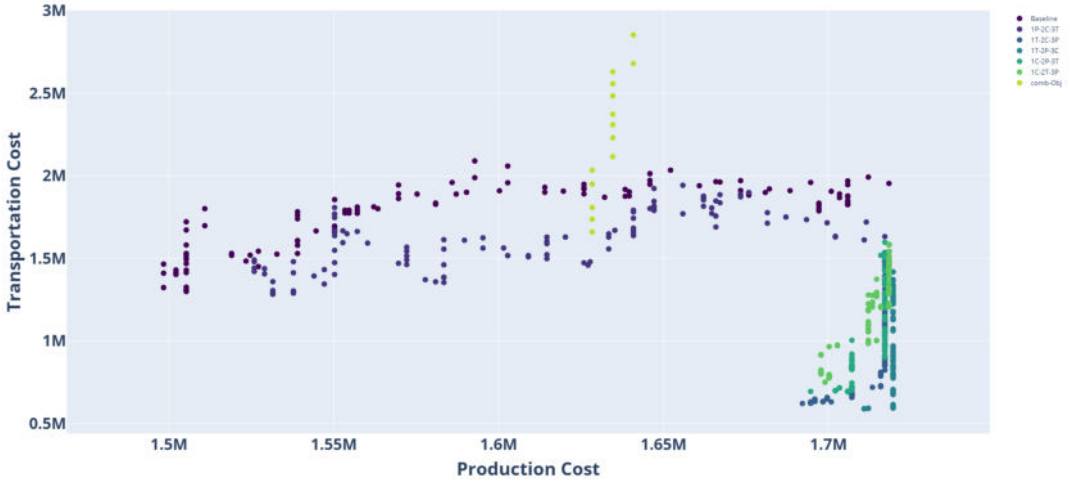


Figure 23: Evolution of the Production Costs with respect to the Transportation Costs for the tested optimization statements

Heuristic	(optim-1P-2C-3T)		(optim-1T-2P-3C)		(optim-1T-2C-3P)		(optim-1C-2P-3T)		(optim-1C-2T-3P)		(optim-comb-Obj)	
	p	t	p	t	p	t	p	t	p	t	p	t
(**baseline**)	0.61	0.21	0.2	0.18	0.21	0.2	0.2	0.17	0.2	0.16	0.25	0.14
(optim-1P-2C-3T)	1	1	0.25	0.17	0.28	0.17	0.26	0.18	0.25	0.18	0.24	0.14
(optim-1T-2P-3C)			1	1	0.65	0.43	0.57	0.31	0.61	0.32	0.25	0.17
(optim-1T-2C-3P)					1	1	0.55	0.31	0.57	0.3	0.28	0.17
(optim-1C-2P-3T)							1	1	0.51	0.31	0.21	0.16
(optim-1C-2T-3P)									1	1	0.22	0.15
(optim-comb-Obj)											1	1

Table 18: Similarity comparison for all optimization statements with respect to the production sites (p) and transportation means (t).

	number of models found after	
Optimzation statement	optimum found	36.000sec
(optim-1P-2C-3T)	no	86%
(optim-1T-2P-3C)	no	107%
(optim-1T-2C-3P)	no	110%
(optim-1C-2P-3T)	no	99%
(optim-1C-2T-3P)	no	84%
(optim-comb-Obj)	no	10%

Table 19: The number of models found for the different optimization statements

5.5.3. Search Performance

Grounding too between 0.39 and 0.41 seconds for all configuration except for the **(optim-comb-Obj)**, which took almost five minutes. Finally, the number of found models can be seen in Table 19. The only configuration with Production Costs as the top priority like the **(baseline)** configuration found 14% less models. The two configurations prioritizing the Transportation Costs are producing few more models than the **(baseline)** while the two prioritizing CO_2 produce few less. No optimum was found for any configuration.

5.6. Consolidated Result

To get the full and consolidated picture of all the tested features, all experimental results are gathered in one scatter plot and compared with trivial validation optimization runs. These validation runs were configured as described in Section 4 and are supposed to provide the lowest values that can be achieved for Production and Transportation Costs optimized exclusively while at least holding the minimum constraints of feasibility. For the best achievable costs the rule **(workshare-IC)** from the **(baseline)** configuration was removed, so that solutions were only constrained by the facts **(pProduceableAt)**. As a consequence, the resulting Production Costs re-

flect the best possible cost under realistic conditions, which is 1.442.125 (96% versus **(baseline)**. The best achievable Transportation Costs is 522.286 (40% versus **(baseline)**) accordingly.
See Figure 24 for a consolidated result with all tested features included. Every feature out of 11 is plotted with a dedicated color. Now the Pareto Front is clearly visible. The distribution of the models across the scatter plot shows the diverse penetration of the search space as only different models can produce different KPIs. The Pareto Front is approached from diverse areas of the search space into different directions. The two lines mark the results of the best Production Costs (orange) and Transportation Costs (green) from the trivial configuration as described above. It shows that the best possible Transportation Costs is almost reached and the best Production Costs very close (note, that the scale of the Production Costs axis is much finer).

