An Assessment of Constraint Programming for Solving the Liquid Load Assignment Problem

Master thesis
Industrial Engineering & Management
University of Twente

Niels Uenk

April 27, 2009

This thesis is written to obtain the degree of Master of Science in Industrial Engineering and Management.

Institute:
University of Twente
School of Management and Governance
Department of Operational Methods for Production and Logistics

Supervisors University of Twente:
Marco Schutten
Erwin Hans

Supervisor ORTEC:
Gerhard Post

Author:
Niels Uenk

Acknowledgements

This report is the result of my graduation project at ORTEC, to obtain the degree of Master in Industrial Engineering and Management at the University of Twente. I could not have written this thesis without the help of so many people.

I thank ORTEC for giving me the opportunity to do this research project in a professional, comfortable, and motivating working atmosphere. I thank the many friends, colleagues, and family members that have contributed to this thesis, and I express my gratitude in particular to the following people.

Gerhard Post, I thank you for all the efforts that you have put into this graduation project. Your ideas and advise have helped me throughout this graduation project. You were an ORTEC colleague rather than a supervisor, and I have had a great time working with you.

I thank Marco Schutten and Erwin Hans for their guidance and input in this graduation project. Your constructive criticism has often helped me to see things in the right perspective, and the both of you have an eye for detail that has really improved this thesis. I thank Alexander Poot for investing so much time in guiding me in my work related activities at ORTEC, and I thank Frederic Leroy for helping me with the Inovia tools, and answering my Constraint Programming questions.

Marthe, I thank you for always being there for me in so many ways, for your love and support. And finally, I thank my parents. Throughout my life you have always been there for me, and supported my choices unconditionally. Thank you!

Gouda, April 2009
Niels Uenk

Contents

Summary

Introduction

ORTEC offers a variety of advanced planning and optimization products. One of these products is ORTEC Transport & Distribution (OTD), which helps planners to create an efficient transportation schedule. An important element of OTD is vehicle route optimization, commonly known as the Vehicle Routing Problem (VRP). ORTEC wishes to extend the vehicle routing functionality by adding realistic restrictions for the transportation of liquid load in vehicle compartments, and ultimately achieve the combined optimization of vehicle load and routes. Furthermore, ORTEC recently acquired the French company Inovia, and with it access to Inovia's Constraint Programming (CP) tools, a technology that is new to ORTEC. ORTEC wants to identify how it may benefit from applying Constraint Programming in its current and future logistical products.

Combining the need for the extension of the load assignment functionality and the need to gain knowledge about CP, we assess the suitability of CP for solving the liquid load assignment problem.

Algorithm requirements

For the combined optimization of vehicle load and routes, ORTEC's approach is to create *optimal* routes, while maintaining a *feasible* assignment of order load to vehicle compartments. This results in a framework in which a VRP algorithm generates vehicle routes for visiting customers, while checking the feasibility of adding customers to a route by calling a *load assignment* algorithm. This framework requires that the load assignment algorithm is very fast. Several realistic restrictions for the transportation of liquid load in vehicle compartments are not incorporated in the current load assignment algorithm. Therefore, ORTEC requires a new approach to solving the liquid load assignment problem.

Literature review

We review the literature to identify approaches for solving similar problems, and to determine criteria for the successful application of Constraint Programming. The literature does not describe (approaches for solving) the VRP with such specific load assignment restrictions. According to the literature, CP is suitable for solving highly constrained problems to feasibility. These criteria match the load assignment problem, and we focus our attention to the development of a Constraint Programming approach for solving the load assignment problem.

Constraint Programming approach

Using Inovia's CP tools, we develop a model with all of the required load assignment restrictions. This model is the basis of the CP approach, which we improve in two ways. First, we develop new global constraints to improve the solving efficiency through better constraint propagation (the mechanism that removes infeasible values from the variable domains). Testing this improvement for twelve test problems reveals that using the new global constraints reduces the computation time up to 99%. Secondly, we test various configurations of *variable* and *value choice heuristics*, improving the choice for variable instantiations. This further reduces the computation times of the CP approach for the test problems.

Benchmark development

For an objective assessment of the CP approach, we develop an ILP benchmark approach. The ILP benchmark approach provides a perspective for both the computation time and qualitative development aspects like modeling efforts and algorithm flexibility. We compare the computation times for twelve test problems of the CP approach to two ILP variants: a *benchmark algorithm* and an *equivalent algorithm*. Although Constraint Programming requires integer decision variables, this is not actually a functional requirement. Therefore, the benchmark ILP approach has continuous decision variables, as this generally leads to better results. To objectively compare Constraint Programming to Integer Linear Programming, apart from the specific requirements of the load assignment problem, we use integer decision variables in the equivalent ILP approach.

Performance comparison

Figure 1 depicts the computation times in milliseconds for the CP approach and the benchmark ILP approach for twelve realistic test problems. Figure 2 depicts the computation times for the CP and the equivalent ILP approaches. The test problems are clustered in three groups of similar parameter values. Both figures represent these clus-

ters in increasing order of parameter values. We observe two trends: (1) the CP approach performs worse for larger parameter values, and (2) both ILP approaches perform better for the larger parameter values.

For the purpose of the load assignment algorithm, fast identification of problem infeasibility is just as important as finding a feasible solution. To test the behavior in case of various degrees of problem infeasibility, we systematically increase the infeasibility of four test problems by reducing the cumulative compartment capacity in steps of 10%. Table 1 presents for all three approaches the degree to which the problem is infeasible, that is detected within 1 second. For each problem and algorithm approach it holds that if the gap to feasibility is smaller than the percentage that is indicated in the Table 1, it takes more than 1 second to detect infeasibility.

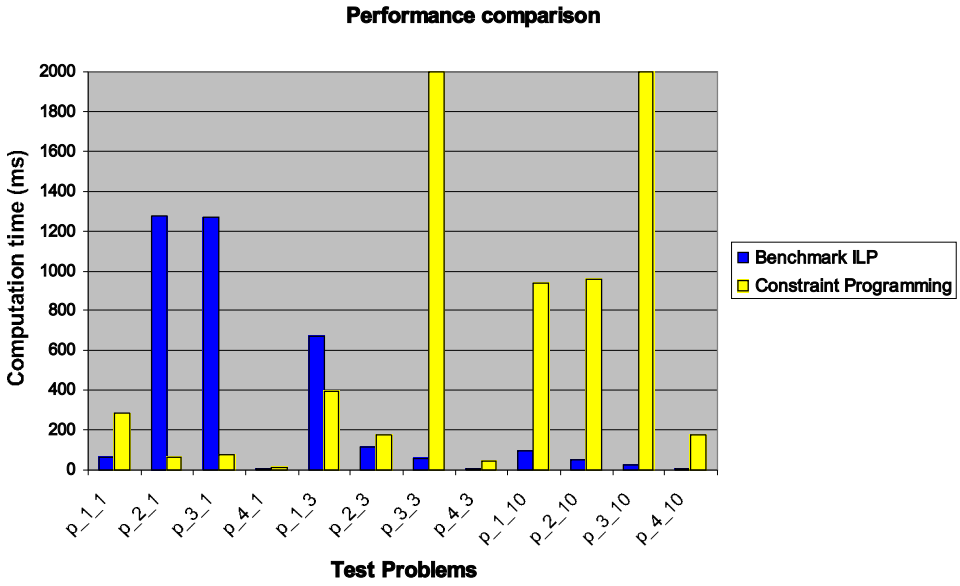

Figure 1: Computation times (ms) CP and benchmark ILP solutions

	CP	Benchmark ILP	Equivalent ILP
p_1_1	70%	20%	20%
p_2_1	50%	<10%	<10%
p_3_1	70%	<10%	<10%
p_4_1	30%	<10%	<10%

Table 1: Degree (%) of problem infeasibility that is detected within 1 second

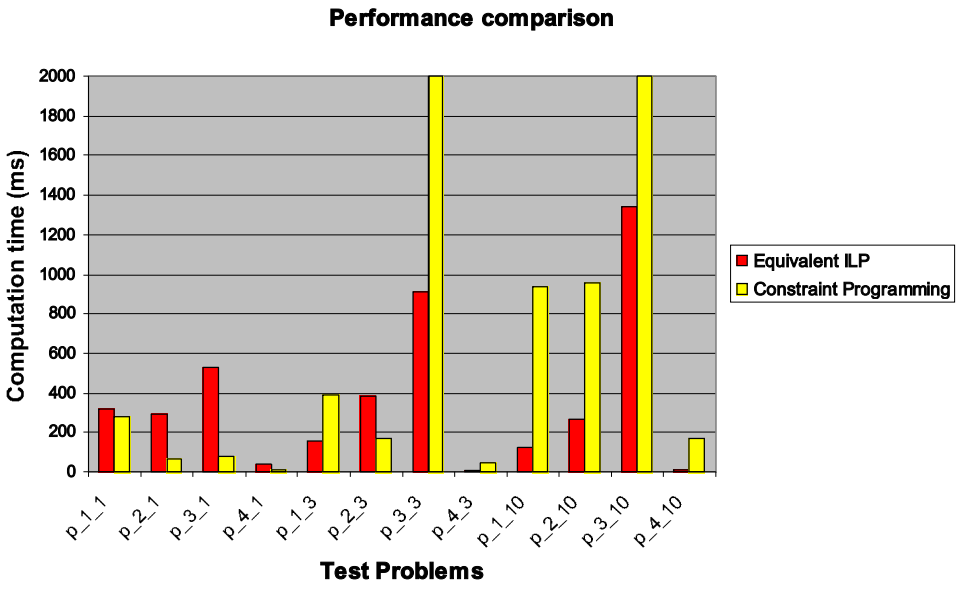

Figure 2: Computation times (ms) CP and equivalent ILP solutions

Conclusions and recommendations

The CP approach is generally slower than the benchmark ILP approach, and the equivalent ILP approach is faster for the larger problem variants. Both ILP approaches outperform the CP approach for infeasibility detection. The computation time is a weakness of the CP approach. However, all three approaches are too slow for the successful combination with a VRP algorithm. The long computation times of the CP approach relates to the size of the domains of the decision variables. Despite powerful global constraints to prune inconsistent domain values, and heuristics for 'smart' variable and value choices for instantiations, the CP approach still enumerates feasible order amounts from time to time. As order sizes increase, this enumeration becomes very inefficient. Therefore we conclude that the CP approach is not suitable for integration with the VRP algorithm to achieve the combined optimization of load and routes.

The strength of Constraint Programming is the expressiveness of the global constraints. CP allows the translation of complex problem relations to a relatively simple model, while maintaining a flexible algorithm. The solving mechanics of CP are not based on strong mathematical principles, making the computation time its major weakness, especially for problems with a large feasible region. We advise to use Constraint Programming in

situations that require a flexible algorithm, with complex relations that benefit from the global constraints, and where computation time is less important. Potentially successful applications for ORTEC are variants of the load assignment problem that do not allow load division, workforce scheduling, and tactical route planning.

Chapter 1

Context Description and Research Design

1.1 Context description

ORTEC is one of the largest providers of advanced planning and optimization software products and consulting services in the world. More than 700 employees work on products and services that result in optimized fleet routing and dispatch, vehicle and pallet loading, workforce scheduling, delivery forecasting, and network planning. ORTEC has two divisions: Logistics and Finance. ORTEC split in 2008, making both divisions formally independent. This research project takes place at ORTEC Logistics.

ORTEC recently acquired the French company Inovia, a company specialized in applying Constraint Programming to solve a variety of logistical problems. Constraint Programming is relatively new to ORTEC. Therefore ORTEC is interested in gaining knowledge about Constraint Programming in general, and wants to identify how to benefit from applying Constraint Programming in its current and future logistical products. To this end, we study Constraint Programming, by applying it to an important sub problem in one of ORTEC's main products, ORTEC Transport & Distribution.

ORTEC Transport & Distribution (OTD) is a planning system that provides the necessary tools to support planners in creating an efficient transportation schedule. It deals with matters such as time frames for loading and unloading, availability of materials and drivers, transportation costs, and (route) optimization. OTD is capable of handling large and complex scheduling problems, enabling many planners to work simultaneously on the same schedule.

A key element of the functionality of OTD is vehicle route optimization, commonly

known as the Vehicle Routing Problem (VRP). The Vehicle Routing Problem is known to be difficult to solve, being at the same time a very important model for practical issues in transportation logistics. An assumption that is commonly made in solving the VRP is that the load behaves additively: the capacity used by two sequential pickups is the sum of the sizes of the corresponding load. This assumption is not always correct in practice, since the used capacity may depend upon the way the vehicle is loaded. In particular this is the case when vehicles are divided into compartments. Then the determination of an admissible assignment of load to compartments becomes itself a difficult problem.

The current version of OTD has some functionality to support load assignment to vehicle compartments. However, OTD lacks specific restrictions for liquid load transportation, and it only supports routes in which all customer orders are picked up before the first delivery takes place.

ORTEC wishes to extend the load assignment functionality in OTD with respect to transportation of liquid load, to provide more realistic planning options to its clients. The ultimate goal is to combine the extended load assignment algorithm with the VRP algorithm, achieving combined optimization of load assignment and vehicle routes, i.e., solving the Multiple Compartment Vehicle Routing Problem (MC-VRP). The emphasis in this research project is to model and solve a realistic load assignment problem, incorporating restrictions for the transportation of liquid products in vehicle trips with mixed order pickup and delivery, using Inovia's Constraint Programming tools. We discuss all aspects of applying Constraint Programming to this problem, in comparison to the well known technology Integer Linear Programming. The actual integration of the proposed load assignment approach with the current VRP algorithm in OTD is outside the bounds of this research project.

This chapter describes the context, goals, and design of this research project. The chapter is built up as follows. Section 1.2 briefly introduces the load assignment problem. Section 1.3 describes the current load assignment algorithm in OTD. Section 1.4 introduces the required extensions of load assignment algorithm. Section 1.5 presents the goals of this project and the research questions following from these goals, and describes the approach to solving the load assignment problem.

1.2 Introduction of the load assignment problem

OTD assists planners to construct a transportation schedule, assigning customer *orders* to vehicles from a fixed heterogeneous fleet. An order consists of an amount of a product that has to be moved from one address to another. We consider the specific situation

where the products of the customer orders are liquid, and the transportation requires vehicles with *compartments* fit to carry liquids. An example of liquids transportation is the delivery of fuel to gas stations by tank trucks. Planning the transportation of one or multiple of such orders in a vehicle combination requires the assignment of the amounts of the products to vehicle compartments. This is what we refer to as the *load assignment problem*. A *load assignment* states for each order, how much of the order load is assigned to which compartment(s). Chapter 2 provides a more elaborate problem description.

1.3 Current load assignment in ORTEC Transport & Distribution

The current load assignment functionality in OTD supports the assignment of customer orders to vehicle compartments with the following restrictions. It is both possible that one order is assigned to multiple compartments and that multiple orders are assigned to the same compartment. Each compartment may only be assigned one product. The assignment of customer orders to compartments must respect for every compartment its volume capacity. Finally, the current load assignment functionality requires that all customer orders are picked up (loaded) before the first order is delivered. We refer to the problem with this restriction as the *distribution problem*, and to a model with this limitation as a *distribution model*. For the situation where orders are picked up at different addresses this requirement severely limits the amount of possible vehicle routes. In the current OTD the assignment of orders to compartments is performed by an algorithm called CompartX.

1.4 Required extensions of the load assignment functionality

The new load assignment functionality should support the following restrictions that are relevant for transporting liquid load:

- upper and lower bounds to the allowed weight on vehicle axles,
- a maximum weight capacity of the transporting vehicle besides the volume capacities of its compartments, and
- rules that require the vehicle compartments to be almost empty *or* almost full, if the vehicle is transporting hazardous products.

Also, ORTEC wants to remove the *distribution model* restriction, allowing pickups and delivers to alternate in any way, as long as an order is picked up before it is delivered.

These changes apply to the functionality of the load assignment algorithm itself. ORTEC also intends to use the load assignment algorithm in a different way. Currently, OTD *calls* the load assignment algorithm if a planner manually plans to transport a set of orders by a combination of vehicles: in such a case the VRP algorithm returns the optimal route and the current load assignment algorithm CompartX returns an assignment of orders to vehicle compartments if a feasible assignment exists. Besides this use, ORTEC aims to support the combined optimization of routes and load assignment in future versions of OTD.