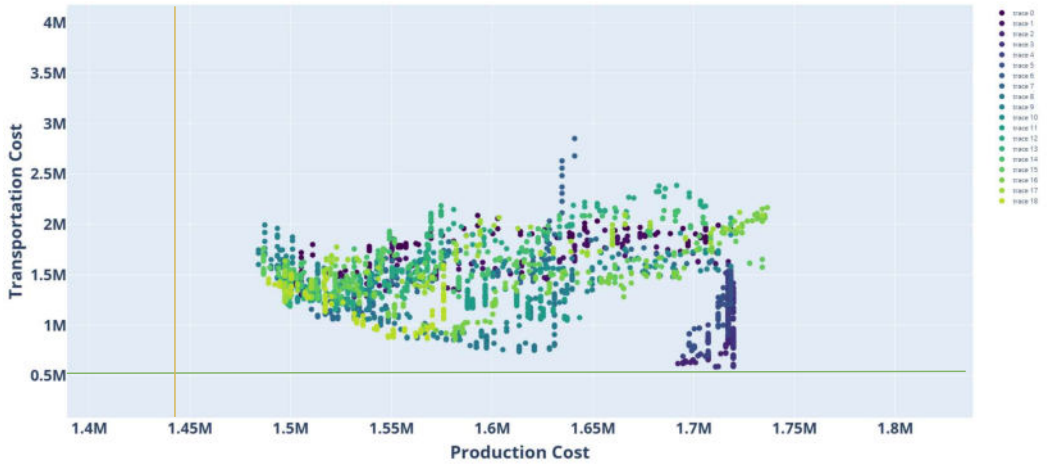


Figure 24: Evolution of the Production Costs with respect to the Transportation Costs for all tested features

Overall the best values were achieved by the rule **(min-prefer-certain-locations)** for Production Costs (highlighted in green in Table 14) and **(optim-1T-2P-3C)** for Transportation and CO_2 Costs (highlighted green in Table 17).

The results of the Main Phase will be interpreted in the next section.

6. Discussion of Main Phase

This section aims at interpreting the results presented in the previous section. It will compare the results with the hypotheses and Research Questions expressed in Section 1 and the specific expected impacts of the features to be tested in Section 3.3.
The first observation made is that, regarding the KPIs, the far biggest impact was achieved by modified weightings and prioritization of the KPIs as implemented with all optimization statements targeting Transportation or CO_2 costs (see Table 17). The improvements of these two KPIs were between 40 to 56% for all tested optimization statements while improvements for Production Costs were rarely achieved by any tested feature or only 1% versus the **(baseline)** configuration which already prioritizes Production Costs. The consolidated scatter plot of the KPIs as in Figure 24 including the different KPI weightings reinforces the visibility of a Pareto Front and demonstrates that with the use of different configurations the best achievable values for Production and Transportation Costs are almost reached for both. This and the observation that the Pareto Front is approached from different directions is providing an answer to the first Research Questions. The selection of the KPIs and the active weighting of the according optimization statements can lead to improved answers sets with regards to the objective functions, not only with better values for individual KPIs, but for many more balanced Pareto optimal solutions in between. A Pareto front is clearly indicated with higher variability of the KPIs and deeper penetration of the solution space. A higher variability of the Answer Sets is supporting this and will be described below.
The impact of the prioritization of the optimization statements seems to play a bigger role for all other means as well. This assumption is supported by looking at the KPIs evolutions for the tested features in Figures 16, 20 and 21. Regarding the Production Costs all tested features show a similar evolution with continuously improving values over time. The impact of the features is limited to accelerating or slowing down the improvements, but the general tendency remains. For the other KPIs, Transportation and CO_2 Costs the evolutions look rather random, even for those heuristics and Weak Constraints that were addressing them directly as described in section 3.3.2. Therefore the conclusions that can be drawn from the impact of the tested features on KPIs which are not prioritized could be limited while features that were targeting Production Costs, i.e. (heu-prefer-certain-locations) and (min-prefer-certain-locations) were leading to slightly better outcomes quicker as it was expected. These two rules also led to higher Search Performance reflected in more models found, especially for the Weak Constraint (+32%), while all other Weak Constraints found fewer models. This could indicate that Weak Constraints directly addressing the prioritized KPI can foster Search Performance.
The possible positive effect of Weak Constraints on Search Performance is also hinted by the tested rule **(nbSites-WC)**. Again the Weak Constraint leads to significantly more models found (+26%, see Table 10) which hints to the assumption that the Workshare constraint could be realized more smoothly than with the **(baseline)** con-