To achieve the combined optimization of vehicle routes and load assignment, solving the *Multiple Compartment Vehicle Routing Problem* with restrictions for liquid load, the new load assignment algorithm will be combined with the current VRP algorithm. In such a framework, the VRP algorithm will frequently call the load assignment algorithm to verify the feasibility of adding a customer to a trip. For practical use of this combined algorithm, the allowed computation time is restricted. Therefore, the time it takes the load assignment algorithm to determine a feasible load assignment is an important factor. Throughout this report we constantly aim to optimize the speed of the proposed load assignment approaches.

Section 2.3 describes the CompartX algorithm in more detail, and we argue that the required load assignment algorithm differs too much from CompartX to simply modify it.

1.5 Research outline

1.5.1 Research objective

We identify on the one hand ORTEC's wish to gain more knowledge about Constraint Programming and find suitable applications, and on the other hand the need to develop a new load assignment algorithm. This research project aims to achieve both, and therefore we formulate the following research objective:

> The objective of this research is to assess the suitability of Constraint Programming for modeling and solving the Load Assignment problem, and to identify the criteria for successful application of Constraint Programming for ORTEC in general.

1.5.2 Goals and research questions

To meet the research objective this research consists of the following parts: (1) an assessment of the suitability of Constraint Programming for solving the Load Assignment problem based on a detailed problem analysis and a literature review, (2) an in depth analysis of Constraint Programming by applying it to solve the load assignment problem, (3) an assessment of the Constraint Programming approach for solving the load assignment problem by comparing it to a benchmark technology, and (4) an analysis of the suitability of Constraint Programming for practical applications within ORTEC. For each part we formulate one or multiple goals and research questions.

The first goal is to provide a formal description of the load assignment problem, and to analyze the relevant aspects of the context of the problem. Besides the formal problem description, we formulate an ILP model for the liquid load assignment problem, based on an ILP model of the *distribution problem*. This goal leads to the following question:

1. How should the ILP model for the distribution problem be modified to represent the liquid load assignment problem?

Chapter 2 presents the formal problem description, a detailed context description, and an ILP model of the load assignment problem. Furthermore, we argue why we choose to develop a new load assignment algorithm instead of modifying CompartX.

Experts in the field of algorithm development within ORTEC believe that Constraint Programming might be a suitable approach for solving the load assignment problem. Constraint Programming is an exact method, suitable for solving highly constrained problems to feasibility (instead of optimality), and is often used for assignment problems. To identify common approaches for comparable problems, and to assess the suitability of Constraint Programming for the load assignment problem, we review the literature.

The second goal is to identify which algorithmic approaches (heuristics, ILP, Constraint Programming) in general are most suitable for solving comparable routing and load assignment problems, which leads to the question:

2. How does the literature propose to solve comparable routing and load assignment problems?

The third goal is to determine whether the characteristics of the load assignment problem are appropriate to apply Constraint Programming. This goal leads to the following question:

3. Is Constraint Programming a suitable technique for modeling and solving the load assignment problem?

And the sub questions:

   (a) Which criteria determine the suitability of Constraint Programming?
   (b) Does the load assignment problem meet these criteria?

Chapter 3 presents the outcome of the literature review, answering research questions 2 and 3.

ORTEC wants to gain knowledge about problem solving with Constraint Programming, especially with the Inovia tools, and how Constraint Programming differs from other techniques such as Integer Linear Programming. The fourth goal of this research project is to model and solve the load assignment problem with Constraint Programming. This goal entails exploring modeling possibilities, developing and implementing a Constraint Programming model, and testing the computational speed of the CP approach. This goal leads to the following research question:

4. How can the load assignment problem be solved with the Inovia Constraint Programming tools, and how does the solution perform?

And the sub questions:

   (a) How can the load assignment problem be modeled?
   (b) How fast is the Constraint Programming approach in terms of the required computation time for test problems?
   (c) How can we improve the speed of the Constraint Programming load assignment approach?

Chapter 4 describes the development of a Constraint Programming approach for solving the load assignment problem, and presents the results of solving various test problems.

To objectively assess the Constraint Programming approach, we develop a benchmark algorithm using a well known technology. The fifth goal is to compare Constraint Programming to Integer Linear Programming with respect to both qualitative aspects such as modeling flexibility and ease of use, and the computational speed. We formulate the following question:

5. What are the advantages and disadvantages of modeling and solving a problem with Constraint Programming compared to other methods such as ILP?

And the sub questions:

(a) How should the current ILP approach for the distribution problem be modified to solve the liquid load assignment problem?

(b) How does the computational speed of the ILP approach compare to the speed of the CP approach?

(c) What are the qualitative differences between Constraint Programming and Integer Linear Programming?

Chapter 5 presents an ILP approach for solving the load assignment problem, discusses the test results of this algorithm, and compares both qualitative and quantitative aspects of the CP and ILP approaches.

The sixth goal of this research project is to determine opportunities for ORTEC to apply Constraint Programming. Chapter 6 answers the following question:

6. For which type of applications might Constraint Programming be appropriate, and what are concrete opportunities for ORTEC?

To conclude this report, Chapter 7 presents the conclusions of this research project.

Chapter 2

Problem Description

Chapter 2 presents the complete description of the load assignment problem introduced in Section 1.2. Section 2.1 provides definitions and examples of the concepts that we frequently use throughout this report. Section 2.2 presents the complete problem description and the assumptions. Section 2.3 describes the functionality of CompartX. Section 2.4 presents an Integer Linear Programming formulation of the load assignment problem.

2.1 Frequently used concepts

We use some concepts very frequently throughout the report in the discussion of the problem and the approaches for solving the problem. This section provides a definition for these concepts to prevent any confusion about what we mean.

Order

An *order* consists of a certain amount of one product, required to be moved from a starting (pick up) address to a finish (delivery) address. To execute an order, a vehicle should pick up the total amount of the order product at its pick up address, put the product in one or more of its compartments, drive to its finish address, and deliver the total amount of product. The relevant information for one order is the amount of product, and its pickup and delivery addresses. Figure 2.1 displays an example of four different orders.

Resource, resource combination, and configuration

We refer to truck, trailer, and driver as *resources*. We refer to an arrangement of resources used to transport orders as a *resource combination* (see Figure 2.2 for an example). Resources will have a fixed arrangement of compartments, which we refer to as the *configuration*. The configuration has a weight capacity. Furthermore, each compartment

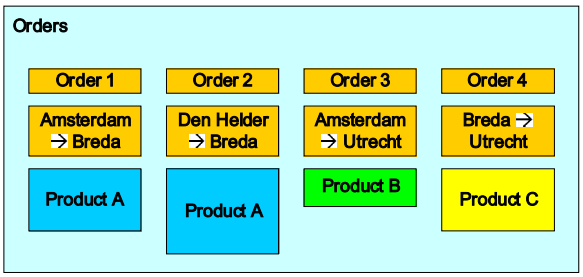

Figure 2.1: Example of 4 orders

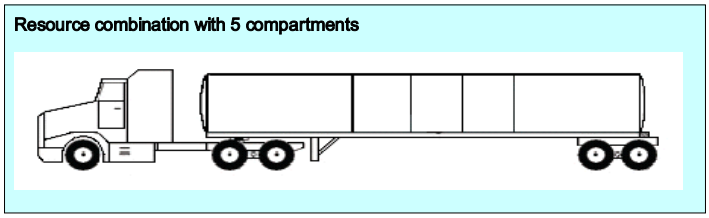

Figure 2.2: Example of a truck with a 5 compartment trailer

has a volume capacity. Often, the complete arrangement of resources is not relevant for the discussion as we are only interested in the associated compartments that can hold products. One exception is a resource combination in which both a truck and a trailer have compartments. Figure 2.3 depicts an example of such a resource combination. For this case the arrangement of resources is relevant with respect to the weight contribution of compartments to resource axles: load in compartments of configuration B does **not** contribute weight to the axles carrying configuration A. These resource combinations are not common in Europe, but occur frequently in larger countries such as the USA and Australia.

Trip, stop, and section

A trip consists of one or multiple orders, scheduled to be transported by a resource combination. Basically a trip is a sequence of *stops*. Each stop occurs at a different

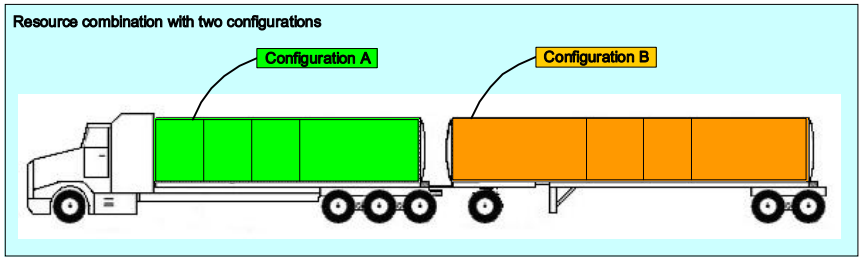

Figure 2.3: Example of a resource combination with two configurations

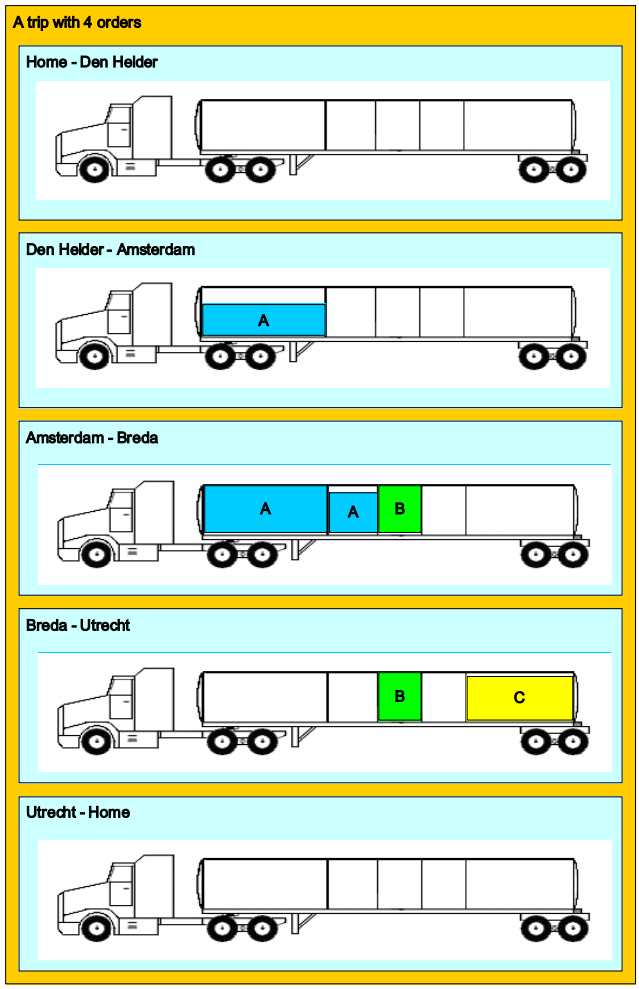

Figure 2.4: Example of a trip with 4 orders

address, and at each stop, one or multiple orders are picked up, delivered, or both. Order pickups and deliveries may alternate as long as the delivery of a specific order is scheduled after the pickup of that order. Following the example of the 4 orders in Figure 2.1, a resource would make the trip 'home - Den Helder - Amsterdam - Breda - Utrecht - home'. Figure 2.4 shows a *possible* state of the resource after each pick up and delivery. Notice that two orders (both of product A) are delivered before the fourth order (of product C) is picked up. Applying the distribution model, the truck would have been sent to Breda to pickup order 4 first, before delivering all orders. The trip is divided in *sections*: one section between every pair of consecutive stops, representing each combination of orders on board of the resource combination. Figure 2.5 indicates the different sections on a map for the trip from the previous example, where the home-location is Haarlem. Intersected lines indicate the resource-combination is empty, non-intersected lines indicate the resource combination is carrying load.

Figure 2.5: Example of stops and sections in a trip

Load

We refer to the amount(s) of product(s) of one or multiple orders as *load*.

2.2 The liquid load assignment problem

This section describes in detail the background of the relevant aspects of the problem.

2.2.1 Problem formulation

If an order is planned in a trip, the load is assigned to the compartments of the resource combination. Hence, the starting point for load assignment is:

- A set of orders to be transported in a resource combination
- A set of compartments to which the products of the orders have to be assigned

The objective is to obtain a **feasible** load assignment. We do not consider any costs.

Before any amount of load is assigned to a compartment, the order - compartment compatibility needs to be checked. This compatibility applies to the entire trip. We allow scheduling a new trip with a resource combination that has not been cleaned after the previous use. Also, we allow scheduling the *remainder* of a trip, meaning that the planner has already manually assigned one or multiple orders to the resource compartments. Therefore, the order - compartment compatibility depends on the following aspects:

- *The cleaning history of the compartment:* if a compartment was not cleaned after previous use, it is only compatible with orders of the same product that was previously assigned to the compartment. We assume that no cleaning takes place within a trip.
- *If the planner has fixed (part of) an order to a compartment.* it cannot be moved to another compartment to optimize the load assignment anymore. By fixing an order in a compartment, the planner forces this order to be assigned to that compartment. Note that it **is** allowed to put other orders in the compartment, as long as all other restrictions are satisfied.
- *The current load pre-assigned to the compartment*: it is only allowed to assign load of multiple orders to the same compartment if they are the same product, even after the current order has been delivered (due to contamination). The difference between pre-assigned load and fixed orders, is that pre-assigned load is already actually present in the resource combination before picking up the first order from

the set of orders that are to be scheduled, while an order fixed in a compartment must be part of the set of orders that are to be scheduled. Hence, at the start of the trip, pre-assigned load is already in the resource combination, fixed orders are not.

- For *Single Order* compartments, it is not allowed to put load of different orders together in that compartment, even if the product is the same. The previous two aspects apply to Single Order compartments as well, meaning that orders can only be assigned to Single Order compartments in sections where these compartments do not contain fixed or pre-assigned orders.

- *Allowed maximum temperature of the compartment:* the loading temperature of the product may not exceed the allowed maximum temperature of the compartment. The maximum temperature of a compartment is independent of neighbor-compartments. We do not consider restrictions with respect to temperature differences in adjacent compartments.

- *The resource availability at the pickup and the delivery location:* the schedule may require a trailer to be decoupled at some point in the trip, making loading the trailer for pickups or unloading the trailer for deliveries impossible.

Note that these aspects only indicate whether an order is allowed in a compartment, not whether it fits. These checks result in a compatibility matrix, which is used (as input) to make the actual load assignment. For the actual load assignment, the following aspects are considered.

- The orders can be divided over several compartments.

- The compartment (residual) capacity should not be exceeded by the amount of assigned product. This capacity is measured in volume units.

- For *Single Order* compartments, in each trip section only one order can be assigned to these compartments.

- The total weight on each wheel-axle of the resource should not exceed its minimum or maximum allowed weight. A violation of minimum axle weight may occur if load assigned to a compartment located behind the last axle makes the resource tilt (see Figure 2.6).

- The resource weight capacity should not be exceeded. This restriction may seem redundant at first, as there is already a volume capacity per compartment. However, it is possible that the summed weight of the load in the compartments exceeds the weight capacity of the whole resource.

Figure 2.6: Possibility of vehicle tilting

- Apart from the maximum capacity, two other volume restrictions may be relevant for a compartment: in case the product is dangerous, the compartment should either be almost empty or almost full. This means the load should be either smaller than the 'maximum almost empty percentage' or larger than the 'minimum almost full percentage'. These are referred to as *ullage rates* and prevent dangerous shifting of load in case of dangerous liquids transport. Some companies apply the ullage rate rules to all liquid products, whether they are considered dangerous or not.

The current load assignment functionality does not incorporate the last three issues. This, and the removal of the distribution model restriction, constitute the required functional extensions. An effect of these changes is that the feasibility of the load assignment has to be checked for each trip section. Altogether this means that CompartX in its current form is not sufficient anymore. Section 2.3 explains why we do not redesign CompartX to incorporate the new functionality.

2.2.2 Optimization criterion

The main goal of load assignment is to obtain a feasible solution in very little computation time. Any solution that satisfies all constraints is acceptable and there is no optimization criterion.

2.2.3 Assumptions

We will restrict the calculation of load assignment to the case that the (relevant) resources are fixed for the trip. This means that the compartment configuration does not change within a trip and neither do the weight restrictions on the wheel axles.