figuration and the according implementation as an integrity constraint, i.e. **(workshare-IC)**. The **(minVal-IC)** still finds +13% more models. All together these observations can support the hypothesis that Weak Constraints can guide the solver more efficiently. Nevertheless it needs to be considered that KPIs of the different Workshare configurations can not be compared directly as they modify the solution space differently.
Overall the rule **(heu-prefer-certain-locations)** had a positive impact on all KPIs while also slightly increasing Search Performance making it a good choice for general implementation. The Weak Constraint **(min-prefer-certain-locations)** produced the best Production Costs from all features and increased Search Performance the most. **heu-hubs-on-location)** produced better results for the Pareto optimization of Production and Transportation Costs. The optimization statement **(optim-1T-2P-3C)** performed best for Transportation and CO_2 Costs. The Workshare constraint implementation **(comb. minVal-IC/nbSites-WC)** produced good KPIs and still made solutions more desirable, because it incorporated important domain knowledge and is therefore recommended for all future runs.
Regarding Research Question 3, the results hint at positive impacts of the heuristics and Weak Constraints on Search Performance, especially when they target higher prioritized objective functions.
An observation worth mentioning are the results regarding the features concerning the optimization statements. As stated at the beginning of this section, only the prioritized KPI was showing the typical behavior of continuous improvement over time for all tested features, while Transportation and especially CO_2 Costs rather evolved randomly. This behavior changes for all optimization statements prioritizing Transportation or CO_2 costs, making the values continuously improving. This emphasizes that it can make sense to apply active and alternative weightings of the optimization statements in order to be able to optimize as well for the other KPIs and to generate solutions that might dominate the solutions from the **(baseline)** configuration in the meaning of a Pareto optimization as described in Section 1.2.3. It could be observed that Search Performance was highest when the optimization prioritization was on the KPI with the highest variability – both in terms of KPI values and solution candidates – which is Transportation Costs in this case. This might be a consequence of the solvers behavior when optimizing for multiple objective functions. Once it improves the value of the primary KPI, it will first optimize the lower priority KPIs before turning back to the highest priority and improve the value with one iteration. When the lower priority KPIs have significantly higher variability, this might be more time consuming for the overall optimization run. Further research is needed to investigate this hypothesis. The **(optim-comb-Obj)** configuration shows promising directions when looking at the KPI evolution curves (see Figure 21). Though, the Search Performance is low and the grounding took roughly five minutes compared to less than 0.4 minutes for all other configurations.
It is clearly visible that all features that were tested were leading to diverse Answer

Sets as reflected in the similarity comparison Tables 9, 12, 15 and 18. Throughout all features similarities of the last models were between 0.14 and 0.43 for transportation and generally higher for production from 0.2 to 0.69. As a consequence, the variability of the Production Costs KPI was also low as described earlier. If a global optimum exists, this values could also indicate how close the solutions are to that with the experiments conducted so far with a timeout of 36.000 seconds. In the case of a global optimum for Production Costs, the similarities need to reach 1.0 for different configurations. With all last models prioritizing the Production Costs, the average production similarity $\bar{J(p)}$ is relatively high, but still far from 1.0 with $\bar{J(p)} = 0.5$. This value incorporates a noticeable low production similarity of the **(heu-prefer-trucks)** configuration of only between 0.15 and 0.22 compared with the other heuristics applied. This might be the result of reducing feasible solutions to the assignment of parts to production sites when preferring truck transportation as not all sites can be connected via truck transportation. But the impact still seems higher than expected considering that still the **(workshare-IC)** rule from the **(baseline)** configuration requires all production sites to be involved in the logistics network.
The results presented earlier and the fact that production similarities average is 0.5 and transportation even below, imply that the timeout of 36.000 seconds is not sufficient to converge to any optimum and to enable deeper conclusions about the features implemented. This is emphasized by the fact that no configuration found an optimum. It can also be concluded that the search space has still not been pruned sufficiently and further constraints or even facts need to be added to achieve convergence.
Some results are unexpected, such as the fact that the rules **(heu-hubs-on-locations)** and **(min-reduce-back-forth)** start with noticeable better solutions with regards to Production Costs despite the fact that these features target Transportation Costs optimization, so an adverse impact was rather expected.
Finally, the behavior of the combined objective function implemented with the rule **(optim-comb-Obj)** is remarkable (see Figure 22). It starts with the lowest Production Costs values and the Transportation Costs improve drastically. As the combined objective function requires an additional optimization statement which includes the sum of two terms that are subject to two optimization statements themselves, the grounding times are much higher and Search Performance gets very low due to the hugely increased number of ground rules (30 million vs. 1.5 million for all other configurations). Nevertheless, the fact that the evolution of the curves look promising and a combined objective function is a way to equally weigh two KPIs it could lead to interesting results as it looks for more balanced Pareto optimal solutions than the weighted optimization statements which will look to find solutions on the Pareto Front that are minimal with regards to the highest weighted KPI. The observation of the combined objective functions behavior seems promising on much longer runtime and guided search.

Summary regarding the Research Questions:

1. yes, the active and alternate weighting of the objective functions has a major impact on the results and does produce variability in the Answer Sets, increasing the solution space penetration and finding better solutions with regards to some specific KPI, but also finding more Pareto optimal solutions on the Pareto Front. A combined objective function which weights different KPIs equally can be a good alternative for more balanced Answer Sets on the Pareto Front.

2. yes, with constraints: heuristics and Weak Constraints can help finding better solutions quicker, but only when they address the primary KPI. Otherwise their impact is weakened, but can still lead to much higher variability of Answer Sets during the search. The impact of heuristics and Weak Constraints on the overall outcome of objective functions seems moderate. But especially Weak Constraints seem to be able to increase Search Performance by guiding the search more smoothly than an Integrity Constraint could do.

3. yes, the combined use of all features can lead to better answers sets with regards to the objective functions and more variability of the solutions. The use of heuristics and Weak Constraints should be aligned with the weighting of the objective functions to find better solutions quicker.

Still, no configuration found an optimum, so the solution space was still not pruned sufficiently and the tested features were not able to provide enough guidance to converge to an optimum.
Following the results of the Main Hhase and the according discussion here, the following configurations will be tested with a 360.000 second timeout in the Final Phase:

- **(conclusion-1)**: targeting all KPIs, the best performing heuristic **(heu-prefer-certain-locations)** will be tested together with the generally implemented ones as below. the optimization statements remain as for the **(baseline)** configuration.

- **(conclusion-2)**: targeting Transportation and CO_2 Costs, the best performing rules **(heu-hubs-on-location)** and the optimization statement **(optim-1T-2P-3C)** will be applied

- **(conclusion-3)**: in order to increase search efficiency, the rule **(min-prefer-certain-locations)** will be applied together with the combined objective function **(optim-comb-Obj)** which looked promising with regards to more balanced KPIs.

All tested configurations will implement **(comb. minVal-IC/nbSites-WC)** into the baseline. The timeout will be extended to 360.000 seconds (10 times of previous experiments). These configurations follow the conclusion that the prioritization of the objective functions is a key driver of the runs' behavior and the impact of applied features is strongly dependent on it.

Find the results of these final experiments in the next section.

	% of models cost to last (**baseline**)		
final configuration	Production	Transport	CO_2
last (conclusion-1)	97%	105%	182%
last (conclusion-2)	107%	52%	152%
last (conclusion-3)	108%	96%	143%

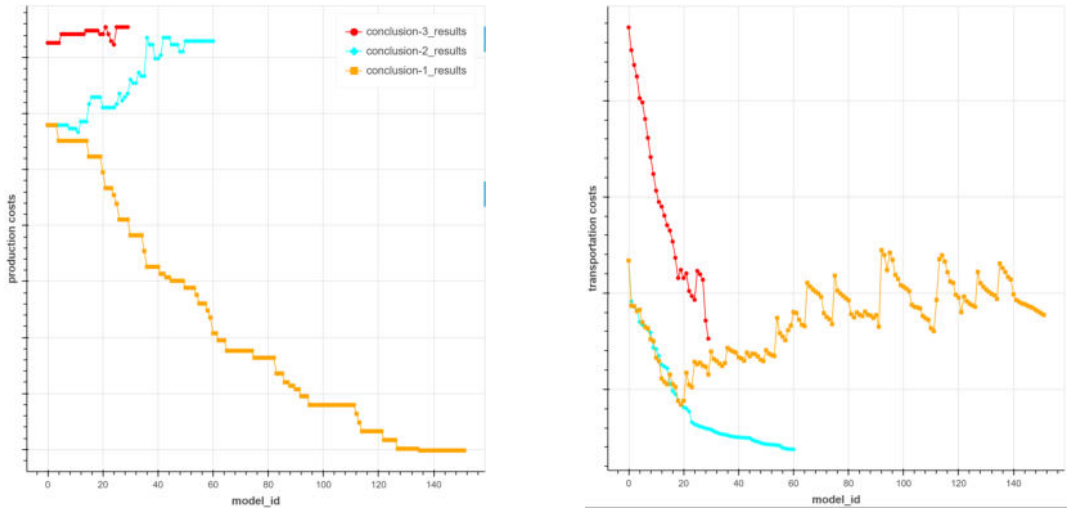
Table 20: The KPI percentages of each final configuration in relation to the last (baseline) model.