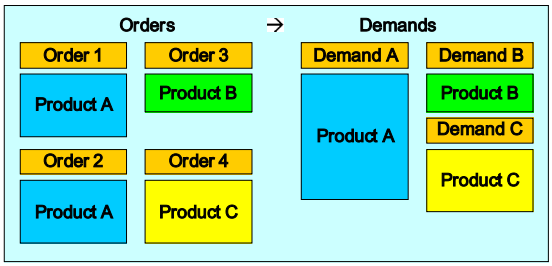

Figure 2.7: Orders converted to demands

2.2.4 Required computation time

For the successful combination of the VRP and load assignment, the computation time required to determine a feasible load assignment is an important factor. Testing and improving the performance of the algorithmic approaches is an important goal of this research project.

2.3 The current load assignment functionality

To describe the context of the current load assignment functionality, two *subsystems* of OTD are relevant: the COMTEC Distributed Calculation System (CDCS) and the COMTEC Optimization System (COPS). CDCS manages the actual schedule of planned trips and all data that is necessary to maintain a feasible schedule. COPS is the subsystem that performs all necessary optimizations and assignments such as shortest path calculations and load assignment.

Typically, CDCS and COPS interact for various planning actions. Consider for example planning a new trip in which several orders are picked up at one location and delivered at their destination. The planner chooses one or multiple orders and a combination of a truck and trailer to transport the orders. CDCS then gathers all relevant data, wraps it up in a command, and sends it to COPS to come up with a feasible load assignment. COPS either returns a feasible load assignment, or a warning that there is no feasible load assignment. CompartX only supports situations in which all order pickups are planned before the first order delivery.

To illustrate why it is not trivial to add the new functionality to CompartX, this section provides a short description of the algorithm. The input for CompartX consists of a set of orders, a set of compartments, and the compartment-order compatibility matrix, subject to the same rules as mentioned in Section 2.2. CompartX converts orders in demands of equal products, summing the quantity of each product as depicted in Figure

2.7. In CompartX a Dynamical Programming algorithm then assigns these demands to compatible resource compartments, dividing the volume of a demand over several compartments if necessary. If no compartment capacity is violated, CompartX returns the assignment of demands to COPS, which in turn converts it to an assignment of separate orders to compartments. The assignment CompartX returns to COPS states for each compartment whether or not it is used for a demand. Figure 2.8 depicts the current load assignment function and interface in OTD.

CompartX only checks the assignment of *demands* to compartments, basically checking the situation after the last pickup in the distribution model. CompartX does not consider every unique combination of orders that is present in the resource at some time the resource is driving, while ullage rates and axle weights actually depend on the assigned *amounts* of these specific order combinations. Furthermore, CompartX is not capable of dealing with pickups after deliveries: CompartX considers the load of all orders as actually loaded at the same time. This makes that adding the new functionality to CompartX would require such a radical redesign that we do not consider this option.

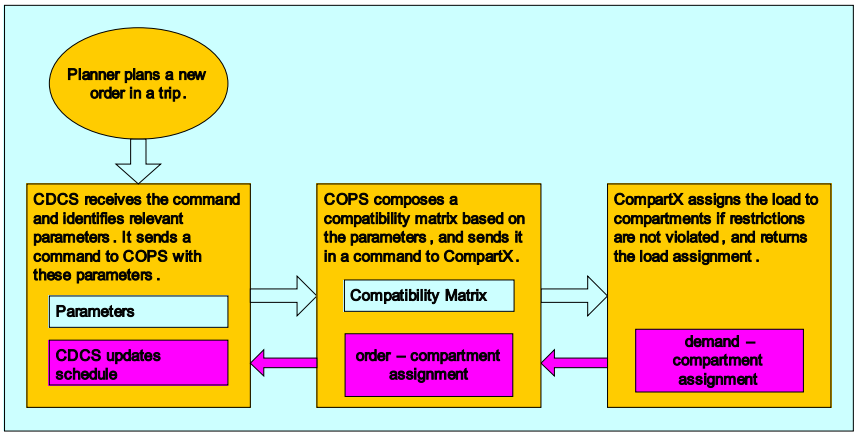

Figure 2.8: Interface between CDCS, COPS and CompartX

2.4 ILP problem formulation

The problem of assigning load to compartments of resources used in trips is captured by the following ILP model, derived from the current OTD ILP prototype for the *distribution model*.

Some of the constraints have to be satisfied for every combination of load that is actually transported by the resource combination. This is modeled by dividing the trip in sections, where a new section starts whenever the set of orders on board of the resource

combination changes. All sections together therefore cover all different order combinations that are on board at some time of driving.

Sets

Orders	***Set O (index i)***
Regular Compartments	***Set C^R (index j)***
Single Order Compartments	***Set C^S (index j)***
Compartments	***Set $C = C^R \cup C^S$ (index j)***
Axles	***Set A (index k)***
Configurations	***Set F (index l)***
Sections	***Set S (index m)***

Parameters

$Demand_i$	Requested amount of order i (liters)
$MaxVol_j$	Maximum capacity of compartment j (liters)
$MaxAlmostEmpty\%_j$	Maximum percentage to which compartment j may be filled when almost empty: lower ullage rate
$MinAlmostFull\%_j$	Minimum percentage to which compartment j should be filled when almost full: upper ullage rate
$MaxWeightAxle_k$	Maximum weight allowed on axle k (kilograms)
$MinWeightAxle_k$	Minimum weight allowed on axle k (kilograms)
$MaxWeightConfig_l$	Maximum capacity of configuration l (kilograms)
$Weight_i$	Weight of 1 liter of order i (kilograms)
w_{jk}	Contribution of the weight of the load in compartment j to the total weight on axle k
$CompInConfig_{jl}$	Binary parameter: 1 if compartment j is part of configuration l, 0 otherwise
$OInS_{im}$	Short for $OrderInSection_{im}$; Binary parameter: 1 if order i is transported in section m, 0 otherwise

Decision Variables

x_{ij}	Quantity (in liters) of order i assigned to compartment j

Auxiliary Variables

$$\delta_{ij} = \begin{cases} 1 & \text{if (part of) the order } i \text{ is assigned to compartment } j \\ 0 & \text{otherwise} \end{cases}$$

$$z_{jm} = \begin{cases} 1 & \text{if compartment } j \text{ in section } m \text{ is loaded above the ullage rate upper bound} \\ 0 & \text{if compartment } j \text{ in section } m \text{ is loaded below the ullage rate lower bound} \end{cases}$$

Objective function

We are only interested in finding a feasible solution, therefore we do not specify an objective function.

Constraints

Deliver the amount the customer requires:

$$\sum_{j \in C} x_{ij} = Demand_i \qquad \forall i \tag{2.1}$$

If any (non-zero) amount of order i is assigned to compartment j, δ_{ij} should be 1:

$$x_{ij} - min\{Demand_i, MaxVol_j\} \cdot \delta_{ij} \leq 0 \qquad \forall i, j \tag{2.2}$$

For single-order compartments, in each section only one order may be assigned to the compartment:

$$\sum_{i \in O} \delta_{ij} \cdot OInS_{im} \leq 1 \qquad \forall m, j \in C^S \tag{2.3}$$

Respect compartment capacity:

$$\sum_{i \in O} x_{ij} \cdot OInS_{im} \leq MaxVol_j \qquad \forall j, m \tag{2.4}$$

The load in compartment j should be less than the lower ullage rate:

$$\sum_{i \in O} x_{ij} \cdot OInS_{im} - MaxVol_j \cdot z_{jm} \leq MaxVol_j \cdot MaxAlmostEmpty\%_j \qquad \forall j, m \tag{2.5}$$

or:
The load in compartment j should be larger than the upper ullage rate:

$$\sum_{i \in O} x_{ij} \cdot OInS_{im} + MaxVol_j \cdot (1 - z_{jm}) \geq MaxVol_j \cdot MinAlmostFull\%_j \qquad \forall j, m \quad (2.6)$$

Load may not exceed the maximum weight capacity on axle k:

$$\sum_{j \in C} w_{jk} \sum_{i \in O} x_{ij} \cdot OInS_{im} \cdot Weight_i \leq MaxWeightAxle_k \qquad \forall k, m \quad (2.7)$$

Load should exceed the minimum weight allowed on axle k to prevent tilting of a resource (see Figure 2.6):

$$\sum_{j \in C} w_{jk} \sum_{i \in O} x_{ij} \cdot OInS_{im} \cdot Weight_i \geq MinWeightAxle_k \qquad \forall k, m \quad (2.8)$$

The total weight in the configuration should be less than configuration capacity:

$$\sum_{j \in C} \sum_{i \in O} x_{ij} \cdot OInS_{im} \cdot Weight_i \cdot CompInConfig_{jl} \leq MaxWeightConfig_l \qquad \forall l, m \quad (2.9)$$

The following constraints represent the order/compartment compatibility matrix:

$$\delta_{ij} = 0 \qquad \text{For all orders i and compartments j that are not compatible} \quad (2.10)$$

Furthermore, for orders i and i' that cannot be put in the same compartment because of different products, the following constraint applies:

$$\delta_{ij} + \delta_{i'j} \leq 1 \qquad \forall j \quad (2.11)$$

Chapter 3

Literature Review

The literature review serves two goals for this research project. The first goal is to identify which realistic (multiple compartment) Vehicle Routing Problems have already been investigated, and how they are solved. The second goal is to determine whether Constraint Programming is a suitable technique for solving the load assignment problem considered in this research project. This chapter consists of two sections. Section 3.1 presents an overview of the most relevant articles on Multiple Compartment Vehicle Routing Problems (MC-VRP) and other realistic Vehicle Routing problems. The VRP is a very well known and often described problem. Realistic applications often require extending the problem, for instance adding details like compartments. The amount of these extended problems described in the literature is limited.

Section 3.2 mentions some applications of Constraint Programming (CP). Many recent articles describe the application of CP to various vehicle routing problems, most of which combine CP with a different heuristic or programming method such as (M)ILP. The focus of this part of the literature review is on identifying the typical problem characteristics for which CP is suitable and characteristics for which CP is less suitable. Based on these two sections we draw conclusions whether CP is a promising technique to solve the load assignment problem.

3.1 Realistic Vehicle Routing Problems

According to our knowledge, the MC-VRP with the various realistic constraints described in this report has not been addressed in the literature so far. Only a few authors have addressed the MC-VRP or other variants of the VRP, modified to more closely resemble a real-life application. This section describes the most similar or relevant (MC-)Vehicle Routing Problems addressed in the literature.

Repoussis et al. (2006) describe the following problem: a heterogeneous fixed fleet of depot returning vehicles with multiple storage compartments transports different commodities to a set of customers of known demand within an interval during which delivery has to take place. Each customer is served only once by exactly one vehicle. The objective is to design a minimum cost fleet mix using the available fleet composition following routes of minimum distance, such that vehicle (and compartment) capacity, loading and time window constraints are satisfied. The only compartment related restrictions used in this model are the capacity restriction and that only one type of product can be assigned to a compartment.

Repoussis et al. propose a hybrid meta-heuristic which employs Greedy Randomized Adaptive Search Procedures (GRASP) for diversifying local search and Variable Neighborhood Search (VNS) for intensifying local search. The greedy randomization mechanism of GRASP is characterized by a dynamic constructive heuristic, adding new elements that are randomly picked from a candidate list to a partial incomplete solution. Subsequently, the feasible solution found in each iteration is subject to local search. This overall procedure is repeated until some termination conditions are met. While the GRASP procedures diversify the search, VNS is used as an improvement heuristic that is tuned to intensify the local search. The proposed hybrid meta-heuristic has been tested with benchmark data sets.

Fallahi et al. (2008) also describe the MC-VRP, where each compartment is dedicated for only one product or a product group, sharing some common characteristic that necessitates separate storage such as loading temperature. Each product required by a customer must be delivered by only one vehicle, but the different products ordered by one customer may be delivered by different vehicles. The objective is again to minimize total distribution costs, satisfying all constraints.

Fallahi et al. propose two methods to solve the problem: a *memetic algorithm* either or not improved by a *path relinking* algorithm, and a *tabu search* algorithm. We briefly describe these three concepts next.

A *memetic algorithm*, sometimes referred to as genetic local search, is a genetic algorithm hybridized with a local search procedure. The algorithm produces an initial *population* Π of $\boldsymbol{n}$ well diversified solutions, sorted in ascending cost order. Until some stopping criteria are met, in each iteration of the memetic algorithm two of these solutions are randomly picked and combined to form a new *child* solution. With a certain probability, the solution is improved by a local search. If the resulting solution is better than $\Pi_{\boldsymbol{n}}$, the child replaces solution $\Pi_{\boldsymbol{k}}$, where integer $\boldsymbol{k}$ is randomly selected from $[\boldsymbol{n/2}, \boldsymbol{n}]$.

Path relinking is a post-optimization procedure that starts with a small pool of high-quality solutions, generating a *path* between each pair (A, B) of solutions from the solution pool. A path is a sequence of solutions $(S_0 = A, S_1, ..., S_n = B)$ in which A is transformed into B by progressively introducing elements of solution B. In general, local search must be applied to the new solutions $S_1, ..., S_{n-1}$, otherwise improvements may be incremental.

Various strategies are available to update the solution pool. Fallahi et al. construct paths from A to B and from B to A for every disjunct pair of solutions (A, B). Evaluating every pair of paths, if the best solution C of both paths improves the best solution S found in previous pair of paths, $S = C$. After exploring all paths, if S improves the current worst solution in the pool, S replaces this solution. The algorithm stops if no better solution is found.

Tabu Search is a meta-heuristic exploring the solution space by moving in each iteration from the incumbent solution S to the best solution S' in its neighborhood $N(S)$, even if solution S' is worse than solution S. To avoid coming back to recently visited solutions, the m last visited solutions are put on a tabu list. Moving to these solutions is not allowed.

Fallahi et al. test these methods on twenty well-known instances for VRP, which have been transformed to MC-VRP instances for vehicles with two compartments by splitting the requests of customers in two amounts. The test results are compared to the results of classical VRP solutions.

Chajakis and Guignard (2003) discuss the MC-VRP and mention two applications: transporting liquids (or gas) as in the oil industry, where compartments are necessary to avoid mixing the load, and delivering products to convenient stores, where compartments are used to keep products at different temperatures, also mentioned by *Fallahi et al.* (2008). The authors only mention the necessity of compartments for liquid load transport. Test problems of the convenient store delivery problem are solved using Lagrangean relaxations of the integer problem formulation.

Abdelaziz et al. (2002) describe a MC-VRP (without time windows) for delivering liquid fuels, resembling the problem addressed in this report. Starting from some initial solution, they propose a Variable Neighborhood Search (VNS) algorithm to obtain a near-optimal solution. The heuristic is tested with only one small problem instance and nothing is mentioned about running time.

Semet and Taillard (1993) tackle another real life VRP, describing a delivery problem to 45 Swiss grocery stores with 70 to 90 orders, occupying a heterogeneous fleet (without compartments) with volume and weight loading restrictions, time window restrictions, truck/trailer compatibility restrictions and grocery store/truck-trailer compatibility restrictions. The objective is to minimize distribution costs. An initial feasible solution is obtained with a constructive heuristic based on that of *Fisher and Jaikumar* (1981), first clustering customers and then solving a TSP for each cluster. To improve the solution, a Tabu search heuristic is applied, in which an iteration consists of removing a customer from one trip and adding it to another. A dummy trip contains orders that are not assigned to a real trip. (including orders in a *dummy trip*, containing orders that are currently not assigned to any existing trip). Several tests are reported, changing the Tabu search parameters and some other model variants. The running times of these tests vary from approximately one to ten minutes.

The algorithms mentioned above all extend the classical VRP with some characteristics to make the model applicable to (more) realistic logistical problems. These extensions include dealing with multiple compartments, time windows, and some simple loading constraints. Still we do not choose to apply one of the described approaches for a number of reasons. First, none of the described problems incorporate such specific and detailed loading restrictions as compartment ullage rates (for dangerous liquid products) or minimum and maximum axle weights. Each of the mentioned algorithms would require extensive re-design to incorporate these restrictions, while the performance and behavior of the mentioned algorithms may suffer from the modifications. Second, we would not take advantage of the current VRP algorithm present in OTD. Third, as our load assignment problem itself is a hard problem to solve, we think a better approach is to consider the load assignment problem separately, developing a 'modular' algorithm that can be combined with the current VRP algorithm or be used stand-alone.