7. Results of Final Phase

This part presents the results of the conducted experiments of the Final Phase as described in Section 4.
Experiments were again carried out on an AMD EPYC 7443P 24-Core Processor, 64GB RAM type DDR4-3200. The timeout was set to 360.000 seconds. Note that in all figures, only every 100th model is plotted.

7.1. KPI evolution

Taking the results from the Main Phase, the discussion of these and the according proposed configurations to be tested in the Final Phase, find now the results of the final configurations **(conclusion-1)**, **(conclusion-2)** and **(conclusion-3)** as described at the end of Section 6.
The **(conclusion-1)** configuration which intends to further improve the Production Costs KPI was able to find the best value so far with 97% versus **(baseline)** (see Table 20 and practically finding the best possible result holding the according constraints. This is supported by the evolution of the Production Costs over time, which implies a saturation. The Transportation and CO_2 Costs are worse than for the **(baseline)** run though.
The last model of **(conclusion-2)** does not reach as good results for its main target Transportation Costs (-48%) compared with **(optim-1T-2P-3C)** (-56%) versus the **(baseline)**, but is better for Production Costs making the configuration as good at least for a Pareto optimization of these two KPIs. CO_2 costs are significantly worse. This configuration also shows first indications of a saturation as visible in Figure 25.
Finally, configuration **(conclusion-3)** with a combined objective function remains above all reference **(baseline)** KPIs, but the values are better balanced across the KPIs and the evolution of the KPIs still looks promising as still no signs of saturation are shown for Transportation Costs.



(a) Evolution of the production costs for the final configurations

(b) Evolution of the transportation costs for the final configurations

Figure 25: Overview of the resulting KPIs from the final configuration runs

final configuration	(conclusion-1)		(conclusion-2)		(conclusion-3)	
	p	t	p	t	p	t
(**baseline**)	0.57	0.22	0.21	0.13	0.21	0.13
(heu-prefer-certain-locations)	0.65	0.2			0.21	0.14
(optim-1T-2P-3C)			0.16	0.14	0.22	0.14

Table 21: Similarity comparison for all final configurations, with respect to the production sites (p) and transportation means (t).

7.2. Variability

The similarities among the final configurations and their main inspired configurations show that they still differ strongly regarding their final models (see Table 21). Production similarity between **(min-prefer-certain-locations)** and **(conclusion-1)** is relatively high with 0.65, but still distant from 1.0.

7.3. Search Performance

The number of models found over time is steadily decreasing for all configurations. The solver seems to be struggling to find new, improved models especially for **(conclusion-2)**, which found no better model anymore after the 6.000th model which was already after 1.180 minutes. **(conclusion-3)** showed a heterogeneous behavior, generally still slow in finding models, but partially with better performance and phases of longer pauses. See Table 22 for details.

	number of models found after	
final configuration	optimum found	360.000sec
(conclusion-1)	no	114%
(conclusion-2)	no	56%
(conclusion-3)	no	27%

Table 22: The number of models found for the different final configurations

The results of the Final Phase will be discussed next.