Having decided to solve the load assignment problem separately, we still need to determine a suitable technology. Section 3.2 discusses some applications of Constraint Programming to identify the specific problem characteristics for which Constraint Programming is a suitable problem solving technique. Based on the analysis in Section 3.2, we are able to conclude whether tackling the load assignment problem with Constraint Programming is a sensible idea.

3.2 Constraint Programming applications

Constraint Programming (CP) is a method for representing and solving a wide variety of problems. Problems are expressed in terms of variables, domains for those variables and constraints between the variables (*Backer et al.*, 2000). CP appears to be an appropriate technique to use in case of highly constrained problems, and therefore seems a suitable approach to solve the load assignment problem described in this report. We found evidence in the literature supporting this assumption.

Shaw (1998) indicates that CP appears to be a good technology to apply to VRPs because of the abundance of side constraints in real-world problems. *Barták* (1999) addresses the background of CP, explains its fundamental concepts, and mentions applications and limitations. According to Barták, assignment problems were the first type of industrial application solved with CP and scheduling problems are 'probably the most successful application area' for CP.

Limitations to CP are that the efficiency of constraint programs is unpredictable when applied to problems with a large solution space. Also, cost optimization may cause problems: sometimes it is very hard to improve an initial solution, and even a small improvement may take much more time than finding an initial solution. *Backer et al.* (2000) support these remarks, stating CP is an ideal candidate for (solving) VRPs due to the richness of the language used to express problems, but that it can be too time consuming to solve problems of practical size of the NP-hard VRP to optimality.

We make three observations:

1. CP seems very suitable for highly constrained problems;
2. Algorithm efficiency is unpredictable for problems with a large solution space;
3. Cost optimization may cause problems (contrary to just finding a feasible solution).

The many applications of CP reported in the literature seem to justify these observations. Although the applicability of CP is confirmed by various authors, it is scarcely used as a stand-alone method to solve an entire problem. As *Lambert* (2004) recognizes, current efforts in CP appear to be oriented towards combining CP with classical Operations Research (OR) approaches and heuristics (hybrid approaches) in an attempt to 'capitalize on their respective strengths'.

An example is the combination of CP with Mixed Integer Programming, in which a problem is split in a master problem and sub problems, the former being solved with CP

and the latter with MIP, or vice versa. *Corréa et al.* (2007) present a literature overview of applications of such hybrid CP/MIP models. Besides the combination with MIP, there is a large variety of hybrid models described in the literature, combining CP with various (meta-)heuristics such as Local Search, Tabu Search, and Variable Neighborhood Search. We mention two examples.

Shaw (1998) solves VRPs with a local search method called Large Neighborhood Search (LNS). LNS makes moves like in local search, but uses a tree-based search with constraint propagation to evaluate the cost and legality of the move. Shaw indicates that by using a *heavy weight* technique like CP to evaluate the move, fewer moves per second can be evaluated than is normally the case for local search. However, these moves can be more powerful, moving the search further in each step (hence *large neighborhood search*). Shaw uses the LNS to solve various benchmark capacitated VRPs and VRPs with time windows. While the results are equally good on these benchmark studies compared to (other) OR methods, the authors indicate that "CP holds more promise for tackling real-world problems due to its ability to effectively address side constraints".

Backer et al. (2000) use a similar approach to solve VRPs: various combinations of CP with local search (meta-)heuristics are tested, in which the respective local search methods are used to search solutions and CP is used to check the validity of the potential solutions.

Based on the observations that the algorithm efficiency is unpredictable for large (NP-hard) problems and that cost optimization may cause problems, CP is probably not an appropriate method to use for solving the complete MC-VRP with all load assignment constraints described in this report. Instances of the complete problem may be too large, and cost optimization is obviously the main goal of solving the MC-VRP. This observation supports the decision to retain the decomposed MC-VRP into two sub problems: determination of vehicle routes (trips) with their assigned resources and customers, and determination of a feasible assignment of customer orders to the resources in the trip. The latter is a multi-constrained assignment problem for which any feasible solution is sufficient. For this sub problem CP seems an appropriate modeling and solving approach.

Chapter 4

Constraint Programming Approach

To learn about *Constraint Programming* (CP) in general and the application of Inovia's CP tools specifically, we use the Inovia tools to model and solve the load assignment problem. CP is a programming method that represents the relations between variables in the form of constraints. These constraints do not specify the steps in which a solution to the problem should be found, but rather they specify properties of the solution.

As mentioned in Chapter 3, Constraint Programming can be very effective for solving highly constrained problems because the solver actually uses the constraints to find a feasible solution. In this chapter we propose a CP approach to solve the load assignment problem, and apply it to several self-constructed test problems.

The chapter is organized as follows: Section 4.1 describes the theory behind problem solving with Constraint Programming, emphasizing how constraints are used in the search for a solution. Section 4.2 discusses the algorithm development approach and describes the software tools used to model and solve the load assignment problem. Section 4.3 presents the CP load assignment model, which, in conjunction with the CP solver, constitutes the basic CP algorithm. Section 4.4 presents a set of test problems that we use to test the CP algorithm, and discusses our approach to testing. Section 4.5 proposes a variety of improvements to the basic CP algorithm. To determine the effect of these improvements, we test multiple CP approaches, each with a different algorithmic configuration. Section 4.6 presents the results of these tests and presents preliminary conclusions.

4.1 Constraint Programming theory

To describe how a Constraint Programming solver works, it is necessary to distinguish between two variants of CP: *constraint satisfaction* and *constraint solving*, differing in origins and solving technologies. In a Constraint Satisfaction Problem (CSP), variables

are defined over finite domains and the solving technology is based on methods that systematically search through the possible assignments of values to variables. For constraint solving, constraints are defined (mostly) over infinite domains and solving algorithms are based on mathematical techniques such as Taylor series or the Newton method. We will not deal with this last variant. The load assignment problem is, like most industrial applications, a typical example of a Constraint Satisfaction Problem (*Barták*, 1999).

A Constraint Satisfaction Problem is defined as:

- A set of variables $\boldsymbol{X = \{x_1, .., x_n\}}$,
- For each variable $\boldsymbol{x_i}$ a set $\boldsymbol{D_i}$ of possible values (the domain), and
- A set of constraints, restricting the values that the variables can simultaneously take.

A solution is an assignment of a value to each variable from its domain, without violating any of the constraints. We refer to assigning a value to a variable as *variable instantiation*. In a feasible solution, every variable is instantiated. The objective may be to find just one feasible solution, the best solution, or all solutions.

To find a solution for the problem, a CP solver repeats the following basic sequence of steps until a stopping criterion is reached, applying a certain logic in each of the steps:

1. Choose a variable that has not been given a value yet, using a *variable choice heuristic*;
2. *Instantiate* that variable with a value from its domain, chosen by a *value choice heuristic*;
3. Reduce the domains of all other uninstantiated variables by *constraint propagation*;
4. If the instantiation of the variable leads to an empty domain for one of the uninstantiated variables (i.e., infeasibility), backtrack and choose another value or variable.

The stopping criterion depends on the objective of the search: for feasibility problems, finding one or several feasible solutions may be sufficient, while optimization problems typically require finding the optimal solution according to a specified cost function.

The efficiency of this enumeration sequence strongly depends on the logic that is applied in each step. Using information from variable domains and constraints, algorithms may improve the efficiency of solving the problem either by reducing the number of possible choices (i.e., decreasing the problem space), or by improving the quality of the

individual choices, thereby reducing the required number of choices to find a feasible or optimal solution. We briefly describe the three types of heuristics that are used in the solving steps:

1. A *variable choice heuristic* determines an appropriate variable to instantiate next. A commonly applied rule is to choose the variable with the smallest domain at that moment, arguing that this domain has the highest probability of becoming empty in case another variable is instantiated first.

2. A *value choice heuristic* determines an appropriate value for the variable identified in step 1. For example, take the lowest value in the domain in case of a minimization problem.

3. *Constraint propagation* is a mechanism that removes infeasible values from variable domains by checking the consistency of domains with individual constraints.

The amount of inconsistent values that a constraint propagation mechanism, or *consistency technique*, removes from a variable domain depends upon the technique that is used. Different consistency techniques propagate different constraint types. Section 4.1.1 provides an overview of different constraint types, introducing the important concept of a **global constraint.** Section 4.1.2 discusses the concept of constraint propagation and introduces three consistency techniques.

4.1.1 Constraint types

We distinguish 4 different types of constraints, based upon the number of decision variables in the equation:

- Unary constraints; equations with only one variable, for example:

$$x < 100 \tag{4.1}$$

- Binary constraints: equations with two variables, for example:

$$x_1 + x_2 < 200 \tag{4.2}$$

- Ternary constraints: equations with three variables, for example:

$$x_1 + x_2 + x_3 < 300 \tag{4.3}$$

- N-ary constraints: equations with N variables, with $N > 3$, for example:

$$x_1 + x_2 + x_3 + x_4 < 400 \tag{4.4}$$

The different types of constraints require different consistency techniques to prune domain variables.

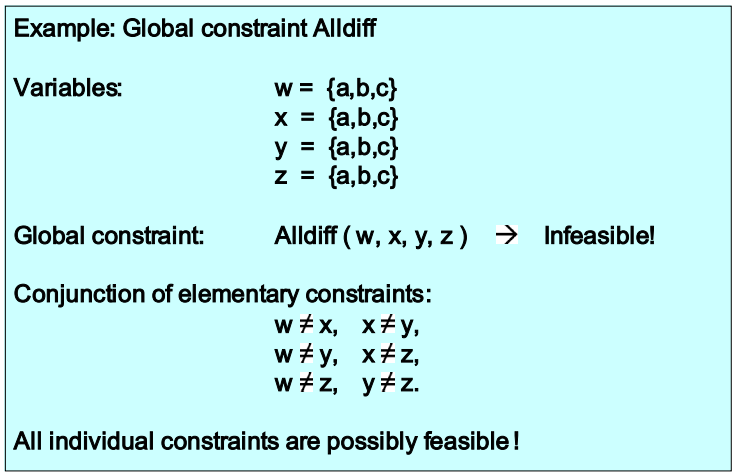

Figure 4.1: Example of the effect of a global constraint

Global constraints

There is one special type of constraints that does not rely on a specific consistency technique for the ability to reduce domains: global constraints. Despite the intuitive nature of the concept of a global constraint, no formal definition exists. We come to the following description by combining characterizations from several papers (*Bessière and Hentenryck*, 2003; *Barták*, 1999).

A global constraint is an expressive condition that applies a certain relation between variables through the use of a domain filtering algorithm. A global constraint represents a conjunction of several elementary constraints, but it has more 'pruning power' than the conjunction of elementary constraints that it comprises by applying a comprehensive ('global') view on the variables.

We clarify these characteristics with an example: Figure 4.1 depicts the global *alldifferent* (alldiff) constraint, imposing on the variables the condition that every variable must have a different value. For the four variables, w, x, y, and z, the alldiff constraint can be decomposed into 6 elementary constraints as demonstrated. For the set of elementary constraints, a consistency technique would iteratively consider each of the individual binary constraints, none of which leads to domain reduction, therefore not noticing infeasibility. The alldiff constraint has a filtering algorithm with a *global* view to the variables. Propagating the alldiff constraint in the example therefore will detect infeasibility.

Consistency techniques consider at most two elementary constraints at the same time, only pruning the values that are infeasible in a single constraint, or the combination of

two binary constraints. The filtering algorithms of global constraints have a high level vision and are capable of detecting inconsistencies across different elementary constraints on an unlimited number of variables. Global constraints are therefore powerful tools for solving CP problems.

4.1.2 Consistency techniques

Consistency techniques aim at checking the consistency of the current (partial) solution and removing inconsistent values from variable domains. Inconsistent values are values that cannot be part of a feasible solution, given the current state of all variable domains. Most often, consistency techniques are used in conjunction with a search method, checking domain consistency before and during the search. There is a variety of consistency techniques, varying in the type of constraints used to check consistency and the respective *pruning power*: the amount of inconsistent values discovered and removed by the respective technique. The most 'weak' techniques only consider instantiated variables (backtracking), where the more powerful techniques also consider the domains of uninstantiated variables. At the basis of all consistency techniques is a mechanism referred to as *constraint propagation*.

Constraint propagation

Bessiere (2006) gives with the following definition of constraint propagation: *Constraint propagation embeds any reasoning which consists in explicitly forbidding values or combinations of values for some variables of a problem because a given subset of its constraints cannot be satisfied otherwise.* Constraint propagation decreases the search space by pruning values from variable domains that cannot be part of a feasible solution according to the constraint that is propagated. Constraint propagation can be applied both before and during the search for a solution. During the search, the mechanism of constraint propagation prunes values that have become infeasible after each variable instantiation. Properly propagating constraints during the search prevents many unnecessary backtracks, enabling more efficient problem solving.

We describe three different consistency techniques, differing in the type of constraints they propagate. The degree of domain consistency that a consistency technique achieves, its pruning power, depends on the constraint type that is propagated. We present three consistency techniques in increasing order of pruning power. The names of the techniques refer to the graph representation of the relations between variables.

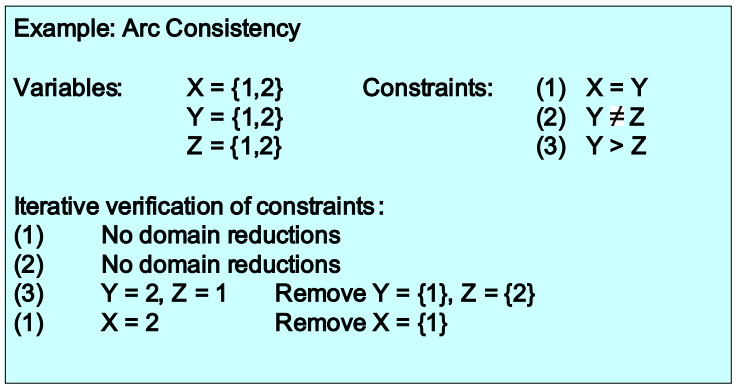

Figure 4.2: Example of domain reduction using Arc Consistency

Node-Consistency

Node-Consistency is the simplest technique: it removes values of variable domains that are inconsistent with unary constraints on the respective variable.

Arc-Consistency

The most widely used consistency technique is Arc-Consistency (AC). Besides checking Node-Consistency, this technique removes values by iteratively propagating individual binary constraints until none of the constraints causes domain reductions.

Consider for example three variables, X, Y, and Z, all with domain {1,2}. Furthermore, there are three constraints: $X = Y$, $X \neq Z$, and $Y > Z$. The AC algorithm iteratively verifies each individual constraint, removing inconsistent values until none of the variable domains contain values that are inconsistent with any of the individual constraints. Figure 4.2 demonstrates the required steps to identify the solution of the example. Verifying the constraint $X = Y$ for the first time does not result in domain reductions, but after the domains change due to the evaluation of the constraint $Y > Z$, the first constraint is re-evaluated. This time it leads to the removal of value 1 in the domain of X, thereby finding the feasible solution.

Path-Consistency

Otherwise known as 3-Consistency, referring to the amount of variables simultaneously considered by this technique, Path-Consistency (PC) checks for any value-combination of two binary constrained variables X and Y, which values of the domain of Z are inconsistent, given the constraints between X and Z, and Y and Z. We demonstrate the higher degree of consistency of PC over AC by modifying the previous example: consider the same variables, but now with the constraints $X \neq Y$, $X \neq Z$, and $Y \neq Z$. Clearly, this is

Constraint type(s) (# of related variables)	Consistency technique	Constraint example
unary (1)	Node-Consistency	$x < 10$
binary (2)	Arc-Consistency	$x_1 + x_2 < 10$
ternary / conjunctions of binary (3)	Path-Consistency	$x_1 + x_2 + x_3 < 10$
> 3	no consistency technique	$x_1 + x_2 + x_3 + x_4 < 10$
global (i)	not required	$Sum(x_1, .., x_i) < 10$

Table 4.1: Overview of constraint types and consistency techniques

an infeasible example, but considering the individual binary constraints (AC) does not detect this. PC considers the possible values for (X,Y): (1,2) and (2,1), and for both possible combinations it checks the possible values of Z by checking X $\neq$ Z and Y $\neq$ Z. There are no consistent values of Z for this problem, therefore PC does detect problem infeasibility.

Global Constraints

Constraint propagation for global constraints does not depend on consistency techniques, as global constraints have their own filtering algorithm.