8. Discussion of Final Phase

In this section the outcome of the final configurations experiments will be discussed and whether the are able to confirm the expectations.
The **(conclusion-1)** configuration, which prioritizes the Production Costs, achieves almost the best possible values with 97% versus **(baseline)** configuration and only 1% above the best achievable value of the *(trivial-production)* configuration already after 4.306 minutes. On the other hand, the value for Transportation Costs is slightly higher (+5%) and CO_2 Cost significantly higher (+82%). A similar effect is observable for the **(conclusion-2**) configuration for the prioritized Transportation Costs KPI and the others. The values are very close to the best achievable value of the **(trivial-transportation)** configuration taking into account that all configurations incorporated the rule **(comb. baseline/minVal-IC)**, which prunes the search space, but also removes possibly better solutions with regards to the KPI.
Two conclusions could be drawn from these figures: First, that with longer runtime the setup with several optimization statements and according weightings, the top-prioritized KPI will be able to almost achieve the best possible value. Second, that this will happen at the expense of the other KPIs, finally leading to more unbalanced KPIs, which are usually not the preferred solutions when going for a Pareto optimization. The **(conclusion-3)** configuration with the combined objective function seems to produce more balanced KPIs, but still has low Search Performance, possibly due to the huge number of ground rules as described earlier. A modified formulation of the combined optimization statement could be attempted to tackle this issue.
Still, none of these configurations did find an optimum, but only showed signs of saturation close to the best possible values of the prioritized KPI for the **(conclusion-1)** and **(conclusion-2)** configurations. They were also struggling to find any new model a lot in advance of the timeout (see below). Apparently, the search space was still not sufficiently pruned, which is the main obstacle to converge to an optimum. Looking at the number of models found to get the best model, it took 15.100 models and 4.306 minutes for **(conclusion-1)** and 6.000 models and 1.180 minutes for **(conclusion-2)** to get there with the better overall results for *(conclusion-2)* (see Table 20. This reinforces the hypothesis of the possible positives impact of putting priority higher for the KPI that has higher variability.

With all the results discussed, find the Conclusion in the next section.

9. Conclusion

This final section will conclude the work in this thesis with summarizing the main contributions and proposing future work.

9.1. Summary of Contributions

With the features tested in this thesis, it was possible to produce a clearly visible Pareto Front for the primary KPIs Production and Transportation Costs and hints of a Pareto Front for all other combinations of KPIs. The best single configurations achieved KPIs that were close to the best achievable values for the according KPIs resulting from the trivial validation runs. The configurations from the Final Phase even reached near-optimal KPIs after relatively short runtime, especially considering the vast solution space.
The tested heuristics have demonstrated that they can find better solutions quicker when they are targeting the prioritized KPI of the optimization run. As expected, solutions are getting closer to configurations without the use of heuristics on the mid and long term.
Similarly Weak Constraints could unfold their impact mainly when they addressed the KPI with the highest prioritization for the optimization run, either with slightly better results or often with a higher Search Performance.
The impact of heuristics or Weak Constraints were strongly weakened, invisible or even counterproductive when not addressing the KPI with the highest priority. All this demonstrates the importance of the optimization statements and the according weighting on the outcome.
This said, the results of the tested optimization statement configurations had the biggest impact on the KPI results and it is recommended to actively manage these to achieve the best results. Combined objective functions can lead to better balanced results of several KPIs than individual, prioritized optimization statements, which tend to optimize in favor of the highest prioritized KPI at the expense of the others.
All tested features led to diverse Answer Sets. If there is an interest in exploring the solution space, the use of the tested features can help. Additionally, the diverse solution space penetration offers better visibility of the Pareto Front and provides more solutions on it which can be of interest.
None of the runs found an optimum, so the search space was still not pruned sufficiently and timeout times were still too low. Trivial test runs with single objective optimizations and simplified constraints provided lower bounds of the KPIs which showed that the obtained KPIs from the tested features are close to the optima, but failed to converge.
There are some hints that Search Performance can be improved when considering the variability of a KPI or the size of the search space that drives the according KPI. Putting higher priority to the KPI with higher variability could lead to shorter runtime.
The experiments were conducted with a single real-world dataset and confirmed

with validation runs. Due to the high complexity of the data, the IT platform and the Design of Experiments, the features could not be tested on new datasets or smaller test instances. It is recommended to do this to ensure transferability of the results. Overall this thesis can contribute to the activities of industrial architects in reducing times to find good and diverse solutions on the Pareto Front of two KPIs. It can support an iterative process to explore the solution space and find desirable configurations. Taking into account the vast solution space in this thesis, the quality of Answer Sets is impressive and makes ASP a very interesting paradigm to be followed up for industrial design optimization activities.

9.2. Future Work

The future work will tackle shortcomings and limitations of the current work, but also deal with additional capabilities that were not used for this thesis, either to ensure comparability with previous work, lacking maturity of the capabilities or due to resource and time restrictions.
No configuration and run has found an optimum, which necessitates future research to extend the timeout limit. Also, additional constraints and facts need to be added to further prune the solution space. For this, promising and desired elements of generated models can be used to set them as new facts or preferences, either as constraints or heuristics. One idea is to set intervals for the KPIs resulting from the lower boundaries gained from the **(trivial-production)** and **(trivial-transportation)** optimization statements that were described in Section 4.2.1. This way solutions with worse KPIs than the defined interval would be refused and the search space could be effectively pruned.
When testing additional features, especially such as heuristics and Weak Constraints, they should be tested together with optimization statements that prioritize the KPIs which is targeted most by the the feature to be tested. This can help that the impact of the dedicated feature is not weakened. This does not only apply to new features, but also to some features tested in this thesis.
The features should also be tested on new datasets and smaller instances to ensure transferability of the results.
Some capabilities of *clingo* were not used for this thesis, but are planned to be implemented for further research. These include the optimization framework *asprin* and the solver *clasps'* built-in option *parallel-solving*. *asprin* supports computing answers sets with preferences when optimizing with regards to some objective functions. One offered preference type is *pareto* and aims at finding Pareto optimal solutions as intended in this thesis. The option *parallel-solving* of the solver *clasp* on the other hand enables multi-threading and can exploit the resources of the used 24-Core processor hardware much more efficiently.
Finally, some work can be done on the user interface to provide helpful visualizations to the architect of the results so far, especially to browse solutions on the Pareto front.