Table 4.1 summarizes the combinations of constraint types and consistency techniques, and provides an example constraint for each type:

4.2 Development approach

Solving a problem with CP is conceptually quite similar to Integer Linear Programming: both require a model that represents the problem and a solver that interprets the model to solve actual problem instances. As we mentioned in Section 4.1, there are three different ways to influence the problem solving mechanics of a Constraint Programming approach, by means of heuristics that assist the solver in the different steps of the solving process. Therefore, we develop the CP load assignment approach in two phases. In the first phase we develop a basic CP algorithm without emphasis on constraint propagation mechanisms or variable and value choice heuristics. In the second phase we focus our attention to improving the computational speed of the basic CP algorithm by improving the constraint propagation mechanisms and adding variable and value choice heuristics. In both phases we use the software tools developed by Inovia for modeling and testing. Section 4.2.1 provides some details about these software tools. Sections 4.2.2 and 4.2.3 further describe both development phases.

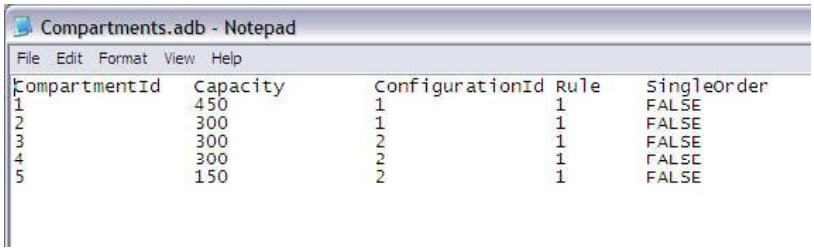

Figure 4.3: Compartment parameters in a flatfile

4.2.1 Constraint Programming tools

Advanced Query Language

We use the modeling language Advanced Query Language (AQL), developed by Inovia based on the Standard Query Language, to model the load assignment problem. All parameters and variables used in the model are represented in tabular format. For example, the following represents the definition of the compartment parameters in AQL :

```
AQL_TABLE Compartments WITH_TITLE
(
    CompartmentId    NUMBER,
    Capacity         NUMBER,
    ConfigurationId  NUMBER,
    Rule             NUMBER,
    SingleOrder      BOOLEAN
) ;
```

Figure 4.3 depicts an example of the corresponding input file with the actual parameter values.

Advanced Solver

To solve instances of the load assignment problem we use the 'Advanced Solver', developed by Inovia. This solver works in conjunction with AQL. A call to the Advanced Solver activates a fixed sequence of steps: it reads the input parameter tables, it creates the variables and domains, it pre-processes the variable domains, and then it starts the search for a solution (see Section 4.1 for the steps of the search method).

Constraint propagation and Arc-Consistency in the Advanced Solver

In the pre-processing step before the start of the search, and after each variable instantiation, the Advanced Solver applies the Arc-Consistency technique to reduce domains.

Note that applying Arc-Consistency to elementary constraints only leads to domain reductions when these constraints are *unary* or *binary*: not to constraints on more than two uninstantiated variables. In other words, propagating elementary constraints in the Advanced Solver only leads to domain reduction in case they are *unary* or *binary*. Elementary constraint on more than two uninstantiated variables will not induce domain reductions after variable instantiation.

Visual Inspector

The Visual Inspector is a tool that calls the Advanced Solver to solve test problems and provides the opportunity to step through each choice for a variable and a corresponding instantiation value. After each variable instantiation, the Visual Inspector displays updated variable domains and domain reductions. Figure 4.4 depicts a screen shot of the Visual Inspector, with in the upper left field the instantiated variables, in the upper right field the uninstantiated variables and their domains, and in the bottom left field additional information about variable instantiations and resulting domain reductions.

4.2.2 Model development

The first phase of algorithm development is modeling the load assignment problem by translating the problem description into appropriate parameters and CP constraints in an AQL model. We validate the model by solving small problem instances and analyzing the output. Section 4.3 presents the actual load assignment model, which can be solved by the Advanced Solver.

4.2.3 Algorithm improvement

The possibility to influence the search method introduces additional degrees of freedom in CP algorithm development. We apply a systematic approach to determine which combination of heuristics leads to the optimal CP algorithm. For our purpose of the load assignment algorithm, the optimal algorithm is the one that returns an arbitrary feasible solution as fast as possible. We measure the computational speed of different CP approaches by solving a set of self-constructed test problems that we introduce in Section 4.4.

Performance debugging

For improving the computational speed of the basic CP algorithm, we use the Visual Inspector to run test problems. Stepping through each solver decision and analyzing the given information to improve problem solving is called *performance debugging*. It provides insight in the variable selection process, the value selection process, and the

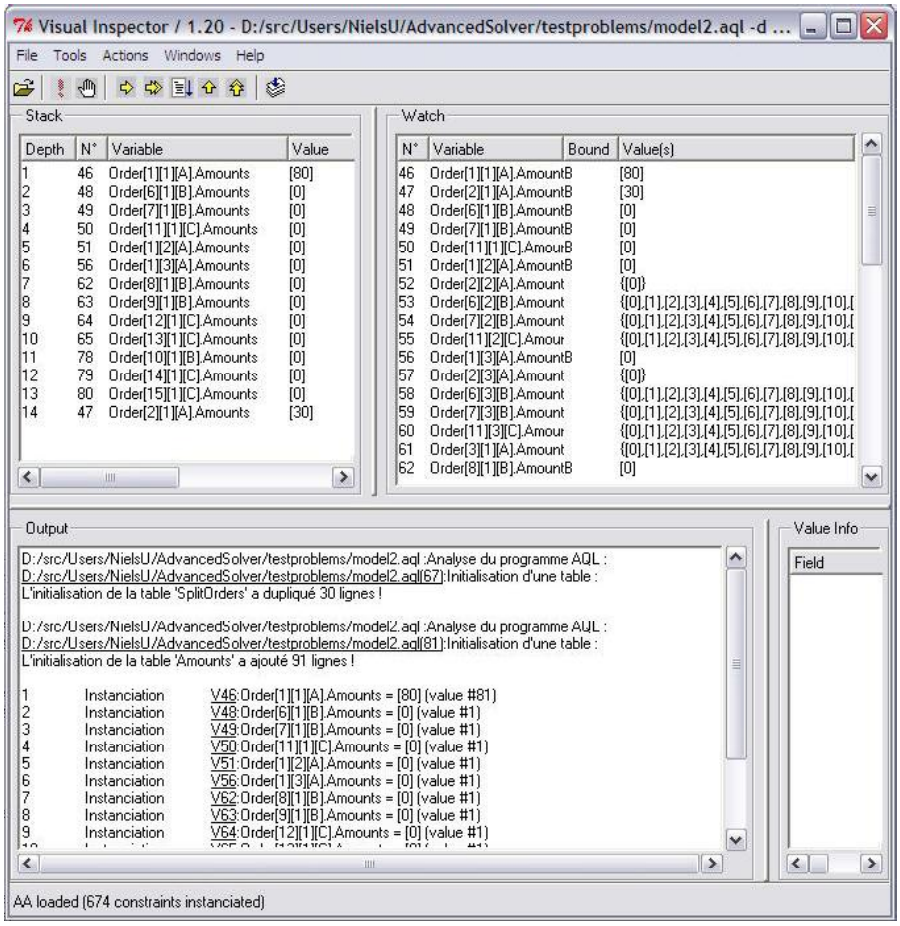

Figure 4.4: Screen shot Visual Inspector

impact of constraints: the way constraints reduce domains after variable instantiations. Performance debugging reveals the bottlenecks in finding a feasible solution.

Iterations of performance improvement

Using the Visual Inspector, it is possible to detect inefficient solver choices for the instantiation of variables and reduction of domains. This allows for several iterations of algorithm improvement in the following sequence of steps:

- solve a test problem with the Visual Inspector,
- detect inefficient solver choices,
- add or modify a constraint or heuristic to improve these specific choices.

Following this sequence of steps, we develop and test several improvements to the CP algorithm. Section 4.4 describes the test problems we use for testing the different CP approaches. Section 4.5 presents the CP algorithm improvements, and Section 4.6 present the results of these tests.

4.3 Load assignment model

Decision variables

A solution to the load assignment problem states for each order the amount of order load that is put into each compartment. Therefore, there is a decision variable for each combination of order o and compartment c: $\boldsymbol{Amount_{oc}}$ with domain {0,..., amount of order $\boldsymbol{o}$}.

Constraints

The model allows for order pickups and delivers to alternate. To determine which orders are on board during each section, the input must specify a pickup and delivery time. We distinguish two groups of constraints: the constraints that apply to the entire trip, which do not depend on the combinations of orders that are actually together in the resource, and the constraints that apply to the individual trip sections, which do depend on the actual orders on board during each section.

The constraints that apply to the entire trip are:

- For each order, the sum of all order amounts assigned to each compartment equals the total order amount.

- Product and compartment must be compatible if the product is assigned to the compartment.
- Only equal products can be assigned to the same compartment (even when loaded at different parts of the trip).

The constraints that apply for each section are:

- For each compartment, the sum of the volumes of the orders assigned to it should be smaller or equal to its volume capacity.
- For each configuration, the sum of the weights of the orders assigned to its compartments should be smaller or equal to its weight capacity.
- For each compartment, the sum of the order amounts should be lower than the ullage rate lower bound or higher than the ullage rate upper bound specified for the compartment.
- For each axle, the sum of the order amounts per compartment, multiplied by the contribution of the compartment to the axle, should be smaller than its maximum allowed weight, and larger than its minimum allowed weight.
- If the compartment is a single order compartment, no more than one order may be assigned to it in each section.

Optimization criterion

As we are only interested in finding the first feasible solution, we do not specify a cost function. Appendix B presents the complete AQL load assignment model.

4.4 Test problems

Testing the load assignment approaches serves three goals:

1. Tests provide an objective measurement of the computational speed of our CP approach. This measurement can be compared with other solving techniques such as ILP.
2. Measuring the required computation time is necessary to assess the suitability of the CP approach as load assignment algorithm within the VRP framework.
3. Testing different CP approaches provides insight in the effect of the proposed improvements (see Section 4.5) and is necessary to determine the optimal CP approach.

To best serve these goals, the test problems should be *realistic* and *challenging*. The problems should be realistic in the sense that they accurately reflect the problems for which the algorithm will finally be used. Testing *challenging* problems gives insight in the average and worst-case performance of the algorithm. This is important to assess the suitability of the current CP approach as load assignment algorithm for VRP: if 9 out of 10 problems are solved in 5 milliseconds, but the 10th problem takes 5 minutes, the CP approach is too slow to use as load assignment algorithm in a framework for optimizing load and routes.

4.4.1 Problem parameters

To determine realistic parameter values, ORTEC business consultants with experience in the field of liquids transportation gave advice on parameters such as the number of compartments and the number of different products in an average trip. Tuning the parameters (while keeping them realistic) such that there are very few feasible solutions, while keeping the individual decision variable domains large enough, ensures challenging test problems. To accomplish this, we make sure that the constraints on groups of variables are binding, opposite to the unary or binary constraints, which would actually help the solver find a feasible solution faster.

For example, a ullage rate constraint does not cut off large parts of individual decision variable domains, but it bounds the union of the domains, which makes it much harder for the solver to detect inconsistent domain values in the early stages of the search tree.

To improve solving efficiency, we scale down all order amounts and capacity parameters with a factor 100. This restricts order division to minimum steps of 100 liters: a compartment with capacity 150 can actually store 150 order parts of 100 liters, representing an actual capacity of 15000 liters. Companies that handle orders of this magnitude commonly determine order sizes in steps of 100 liters. Scaling all order amounts and capacities therefore not only enables much more efficient problem solving, it also prevents unrealistic order quantities and load assignments. Table 4.2 indicates for each problem parameter the range of values used in the test problems. To measure the influence of order sizes, we test three variants of each test problem, differing in order sizes. The different variants of the models are obtained by multiplying all relevant parameters (order sizes and capacities) with a factor 1, 3, and 10. The problem variants can be distinguished by their code: p_(problem number)_(factor).

Problem:	p_1_1	p_2_1	p_3_1	p_4_1
Number of orders	32	18	15	15
Order size	5 - 20	10 - 60	10 - 100	20 - 90
Number of different products	4	3	3	3
Number of compartments	7	6	5	3
Compartment capacities	15 - 33	40 - 50	50 - 150	110
Ullage rates (LB% - UB%)	35 - 65 95 - 99	40-60	45 - 75	50 - 60

Table 4.2: Test problem parameters

4.4.2 Infeasibility detection

Depending on the application of the load assignment algorithm, early detection of problem infeasibility may be just as important as performance in case of feasibility. This certainly holds for our purpose, therefore we also test infeasible problems. We expect a strong correlation between the *degree* of infeasibility and the time required to detect infeasibility: we expect that it is much easier to verify infeasibility if a problem is by far not feasible, for example assigning orders of three different products to a resource with two compartments, compared to problems that are marginally infeasible, for example because every possible load assignment causes a small violation of the axle weight restriction.

To demonstrate this relation we take a feasible problem, systematically make it more and more infeasible by reducing the **summed compartment capacity** in steps of 10% (all of the problems are infeasible after the first reduction of compartment capacity of 10%), and measure the time the solver requires to establish infeasibility for each step.

In each step, we subtract a 10% of the original total compartment capacity from one of the compartments that are left, removing an entire compartment if it becomes too small. Note that this is just one out of many ways to make a problem infeasible by altering the parameters. It is our aim to provide just a rough idea about the order of magnitude of detection time for various degrees of infeasibility.

4.5 Solution improvements

The load assignment model in Section 4.3 is the basis of the CP approach to solving the load assignment problem. This 'basic' algorithm uses no heuristics for variable and value selection and the constraints used in the model are mainly elementary constraints, lacking the global view that provides more pruning power.

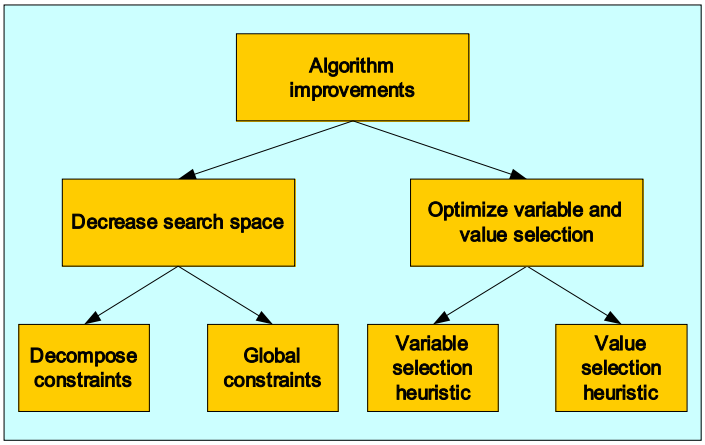

Figure 4.5: Overview performance improvements

Performance debugging reveals that solving efficiency can be improved. Frequently, a variable instantiation leads to obvious possible domain reductions that the constraint propagation mechanism does not apply. Besides checking Arc Consistency and applying domain reductions for the elementary constraints, variable selection and value selection are arbitrary, practically resembling complete enumeration. This chapter describes improvements of the basic algorithm.

The improvements are based on two concepts:

- Reducing the search space by improving constraint pruning power: with fewer variables and smaller domains, the number of possible combinations decreases, making the problem less difficult to solve.
- Traversing the solution space more efficiently by choosing *appropriate* variables to instantiate with *suitable* values.

For both concepts we apply and test multiple improvements to the CP approach. See Figure 4.5 for an overview of improvements.

4.5.1 Constraint improvements

We apply performance debugging to determine possibilities to reduce the problem space. Debugging the basic algorithm reveals solving inefficiencies for the following constraints, all of which sum over a subset of decision variables:

1. Assign total order amount
2. Compartment capacity

3. Configuration capacity

4. Axle weights

5. Compartment ullage rates

The constraint propagation mechanism of these constraints performs poorly: after each variable instantiation, the domains of the remaining uninstantiated variables in the constraints are unaffected, until there are just **two** uninstantiated variables left. We explain this with an example.

Consider the problem in Figure 4.6. Assume order 1 is assigned completely to compartment 1. According to the first constraint, the compartment is full and the amount of order 2, 3, and 4 assigned to compartment 1 can only be 0. Therefore, we expect domain reductions for the variables $Amount_{21}$, $Amount_{31}$, and $Amount_{41}$. These domain reductions do not occur when using the original sum constraint. This leads to unexpected behavior; in this example, having the solver start variable instantiations from high to low domain values, the solver proposes to instantiate variable $Amount_{21}$ with value 100, checks the constraint, backtracks, tries value 99, backtracks, etc., until it reaches value 0.

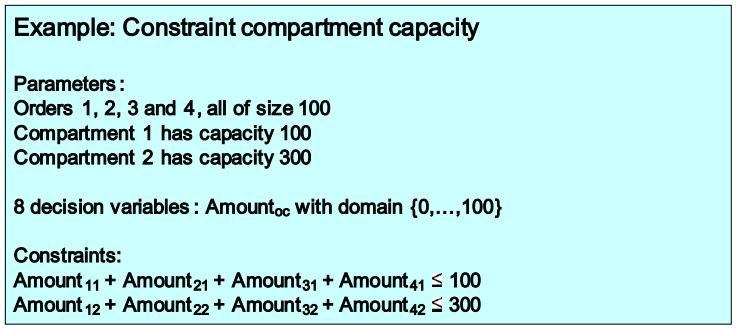

Figure 4.6: Example of the lack of domain reduction - part 1

This phenomenon is caused by the fact that the sum operator in the Advanced Solver behaves like an elementary constraint. Contrary to other CP solvers, in Inovia's Advanced Solver sum is not programmed as a global constraint, but as an elementary operator. Therefore, all constraints modeled in AQL using the sum-operator behave as elementary constraints. Arc-Consistency only leads to domain reduction for these constraints if there are only two uninstantiated variables in the sum.

Consider the following example of the AQL-implementation of the compartment capacity constraints, in which sum acts as a elementary operator:

```
FOR_EACH ?S FROM Sections
{
   FOR_EACH ?C FROM Compartments
   {
      sum( FOR_EACH ?O FROM Orders
           VERIFYING Order[?O].CompartmentId == Compartments[?C].CompartmentId
           AND       Order[?O].PickupTime    <= Sections[?S].StartTime
           AND       Order[?O].DeliverTime   >= Sections[?S].FinishTime
           { result[?O].AffectedAmount }
          )  <= Compartments[?C].Capacity ;
    } ;
} ;
```

Sections are modeled using time-windows: each section has a start time and finish time. An order is on board of the resource in a section if its pickup occurs before or at the start time of the section and its deliver occurs at or after the finish time of the section.

Constraint decomposition

With five constraints using the sum-operator in the basic model, the lack of domain reduction is a serious weakness of this CP approach. One way to overcome this weakness is to decompose multiple-variable constraints into a set of binary constraints, as binary constraints are always successfully propagated. We introduce auxiliary variables $CumulAssigned_{sc}$ with domain $\{0, ..., Capacity_c\}$ that represent the cumulative amount of load assigned to compartment c in step s. Furthermore, we fix the order of variable instantiations such that in step s, order $o = s$ is instantiated. The recursive decomposition of the compartment capacity constraint replaces

$$\sum_{o \in O} Amount_{oc} \leq Capacity_c \qquad \forall c \tag{4.5}$$

by

$$\begin{aligned}
Amount_{11} &= CumulAssigned_{11} \\
Amount_{21} + CumulAssigned_{11} &= CumulAssigned_{21} \\
Amount_{31} + CumulAssigned_{21} &= CumulAssigned_{31} \\
\vdots \qquad\qquad \vdots \qquad &\vdots \qquad\qquad \vdots \\
Amount_{s1} + CumulAssigned_{s-1,1} &= CumulAssigned_{s1}
\end{aligned}$$

with

$$\mathit{CumulAssigned}_{s1} \leq \mathit{Capacity}_1 \tag{4.6}$$

After each instantiation of a decision variable Amount_{oc} the variable $\mathit{CumulAssigned}_{sc}$ is updated. As the variable $\mathit{CumulAssigned}_{sc}$ is used in a binary constraint with each decision variable Amount_{oc}, the domains of all decision variables Amount_{oc} for the corresponding compartment are reduced. Therefore, this constraint decomposition requires n constraints and n auxiliary variables. We refer to this recursive structure of constraints as *recursive constraint decomposition.*

A disadvantage of recursive decomposition is the required fixed order of variable instantiations: Amount_{oc} may only be instantiated after $\mathit{Amount}_{o-1,c}$ otherwise the constraints are incorrect. Thus, the improved constraint propagation comes at the cost of a less generic and less flexible model. Also, the recursion only improves inequality constraints. Another disadvantage of constraint decomposition is that the performance gains due to better constraint propagation may be offset by a performance loss due to the introduction of additional variables and constraints. In other words, it may take less steps to find a feasible solution, but each step takes more time.

New global constraints

As mentioned in the discussion on elementary and global constraints, generally global constraints have more pruning power than an equivalent decomposition of elementary constraints. As a second approach to improve the domain reduction mechanism, we introduce global constraints to replace the original elementary constraints. For each global constraint we describe the filtering algorithm and the original constraint it replaces.

SumSmallerOrEqual and SumLargerOrEqual

The SumSmallerOrEqual constraint replaces:

$$\sum x \leq \mathit{Constant}$$

The SumLargerOrEqual constraint replaces:

$$\sum x \geq \mathit{Constant}$$

Both constraints together replace:

$$\sum x = \mathit{Constant}$$

These constraint types apply to the model constraints *assign total order amount*, *compartment capacity* and *configuration capacity*. In other words, the instances of the original constraints are replaced by instances of the global constraints. Figure 4.7 explains how the filtering algorithm reduces domains for the example of the compartment capacity constraint. The filtering algorithm calculates the residual amount of the lower bound: the actual lower bound for the remaining uninstantiated variables. For each uninstantiated variable, the filtering algorithm removes the domain values that exceed this lower bound. Algorithm 1 depicts the filtering algorithm for the SumSmallerOrEqual con-

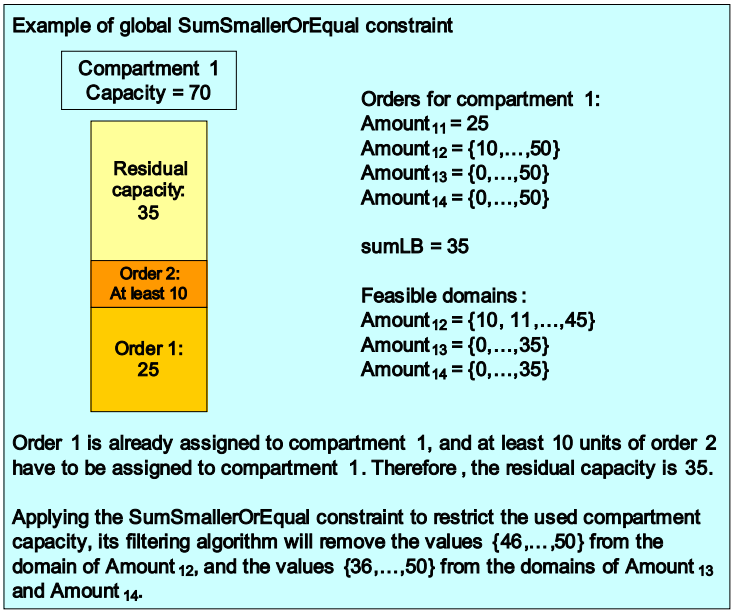

Figure 4.7: Example of filtering algorithm of SumSmallerOrEqual constraint

straint. The filtering algorithm for SumLargerOrEqual works in a similar fashion, only for the opposite bounds.

SumOfProductsSmallerOrEqual

The SumOfProductsSmallerOrEqual constraint replaces:

$$\sum x_i \cdot factor_i \leq Constant$$

that applies to the *axle weight constraint*. The global constraint closely resembles the SumSmallerOrEqual. In fact, the global SumOfProductsSmallerOrEqual constraint is a generalization, and is identical if all factors are equal to 1. The filtering algorithm first applies the multiplication of variable value and factor at the appropriate places, and performs similar steps compared to the filtering algorithm of SumSmallerOrEqual.

Algorithm 1 Filtering algorithm of SumSmallerOrEqual

The set V contains all decision variables v_i in the constraint SumSmallerOrEqual($lowerbound; v_1, ..., v_n$). The subset V_u contains all *uninstantiated* variables v. For each $v \in V$, D_v contains all values d that are in the domain of v.

The filtering algorithm requires two steps: (1) the determination of the sum of all domain lower bounds, and (2) the removal of inconsistent domain values. Note that $\min(D_v)$ returns the value of v for instantiated variables, and the lowest value in D_v for uninstantiated variables.

```
(1)
for all v ∈ V do
   SumLB = SumLB + min(D_v);
end for

(2)
for all v ∈ V_u do
   domainUB = D_v.UpperBound;
   domainLB = D_v.LowerBound;
   if (domainUB + SumLB − domainLB > lowerbound) then
      for all (d ∈ D_v > lowerbound − SumLB + domainLB) do
         remove D_v.d;
      end for
   end if
end for
```

UllageRates

The global UllageRates constraint is much more specific to the load assignment problem, compared to the previous global constraints. It replaces the following structure of two constraints:

$$\sum x \leq Lowerbound \quad OR \quad \sum x \geq Upperbound$$

that applies to the *ullage rate constraint*. The filtering algorithm of this global constraint is a bit more complex, as it should only remove domain values for which both restrictions are violated. Algorithm 2 depicts the filtering algorithm, which is exemplified by Figure 4.8. The determination of the sums of domain lower and upper bounds is similar to the first part of Algorithm 1 and is not depicted in Algorithm 2. The filtering algorithm of the global UllageRates constraint basically removes for each variable those domain values that are not feasible with any combination of values from the other variables.

Consider the example in figure 4.8: if the current sum of all instantiated variables is 8, and the sum of all values may not be between 10 and 50. For the variable $Amount_{12}$ with domain {0,...,62} this means it may take the values smaller or equal to 2, or larger or equal to 22, as the upper bound of the sum of the values of the other two uninstantiated variables is 20. Therefore, to 'reach' the ullage rate upper bound, $Amount_{12}$ must

Algorithm 2 Filtering algorithm for UllageRates

Similar to Algorithm 1, SumLB represents the sum of the domain lower bounds. Likewise, SumUB represents the sum of the domain upper bounds. The set V contains all decision variables v_i in the constraint UllageRates(*lowerbound*;*upperbound*:*CompCapacity* $v_1, ..., v_n$). The subset V_u contains all *uninstantiated* variables v. For each $v \in V$, D_v contains all values d that are in the domain of v.

for all $v \in V_u$ **do**
  $domainUB = D_v.UpperBound$;
  $domainLB = D_v.LowerBound$;
  if $(domainUB + SumLB - domainLB > lowerbound)$
  && $(domainLB + SumUB \quad domainUB < upperbound)$ **then**
    for all $(d \in D_v$ **do**
      if $(d + SumLB - domainLB > lowerbound)$
      && $(d + SumUB - domainUB < upperbound)$ **then**
        remove $D_v.d$;
      end if
    end for
  end if
end for

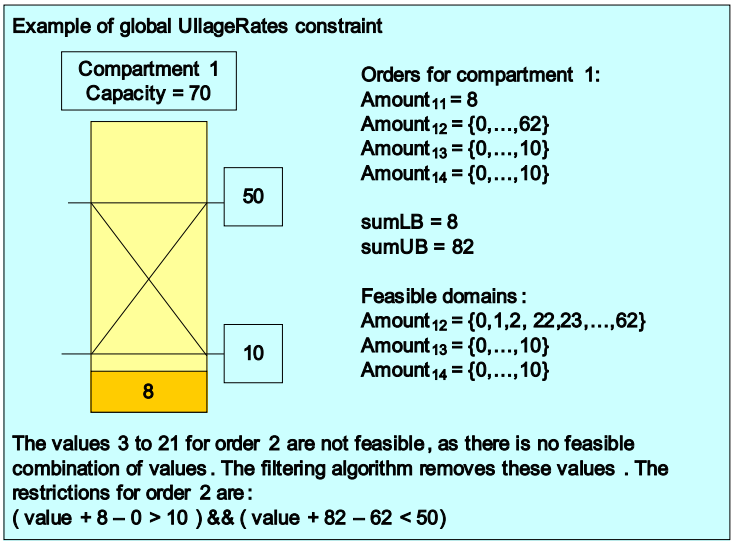

Figure 4.8: Example of global UllageRates constraint

contribute at least 22 to the sum of the variable values. Every value between 2 and 22 can be removed from the domain of the variable $\boldsymbol{Amount_{12}}$.

4.5.2 Value selection improvement

The basic CP approach has no heuristic that determines a suitable instantiation value. The solver uses its default strategy: always take the lowest value in the variable domain. We demonstrate that the **value choice heuristic** can be improved.

The options with respect to value selection are limited due to modeling possibilities: the model either works bottom up or top down through the domain of each variable, and iteratively tries every consecutive value until it finds a value for which no constraints are violated. In the original model, all decision variables are instantiated with the lowest value in the domain. What happens is that all variables $\boldsymbol{Amount_{oc}}$ are instantiated with value 0, until the constraint 'assign total order amount' complains for every order $\boldsymbol{o}$ that the total order amount is not assigned, requiring a lot of backtracks until the constraint is no longer violated. Intuitively, it makes more sense to assign as much of each order to a compartment as soon as possible in the search, applying the *top down* strategy. Adding a *preferred* constraint (soft constraint) to maximize the value of each individual decision variable ensures that the solver starts variable instantiation with the largest value in its domain.

4.5.3 Variable selection improvement

Similar to the value selection, we demonstrate that we can improve the default variable selection strategy - always instantiating the variable with the smallest domain next - with a different **variable choice heuristic.**

We can improve the default variable choice strategy, taking the variable with the smallest domain, by using specific knowledge of the problem. Imposing a fixed variable instantiation order overrules the default variable selection strategy. The solver then follows the order of the variable input table, instantiating the next variable $\boldsymbol{Amount_{oc}}$ only when all previous variables are instantiated. A different variable choice heuristic is then achieved by sorting the variable input table by one or multiple criteria. Therefore we refer to a variable choice heuristic as a **sorting strategy.** For example, sorting by increasing pickup time of the orders makes that the solver assigns the orders with the earliest pickup time first. Note that it is only possible to impose a static variable order: a dynamic strategy in which the choice of the next variable depends upon previous instantiations is not possible with the modeling language AQL.

A good variable choice heuristic improves the computational speed of the algorithm by helping to identify infeasible variable instantiations in an early stage of the search. Early pruning of infeasible combinations of variable values reduces the required number of steps to find a feasible solution, or to detect problem infeasibility. Applying this reasoning to the load assignment problem, we will improve solution performance with a variable choice heuristic that filters out infeasible combinations of assigned order amounts to compartments in an **earlier** stage of problem solving.

To detect infeasible combinations of assigned order amounts to compartments in an early stage of problem solving, we should cluster those combinations of decision variables that together may cause infeasibility. For the constraints of the load assignment problem (compartment capacity, axle weights, ullage rates, and configuration capacity) this means we cluster for orders that are in the resource at the same time by ordering the decision variables on pickup time. Within this clustering, we propose three sub clusters that further specify combinations of variables that (1) affect each other, and (2) that imply an *intuitively* appropriate instantiation ordering. The intuition is that it is best to either (1) assign the largest orders first, (2) fill the largest compartments first, and for compartments of similar size (3) consider each compartment separately.

We propose the following sorting strategies:

- Increasing Pickup time (sorting strategy A)
- Increasing Pickup time, decreasing Order Amount (sorting strategy B)
- Increasing Pickup time, decreasing Compartment Capacity (sorting strategy C)
- Increasing Pickup time, decreasing Compartment Capacity, CompartmentId (sorting strategy D)

4.5.4 Effects of the improvements

The effect of the proposed improvements on the computation time is hard to predict. For example, replacing an elementary constraint by a global constraint with better pruning power reduces the number of infeasible variable instantiations, leading to less backtracks on the one hand. On the other hand, evaluating the global constraint itself takes more time. The net effect is hard to predict, and empirical testing is necessary to assess the quality of the individual improvements, and to determine the optimal CP approach. We systematically compare improvements in the following three categories:

- The type of constraints (original elementary, recursive decomposition, global),

- The value choice heuristics (bottom-up, top down), and
- Variable choice heuristics (A, B, C, D).

For every test we compare the different options within one category, while keeping the 'configuration' of the other two categories constant. Section 4.6 provides an analysis of the test results.