References

[1] E. Dietz, T. Philipp, G. Schramm, and A. Zindel, "A logic programming approach to global logistics in a co-design environment," *Electronic Proceedings in Theoretical Computer Science*, vol. 385, pp. 227–240, 2023.

[2] J. R. R. A. Martins and S. A. Ning, *Engineering design optimization*. Cambridge and New York NY: Cambridge University Press, 2021.

[3] W. Faber, G. Friedrich, M. Gebser, and M. Morak, Eds., *Logics in Artificial Intelligence: 17th European Conference, JELIA 2021, Virtual Event, May 17–20, 2021, Proceedings*, 1st ed., ser. Lecture Notes in Artificial Intelligence. Cham: Springer International Publishing and Imprint Springer, 2021, vol. 12678.

[4] H. Li and R. Lachmayer, "Automated exploration of design solution space applying the generative design approach," in *Proceedings of the 22nd International Conference on Engineering Design (ICED19)*, Delft, The Netherlands, 2019. [Online]. Available: https://www.designsociety.org/publication/41896/Automated%2Bexploration%2Bof%2Bdesign%2Bsolution%2Bspace%2Bapplying%2Bthe%2BGenerative%2BDesign%2BApproach

[5] R. Arista, X. Zheng, J. Lu, and F. Mas, "An ontology-based engineering system to support aircraft manufacturing system design," *Journal of Manufacturing Systems*, vol. 68, pp. 270–288, 2023.

[6] X. Zheng, X. Hu, R. Arista, J. Lu, J. Sorvari, J. Lentes, F. Ubis, and D. Kiritsis, "A semantic-driven tradespace framework to accelerate aircraft manufacturing system design," *Journal of Intelligent Manufacturing*, vol. 35, no. 1, pp. 175–198, 2024.

[7] Airbus, "Airbus production," https://www.airbus.com/en/products-services/commercial-aircraft/the-life-cycle-of-an-aircraft/production, 2025, 20.05.2025.

[8] ——, "Airbus our worldwide presence," https://www.airbus.com/en/about-us/our-worldwide-presence, 2025, 20.05.2025.

[9] A. Falkner, G. Friedrich, K. Schekotihin, R. Taupe, and E. C. Teppan, "Industrial applications of answer set programming," *KI - Künstliche Intelligenz*, vol. 32, no. 2-3, pp. 165–176, 2018.

[10] J. Sun, J. Tang, W. Fu, Z. Chen, and Y. Niu, "Construction of a multi-echelon supply chain complex network evolution model and robustness analysis of cascading failure," *Computers & Industrial Engineering*, vol. 144, p. 106457, 2020.

[11] Q.-S. Hua, Y. Wang, D. Yu, and F. C. Lau, "Dynamic programming based algorithms for set multicover and multiset multicover problems," *Theoretical Computer Science*, vol. 411, no. 26-28, pp. 2467–2474, 2010.

[12] T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein, *Introduction to Algorithms*, 3rd ed. MIT Press, 2009.

[13] K. Thulasiraman and M. N. Swamy, *Graphs: theory and algorithms*. John Wiley & Sons, 2011.

[14] A. J. Schmitt and M. Singh, "A quantitative analysis of disruption risk in a multi-echelon supply chain," *International Journal of Production Economics*, vol. 139, no. 1, pp. 22–32, 2012.

[15] W. Hochstättler and A. Schliep, *CATBox: An interactive course in combinatorial optimization*. Heidelberg: Springer, 2010.

[16] Airbus, "Airbus transportation fleet," https://www.airbus.com/en/newsroom/stories/2023-10-building-a-lower-emission-maritime-transport-fleet, 2025, 20.05.2025.

[17] ——, "Airbus digital twin," https://www.airbus.com/en/newsroom/stories/2025-04-digital-twins-accelerating-aerospace-innovation-from-design-to-operations, 2025, 20.05.2025.