4.6 Test results

In Section 4.5.4 we propose tests for improvements in three categories: (1) constraints, (2) value selection heuristics, and (3) variable selection heuristics. However, the effects of a change in one category may depend on the configuration of the other two categories. For each test we vary the choices within one category, while keeping the configuration of the other two categories fixed. For the configuration of the fixed categories, we choose a default option for each category, for objective comparison. The default options are: (1) global constraints, (2) top-down, and (3) sorting strategy A. In the following sections we present the outcome of the tests per category. For each comparison we briefly mention the solution configuration(s), the test results, and some remarks where relevant.

4.6.1 Effects of improved constraints

Recursive constraint decomposition

Although we assume that in some cases applying recursive constraint decomposition leads to a faster algorithm, we discover in an early stage of the improvement development that replacing the original constraints with recursive constraints actually causes similar or even worse computation times. The reduction of the amount of backtracks due to these restrictions is significant, but so is the increase of the amount of variables that are instantiated, and the additional time spent on instantiating the auxiliary variables in our problems outweighs the time saved by the reduction of backtracks. Furthermore, the required fixed order of variable instantiations for recursive constraint decomposition severely restricts the possibility to improve performance by varying the order of variable instantiations, which we propose as the mechanism for a variable choice heuristic.

Global constraints

To test the effects of the new global constraints we test the CP approach with the original elementary constraints and the approach with the new global constraints. Both approaches apply sorting strategy A as variable choice heuristic, and the *top down* value choice heuristic proposed in Section 4.5. Table 4.3 presents the total running time in milliseconds until the **first** solution is found for both approaches. Entries indicated with

an asterisk (*) represent situations in which no feasible solution was found in 10 minutes.

As expected, for most problems the approach with global constraints finds a feasible solution much faster and the difference increases for problems with larger order amounts. We achieve reductions of running times up to 99%! There is one exception: the approach with new global constraints fails to find a feasible solution for problem p_1_3 within 10 minutes, even though p_1_3 is similar to problem p_1_1, but with larger order amounts and capacities. Another remarkable observation is that the solver requires less than 3 seconds to find a feasible solution to problem p_3_10, but fails to find a solution within 10 minutes to the smaller problem p_3_3, just as the original approach.

	Elementary constraints	Global constraints	Reduction (%)
p_1_1	5670	5090	10%
p_1_3	290000	*	-
p_1_10	*	*	-
p_2_1	233	78	67%
p_2_3	2740	188	93%
p_2_10	102000	1000	99%
p_3_1	296	78	74%
p_3_3	*	*	-
p_3_10	116000	2800	98%
p_4_1	47	15	68%
p_4_3	235	63	73%
p_4_10	2220	172	92%

Table 4.3: Computation time (ms) for the CP approach with elementary constraints and the CP approach with global constraints. An * indicates a computation time larger than 10 minutes.

4.6.2 Effects of improved value selection

There are two options for the value selection process: either start at the highest value in the domain and work downwards, or start at the lowest value and work upwards. We test both heuristics with the approach that applies global constraints and sorting strategy A. Table 4.4 provides the running times in milliseconds until the first feasible solution is found for both approaches. It is obvious that the best performing value choice heuristic is to attempt value instantiation at the highest domain value and work in decreasing order.

	Decreasing value selection	Increasing value selection
p_1_1	5090	18150
p_1_3	*	*
p_1_10	*	*
p_2_1	78	*
p_2_3	188	*
p_2_10	1000	*
p_3_1	**78**	*
p_3_3	*	*
p_3_10	2800	*
p_4_1	15	16
p_4_3	47	47
p_4_10	156	141

Table 4.4: Computation time (ms) for value choice heuristics. An * indicates a computation time larger than 10 minutes.

4.6.3 Effects of improved variable selection

We test 4 different value selection heuristics. The heuristics are actually sorting strategies for the decision variable table, as we force the solver to instantiate decision variables according to the ordering of the decision variable table. We test the following sorting strategies:

- Increasing Pickup time (sorting strategy A)
- Increasing Pickup time, decreasing Order Amount (sorting strategy B)
- Increasing Pickup time, decreasing Compartment Capacity (sorting strategy C)
- Increasing Pickup time, decreasing Compartment Capacity, CompartmentId (sorting strategy D)

Table 4.5 shows the times in milliseconds to find the first feasible solution for all combinations of test problems and sorting strategies. All approaches apply the new global constraints and the top-down value choice heuristic. The best time for each problem is printed bold. The fourth sorting strategy - sorting by increasing pickup time, then by decreasing compartment capacity, and then by all orders for the same compartment - performs best on average. The large differences in running time between strategies emphasize the importance of the variable selection heuristic: even with reduced domains by applying the global constraints mentioned in Section 4.5.1, it takes over 10 minutes to find a solution for some of the problems with less effective sorting strategies.

	A	B	C	D
p_1_1	5090	**265**	4890	282
p_1_3	*	**391**	*	392
p_1_10	*	**860**	*	940
p_2_1	78	*	63	**62**
p_2_3	188	*	**172**	**172**
p_2_10	1000	*	**922**	956
p_3_1	**78**	327	**78**	**78**
p_3_3	*	*	*	*
p_3_10	2800	58000	**2720**	3160
p_4_1	**15**	93	31	**15**
p_4_3	**47**	690	**47**	**47**
p_4_10	**156**	32740	**156**	172

Table 4.5: Computation time (ms) for variable selection heuristics. An * indicates a computation time larger than 10 minutes.

Percentage reduction:	10%	20%	30%	40%	50%	60%	70%
p_1_1	>60	>60	>60	>60	>60	22	<1
p_2_1	>60	>60	>60	10	<1		
p_3_1	>60	>60	57	52	52	32	<1
p_4_1	>60	1,5	<1				

Table 4.6: Required time (s) for the CP approach to identify problem infeasibility. The computation times larger than 60 seconds are indicated with >60.

4.6.4 Infeasibility detection

To test how fast the CP approach detects problem infeasibility, we reduce the summed compartment capacity, making the gap to feasibility larger in each step. As mentioned earlier, this is just one out of many ways in which the problem can be infeasible. These tests are merely intended to get an idea of how the CP algorithm performs for various *degrees* of problem infeasibility. We test with the optimal CP approach determined in previous sections; using global constraints, decreasing value selection, and sorting strategy D. Table 4.6 indicates for four test problems the time in seconds until problem infeasibility is established by the solver, quiting the solver if it is still running after 60 seconds. We increase the gap to feasibility until it takes less than a second for the CP approach to detect infeasibility. The results indicate that fast detection of problem infeasibility is a weakness of the CP approach: even for problems that are by far not feasible - coming short up to 50 percent of the required capacity to transport all orders - the solver still requires approximately a minute or more to establish infeasibility for one of the problems, where each of the problems are solved within half a second if they are feasible.

4.6.5 Conclusions

In this chapter we propose a basic CP algorithm to solve the load assignment problem, and we propose and test various algorithm improvements. We demonstrate the improved pruning power of global versus elementary constraints, reducing the required time to find a solution by as much as 99%. We further optimize the CP approach by testing various heuristics for variable and value selection.

Despite all algorithm improvements, none of the CP approaches is capable of solving each test problem in a small time frame (in the order of seconds). Even the overall best approach performs weak in case of problem infeasibility. A weakness of the approach is the fact that Constraint Programming enumerates the feasible values of the domains of decision variables. For the load assignment problem, this means enumerating order amounts, and despite the application of heuristics that enable 'smarter' enumeration, in some cases the search is still very inefficient.

Chapter 5 presents an ILP approach for the load assignment problem, and compares modeling and performance aspects of both load assignment approaches.

Chapter 5

Benchmark ILP Approach

To objectively assess the CP approach to solving the load assignment problem we develop an Integer Linear Programming approach that serves as benchmark. We use the ILP technology because of certain similarities with CP, and the fact that Integer Linear Programming is such a well known and widely used technology for a large range of problems. We use the ILP approach as a benchmark, testing it with the same set of problems used for the CP approach to put the computational speed of the CP approach into perspective. Also, we compare the qualitative aspects of modeling and solving a problem using CP and ILP.

This chapter is organized as follows. Section 5.1 describes the development approach of the benchmark ILP algorithm. We develop an ILP approach based on the model that was presented in Section 2.4. Section 5.2 proposes modifications to this approach for improving the solving efficiency. Section 5.3 presents the results from testing the original and improved ILP approach with the same set of problem instances used for the CP approach. Section 5.5 presents the conclusions of this chapter.

5.1 ILP algorithm development

The ILP approach uses Matlab to read problem parameters and construct the ILP representation of the problem, and CPLEX 11.2 to solve the actual problem instances. The modular design of the Matlab 'interface' provides the following opportunities:

- switch individual restrictions on or off without requiring extensive model changes,
- read parameters from the same input files used for the Constraint Programming approach,
- automatically solve all problem instances with 'one push of the button',

- write all relevant output for solved problem instances in a readable format to a Microsoft Excel sheet.

The design of the Matlab interface makes it possible to apply and test model changes relatively easy. Section 5.2 proposes such a change, and Section 5.3 presents the results of the tests of the original and improved ILP benchmark approaches.

Optimization criterion

As the load assignment problem is a feasibility problem, there is no cost function to optimize. However, using an optimization criterion may improve the computation time as it provides bounds that are used in the Branch and Bound algorithm for solving an ILP problem. For this reason we use the following objective function:

$$\textit{Maximize} \quad \sum_{i \in O} \sum_{j \in C} x_{ij} \tag{5.1}$$

Although this expression is constant for any feasible solution, it improves the required computation time for solving problem instances.

5.2 ILP algorithm improvements

The ILP formulation in Section 2.4 is derived from an ILP algorithm that is currently used in a development version of OTD for solving the distribution problem. Therefore, improvement of the problem solving speed due to a better ILP formulation is not only of interest for this study. It possibly improves the current ILP solution in OTD as well.

One way of improving the performance of the ILP approach is to reduce the amount of binary variables in the ILP formulation. These binary variables act as branching nodes in solving a (Mixed) Integer Linear Program. Reducing the amount of binary variables reduces the amount of required branching decisions and generally improves the computational speed. In the current formulation, for every (continuous) decision variable x_{ij} there is a binary variable δ_{ij} that we use in the compatibility constraints. We argue that we reduce the amount of required binary variables by grouping compatible orders into *compatibility clusters*, and only allow the assignment of orders out of one cluster to a compartment. The most obvious clustering criterion is clustering by order products, but additional clusters may be necessary to model for example different loading temperatures. Every order belongs to exactly one cluster.

In the worst-case scenario, every order requires its own cluster. In this case, we require exactly the same amount of binary auxiliary variables as in the original model: $i \cdot j$.

However, in the likely situation that there are less clusters than orders, we reduce the amount of required binary auxiliary variables. Solving the problem therefore requires less branching, possibly improving the required computation time for solving problem instances.

We propose to extend the ILP model from Section 2.4 with the following elements:

Sets

Clusters $\boldsymbol{Set\ P\ (index\ n)}$

Parameters

$\boldsymbol{OrderCluster_{in}}$ Binary parameter: 1 if order $\boldsymbol{i}$ is in cluster $\boldsymbol{n}$, 0 otherwise

Auxiliary Variables

$$\boldsymbol{C_{jn}} = \begin{cases} \mathbf{1} & \text{if at least one order of cluster } \boldsymbol{n} \text{ is assigned to compartment } \boldsymbol{j} \\ \mathbf{0} & \text{otherwise} \end{cases}$$

Constraints

If any amount of order $\boldsymbol{i}$ is assigned to compartment $\boldsymbol{j}$, the variable $\boldsymbol{C_{jn}}$ must have value 1 for the cluster of order $\boldsymbol{i}$:

$$\boldsymbol{Demand_i \cdot C_{jn} \geq x_{ij} \cdot OrderCluster_{in}} \qquad \boldsymbol{\forall i, j, n} \tag{5.2}$$

A compartment can only be assigned to one cluster:

$$\sum_{\boldsymbol{n \in P}} \boldsymbol{C_{jn}} \leq \mathbf{1} \qquad \boldsymbol{\forall j} \tag{5.3}$$

A cluster cannot be assigned to a compartment if its products are not compatible with the compartment (replacing constraints 2.10):

$$\boldsymbol{C_{jn} = 0} \qquad \text{For all incompatible compartments j and clusters n} \tag{5.4}$$

Furthermore, we omit all constraints 2.11 as they are no longer necessary.

Note that we still require δ_{ij} in case of single-order compartments; for problem instances with single order compartments we still add these variables, but only for the single-order

compartments. To measure the effects of this proposed model improvement, we test both ILP approaches with the set of test problems introduced in Section 4.4. Section 5.3 presents the computation times for these tests.

5.3 Test results

Testing the ILP approaches presented in Sections 2.4 and 5.2 provides a benchmark for the computation times of the Constraint Programming approach, and illustrates the effect of our improved ILP approach. The presented computation times, measured in milliseconds, do not include building the model in Matlab; they represent the pure CPLEX solving time. Note that the ILP solutions are optimal instead of feasible, although any feasible solution is just as good as another because of the objective function we use (see equation 5.1).

For both the original and the improved ILP model, we test two ILP approaches:

1. A model with continuous decision variables x_{ij}, which we refer to as the 'benchmark' ILP, and
2. A model with integer decision variables x_{ij}, which we refer to as the 'equivalent' ILP.

For solving the load assignment problem, the assigned order amounts are allowed to be continuous. In theory, an ILP model with continuous decision variables is easier to solve than an ILP model with integer decision variables. Therefore we use the ILP model with continuous decision variables as the performance benchmark for solving the load assignment problem.

To extend the comparison between CP and ILP beyond the application to the specific load assignment problem, we also test the exact same problem. As CP allows only integer decision variables, the equivalent ILP approach has integer decision variables. The comparison between a CP approach and an ILP approach with integer decision variables is interesting as we consider other applications - that might require integer decision variables - for CP in Chapter 6. Note that both ILP approaches have other binary variables: the auxiliary δ_{ij} variables in the original model, or the auxiliary C_{jn} variables in the improved model.

5.3.1 Effects of ILP model improvements

Tables 5.1 and 5.2 depict the computation times for the original and improved ILP approach for respectively the *benchmark* and *equivalent* model variants.

	Original ILP	Improved ILP	Reduction
p_1_1	1522	67	96%
p_1_3	248	676	-173%
p_1_10	1564	97	94%
p_2_1	284	1275	-349%
p_2_3	3097	114	96%
p_2_10	84	50	41%
p_3_1	104	1267	-1118%
p_3_3	3616	60	98%
p_3_10	80	29	64%
p_4_1	15	9	40%
p_4_3	19	8	58%
p_4_10	29	6	79%

Table 5.1: Computation times (ms) for original and improved benchmark ILP approach

	Original ILP	Improved ILP	Reduction
p_1_1	4400	322	93%
p_1_3	1396	156	89%
p_1_10	5385	122	98%
p_2_1	1695	296	83%
p_2_3	899	384	57%
p_2_10	1228	268	78%
p_3_1	7354	528	93%
p_3_3	777	910	-18%
p_3_10	*	1339	*
p_4_1	147	36	76%
p_4_3	23	9	61%
p_4_10	66	16	76%

Table 5.2: Computation times (ms) for original and improved equivalent ILP approach. An * indicates a computation time larger than 10 minutes.

	Benchmark ILP		Equivalent ILP	
Percentage reduction:	10%	20%	10%	20%
p_1_1	11.09	0.014	60	0.016
p_1_2	0.009	0.008	0.01	0.009
p_1_3	0.202	0.011	0.197	0.010
p_1_4	0.003	0.003	0.003	0.003

Table 5.3: Required time (s) for the ILP approaches to detect problem infeasibility

Overall we observe that the proposed modifications to the ILP model result in significant reduction of the computation time. Remarkably, for a few test problems the computation time actually increases dramatically. We have no explanation for this phenomenon. As the overall effect of the proposed improvements is positive, especially for the equivalent ILP approach, we use the computation times of the ILP approach with the proposed improvements to compare ILP with CP.

5.3.2 Infeasibility detection

Similar to the CP approach, we also consider the behavior of the ILP approach in case of problem infeasibility. Performing exactly the same infeasibility detection test as described in Section 4.6.4, we measure how well the ILP approach detects problem infeasibility. Table 5.3 presents the computation times in seconds the ILP approach requires to detect problem infeasibility. Again, we are not interested in solution times larger than 60 seconds as this would be too long for the purpose of our load assignment algorithm, and we indicate these cases with >60. In general, the ILP approach is very fast in determining problem infeasibility. As expected, the required time to detect infeasibility decreases when the gap to feasibility increases.