[18] Esra Erdem, Michael Gelfond, Nicola Leone, "Applications of answer set programming," *AI Magazine*, 2016.

[19] M. Gebser, R. Kaminski, B. Kaufmann, and T. Schaub, "Multi-shot asp solving with clingo." [Online]. Available: http://arxiv.org/pdf/1705.09811v2

[20] V. Lifschitz, *Answer Set Programming*, 1st ed., ser. Springer eBooks Computer Science. Cham: Springer, 2019.

[21] J. W. Lloyd, *Foundations of Logic Programming*, 2nd ed., ser. Symbolic computation. Berlin and Heidelberg and New York and London and Paris and Tokyo: Springer, 1987.

[22] M. Gebser, B. Kaufmann, and T. Schaub, "Conflict-driven answer set solving: From theory to practice," *Artificial Intelligence*, vol. 187-188, pp. 52–89, 2012.

[23] A. van Gelder, "Negation as failure using tight derivations for general logic programs," *The Journal of Logic Programming*, vol. 6, no. 1-2, pp. 109–133, 1989.

[24] M. Gelfond and V. Lifschitz, "The stable model semantics for logic programming," in *Proceedings of the Fifth International Conference on Logic Programming (ICLP)*, R. Kowalski and K. Bowen, Eds. MIT Press, 1988, pp. 1070–1080. [Online]. Available: https://www.cs.utexas.edu/~ai-lab/?gel88

[25] A. Ramos, P. van der Tak, and M. J. H. Heule, "Between restarts and backjumps," *Lecture Notes in Computer Science*, vol. 6695, pp. 216–229, 2011.

[26] M. Davis and H. Putnam, "A computing procedure for quantification theory," *Journal of the ACM*, vol. 7, no. 3, pp. 201–215, 1960.

[27] Martin Davis, George Logemann, Donald Loveland, M. Davis, G. Logemann, and D. Loveland, "A machine program for theorem-proving," *Communications of the ACM*, vol. 5, no. 7, pp. 394–397, 1961 // 1962.

[28] J. P. Marques Silva and K. A. Sakallah, "Grasp-a new search algorithm for satisfiability," *Proceedings of International Conference on Computer Aided Design; (1996) S. 220-227*, pp. 220–227, 1996.

[29] Steffen Holldobler, Norbert Manthey, Van Hau Nguyen, Julian Stecklina, and Peter Steinke, *2011 International Conference on Advanced Computer Science and Information Systems (ICACSIS 2011): Jakarta, Indonesia, 17 - 18 December 2011 ; [proceedings*. Piscataway, NJ: IEEE, 2011.

[30] R. Comploi-Taupe, G. Friedrich, K. Schekotihin, and A. Weinzierl, "Domain-specific heuristics in answer set programming: A declarative non-monotonic approach," *Journal of Artificial Intelligence Research*, vol. 76, pp. 59–114, 2023.

[31] M. Gebser, R. Kaminski, B. Kaufmann, M. Ostrowski, T. Schaub, and S. Thiele, *Potassco Guide: An Introduction to Answer Set Programming Tools*, University of Potsdam, 2015, accessed: 2025-07-12. [Online]. Available: https://potassco.org/doc/

[32] M. W. Moskewicz, C. F. Madigan, Y. Zhao, L. Zhang, and S. Malik, "Chaff," in *Proceedings of the 38th conference on Design automation - DAC '01*, J. Rabaey, Ed. New York, New York, USA: ACM Press, 2001, pp. 530–535.

[33] C. Baral, *Knowledge Representation, Reasoning and Declarative Problem Solving*. Cambridge University Press, 2003.

[34] J. Rana, P. L. Gutierrez, and J. C. Oldroyd, "Quantitative methods," in *Global Encyclopedia of Public Administration, Public Policy, and Governance*. Springer Nature Switzerland AG, 2021, pp. 11 202–11 207. [Online]. Available: https://doi.org/10.1007/978-3-030-66252-3_460

[35] G. Salton and M. J. McGill, *Introduction to modern information retrieval*, ser. McGraw-Hill computer science series. New York: McGraw-Hill Book Comp, 1983.

A. Appendix

Find all data of this thesis in the according github repository:
https://github.com/olcayHH/ASP_Master

For all codes used, see:
https://github.com/olcayHH/ASP_Master/tree/main/codes

For all experimental data, see:
https://github.com/olcayHH/ASP_Master/tree/main/experimental%20data

For supplementary figures, see:
https://github.com/olcayHH/ASP_Master/tree/main/graphics