5.4 Comparison of CP and ILP

5.4.1 Computation times

To evaluate the speed of the CP approach for solving the load assignment problem, we compare the computation times of our CP approach to those of the benchmark ILP approach. To evaluate the speed of CP in comparison to ILP for problems that do require integer decision variables, we also compare the CP approach computation times to those of the equivalent ILP approach.

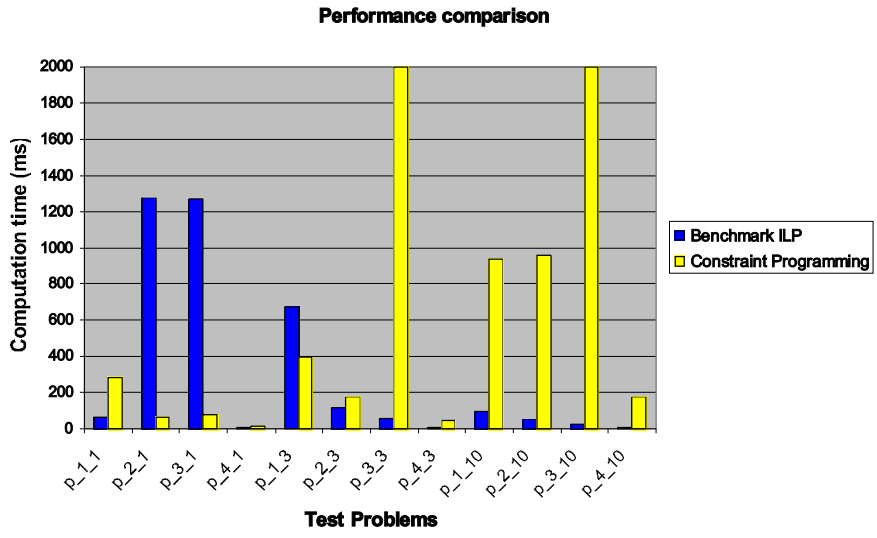

Figure 5.1: Computation times (ms) for CP and benchmark ILP approach

Constraint Programming versus benchmark ILP

Table 5.4 presents the computation times in milliseconds for the test problems for the CP and benchmark ILP approaches. To demonstrate the influence of parameter values on the performance of both approaches, we depict the computation times in a bar chart, clustering the test problems with similar parameter values. Figure 5.1 depicts the computation times for both approaches. The results do not indicate that one approach is always better, but the ILP approach outperforms the CP solution for 75% of the problems. There is an opposite trend in computation times for the two approaches with respect to the parameter values. Where the CP approach - according to our expectations - is consistently slower for larger problems, the ILP approach actually solves the larger problems faster: the ILP approach benefits from the increased amount of possible solutions.

Table 5.5 presents the results for the infeasibility tests for the CP and the benchmark ILP approaches. The percentages represent the gap to problem feasibility for which the algorithms require less than a second to detect that there is no feasible solution. Section 4.4.2 explains exactly how we construct this gap. For the detection of problem infeasibility, the ILP approach clearly outperforms the CP approach, as the CP approach requires a much larger gap for (relatively) fast infeasibility detection.

	CP solution	benchmark ILP
p_1_1	282	**67**
p_1_3	**392**	676
p_1_10	940	**97**
p_2_1	**62**	1275
p_2_3	172	**114**
p_2_10	956	**50**
p_3_1	**78**	1267
p_3_3	*	**60**
p_3_10	3160	**29**
p_4_1	15	**9**
p_4_3	47	**8**
p_4_10	172	**6**

Table 5.4: Computation times (ms) for CP solution and benchmark ILP solution. An * indicates a computation time larger than 10 minutes.

	CP approach	benchmark ILP
p_1_1	70%	20%
p_2_1	50%	10%
p_3_1	70%	10%
p_4_1	30%	10%

Table 5.5: Degree (%) of problem infeasibility that is detected within 1 second

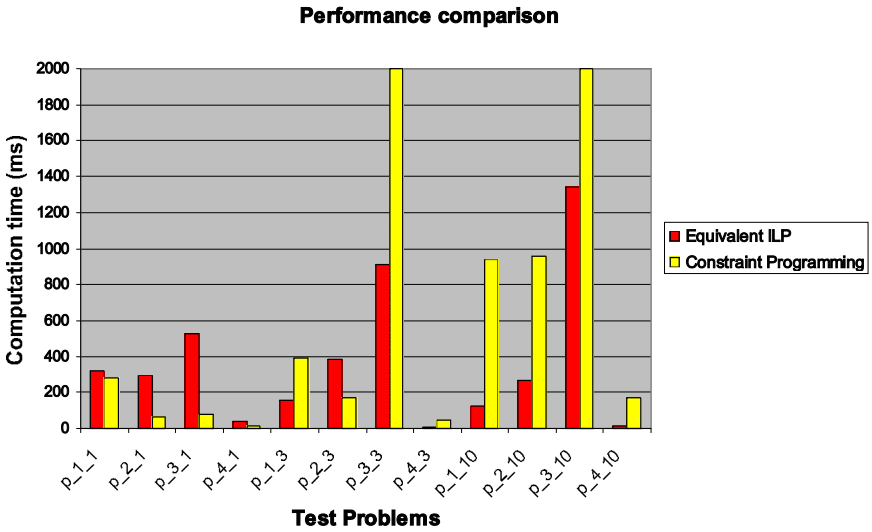

Figure 5.2: Computation times (ms) for CP and benchmark ILP approach

Constraint Programming versus equivalent ILP

Table 5.6 presents the computation times in milliseconds for the the CP and equivalent ILP approaches. Figure 5.2 depicts the computation times for both approaches. As expected, the computation times of the equivalent ILP approach are closer to the computation times of the CP approach compared to the computation times of the benchmark ILP approach. For the smaller problem variants (p_x_1), CP performs better than the equivalent ILP. However, for larger parameter values the equivalent ILP approach performs best. The results of the infeasibility tests of the equivalent ILP are similar to the results of the benchmark ILP: both the benchmark and equivalent ILP approaches perform much better than the CP approach.

5.4.2 Modeling comparison

Constraint Programming and (Integer) Linear Programming have many similarities with respect to the basic elements for modeling a problem. Both technologies require a set of decision variables, and use constraints to represent the relations between the decision variables. Other common modeling elements are parameters (used as weights and bounds in constraints), auxiliary variables, and an objective function representing the total cost or benefit of the decision variable values. Constraint Programming requires explicit specification of decision variable domains, and (I)LP applies domains implicitly through

	CP approach	equivalent ILP
p_1_1	**282**	322
p_1_3	392	**156**
p_1_10	940	**122**
p_2_1	**62**	296
p_2_3	**172**	384
p_2_10	956	**268**
p_3_1	**78**	528
p_3_3	*	**910**
p_3_10	3160	**1339**
p_4_1	**15**	36
p_4_3	47	**9**
p_4_10	172	**16**

Table 5.6: Computation times (ms) for CP and equivalent ILP approaches. An * indicates a computation time larger than 10 minutes.

constraints, with similar effect.

Constraints

The main difference in modeling between CP and ILP is in expressing the relations between decision variables. Mathematical Programming models (LP, ILP, MILP) consist of a set of linear (in)equalities, restricting the way to express the relations between variables to the use of linear (in)equalities. Expressing more complex relations between variables in a set of linear inequalities is not always trivial. There are many modeling tricks to express non-linear relations such as either-or constraints, absolute values, and conditional constraints, but models for problems with many of such relations may become very complex. Solving a problem by Mathematical Programming requires a translation of the actual problem into components that are suitable for the solving mechanism, limiting the modeling freedom.

Constraint Programming aims to remove exactly this limitation of other solving technologies: CP provides many expressive options to model relations between decision variables without having to bother about the problem solving mechanics. Advocates of the Constraint Programming technology claim that it is one of the best approaches in computer science to reach 'the holy grail of programming': the user states the problem, the computer solves it. The expressive global constraints in CP provide modeling opportunities that are much more difficult to achieve with Mathematical Programming. Especially for problems with many non-trivial constraints, a CP solution requires a much less complex model, keeping the algorithm more flexible for future adjustments.

Solving technology

The CP solving technology allows for much control over the problem solving mechanics. This is both a strength and a weakness of CP. It is a strength in the sense that CP provides a far reaching control over the solving mechanism through the combination of powerful global constraints provided in a modeling language, and the modeling options for variable and value choices in the steps of problem solving. Solving a problem with CP therefore requires less programming skills and tools than is required for development of a dedicated search heuristic. Yet the solving mechanics are not based on powerful mathematical concepts, unlike for instance ILP (for which similar methods to control the solving mechanics are possible). Therefore the solving speed of CP approaches relies much more on problem characteristics and the appropriate use of variables and constraints.

5.5 Conclusions

The ILP approach provides an objective benchmark for both qualitative as quantitative aspects of solving the load assignment problem. In this chapter we presented an ILP model for the load assignment problem based on an algorithm in OTD, and we proposed improvements to this model that may also improve the development version of OTD.

The comparison of the computation times of the CP and ILP approaches in Section 5.4.1 reveals that the speed is a weakness of the Constraint Programming approach. The ILP approach performs better on average for both feasible and infeasible problems. As the combined optimization of load and routes in OTD requires a very fast load assignment algorithm, we advise not to use a Constraint Programming approach. However, note that the improved ILP approach does not meet the performance requirements either.

In the comparison of the computation times between CP and ILP we demonstrate the speed is a weakness of CP. The CP approach iterates the feasible parts of the decision variable domains. For domains that represent all possible amounts of a liquid that may be assigned to a set of compartments, this amounts to an inefficient solving process especially for larger order sizes. CP outperforms ILP for smaller problems, but as order sizes increase, the CP speed deteriorates.

The strength of CP is the ability to translate a wide variety of realistic constraints into an algorithm, without introducing modeling complexity, and maintaining a high level of algorithm flexibility. The weakness of CP is its speed for solving large problems. The load assignment problem requires a fast algorithm that is allowed to be complex. Therefore, there is a mismatch between the requirements of the algorithm and what CP

has to offer. Development of a new dedicated load assignment heuristic is a more sensible option, assuming it provides greater speed at the cost of algorithm flexibility.

Chapter 6

Constraint Programming Opportunities

In Chapter 5 we concluded that the Constraint Programming approach does not meet ORTEC's specific requirements for solving the load assignment problem. In this chapter we propose three applications where Constraint Programming might be a suitable approach:

- Load assignment variants,
- Workforce scheduling,
- Tactical Route Planning.

For each of these applications we briefly describe the characteristics of the underlying problem. We propose a modeling approach, argue what the added value of Constraint Programming is to the application, and indicate how ORTEC may benefit from the application of Constraint Programming. Section 6.4 presents a brief overview of the global constraints mentioned in this chapter.

6.1 Load assignment variants

The application of Constraint Programming to the load assignment problem is not successful because of the required computational speed in combination with the large domains of the decision variables. The possibility to divide order load over various compartment causes these large domains. If we remove the possibility to divide order load, the decision becomes whether or not to use a compartment for an order.

Modeling approach

For the load assignment variant that does not allow division of load over multiple compartments, we replace the original decision variable $Amount_{oc}$ with a boolean variable $IsUsed_{oc}$, where $IsUsed_{oc} = 1$ if order o is entirely assigned to compartment c, and 0 otherwise. Modification of the constraints is trivial, proving the flexibility of the Constraint Programming approach. Instead of constraints on $Amount_{oc}$, the order amount is now given, and constraints apply to $(Amount_o \cdot IsUsed_{oc})$.

Added value to ORTEC

At least one important customer of OTD actually transports indivisible orders and has similar requirements for optimizing load and routes. The modified CP approach would be very suitable to apply in the customized version of OTD for this customer. However, this load assignment problem variant is easier and probably faster to solve for the ILP approach too. Advantages of the CP approach are on the one hand the flexibility of the algorithm, and on the other hand the costs of the required tools. The benchmark ILP approach uses the CPLEX solver, which is much more expensive than the Inovia tools. For a lot of OTD customers, buying a CPLEX license just for solving the load assignment problem is not an option. Other (I)LP solvers are less expensive, but they may be slower too. Therefore, the CP approach that we have tested is a good alternative for customers that require a flexible algorithm and that do not want to buy a CPLEX license.

6.2 Workforce scheduling

Workforce scheduling refers to the assignment of employees to shifts or duties over a certain period of time. Workforce scheduling aims to produce a schedule for each employee of an organization. A workforce schedule must satisfy rules and collective labor agreement regulations, and deals with the trade off between satisfying the personal preferences of employees and minimizing the total staffing costs. The problem of finding a good quality schedule can be incredibly difficult, especially in organizations with flexible demand and irregular working hours, that require 24/7 availability of highly skilled staff, and are subject to extensive workforce legislation. Scheduling a hospital staff is probably the best example of such a difficult workforce scheduling problem.

The workforce scheduling problem is an optimization problem. Optimization problems are difficult for Constraint Programming in terms of the required computation time to find the optimal solution. However, finding an optimal workforce schedule is not subject to such a tight time constraint as solving the load assignment problem is. It does not matter that an algorithm requires 10 minutes to determine the optimal workforce

schedule for a month. We argue that CP does have advantages that make it a suitable approach for workforce scheduling.

ORTEC has done much research on workforce scheduling for its standardized scheduling solution Harmony. We base the problem description and modeling approach in this section on *van Draat et al.* (2006), *Burke et al.* (2008), and on internal documents.

Modeling approach

To represent the problem we propose binary decision variables x_{ij} linking a shift i to an employee j. For modeling the restrictions we use two kinds of constraints: *hard* and *soft* constraints. Hard constraints represent restrictions due to national legislation and collective labor agreements; they have to be satisfied to obey the law. Soft constraints represent the preferences of employees or the organization. Such constraints should be satisfied as much as possible to fulfill personal preferences. An optimal schedule satisfies all hard constraints and as much soft constraints as possible, against minimal staffing costs. The relative importance of a constraint is typically represented by a *weight* or *cost penalty* for violating the constraint.

In real-life situations, the number of required constraints can be very high. The extensive labor regulations in health care, and the fact that no two employees are the same, make that problems with up to 100 constraints are normal.

Advantages of Constraint Programming

Due to the amount and the character of the required constraints, and the type of decision variables, Constraint Programming is very suitable for modeling and solving workforce scheduling problems. Various global constraints typically express the required relations between decision variables. We present some examples:

- Use the *GlobalCardinality* constraint to indicate for each shift the exact required amount of employees with certain skills. For example, shift X requires exactly 1 surgeon, 2 nurses, and at least 2 assistants.
- Use the *Count* constraint to bound the number of shifts of a certain type that can be assigned to each employee. For example, allow a maximum of two weekend shifts for each employee.
- Use the *Period* constraint to force a period on the shift types of employees. For example, force the repetition of a shift sequence (day, day, day, night, free, free, day) for employees at least every 14 days.

- Use the *AllDiff* constraint to assign unpopular shift types at most once to the same employee within the planning horizon. For example, assign every weekend night shift for nurses within one month to a different nurse.

Furthermore, CP tools like Inovia's Visual Inspector provide the opportunity to analyze the effects of changing the weights of various soft constraints. Typical for real-life implementations of scheduling algorithms, getting the right configuration of (soft) constraint weights is a time consuming practice. The Visual Inspector provides insight in the contribution of each constraint to the overall cost of a solution, and allows for quick and easy adjustment of the weights of (groups of) constraints. This makes configuring the algorithm both easier and less time consuming.

Another advantage of a Constraint Programming approach is the flexibility to add or modify constraints. Planning human resources is subject to many personal factors, and dealing with changing staff configurations, skills, and employee preferences requires the algorithm to be flexible, which is a strength of Constraint Programming.

Added value to ORTEC

Harmony is one of ORTEC's major standardized products, which specializes in solving a wide variety of workforce scheduling problems. We argue that Constraint Programming may be used to the advantage of Harmony in three different ways:

1. Future functional extensions, especially with respect to adding non-trivial constraints or sub problems.

2. Use CP to provide direct feedback on manual scheduling decisions.

3. Fast prototyping at potential new Harmony customers to demonstrate the scheduling possibilities, without the need to configure an entire Harmony environment to the specific situation of the customer, which is much more time consuming.

Option (2) uses the way CP solves a problem. If a planner manually assigns a shift to a person, this is a (manual) variable instantiation. The domain reductions that result from constraint propagation after the manual variable instantiation may be used to provide relevant information to the planner. The planner directly sees the impact of its planning decision on the remaining (unassigned) shifts and resources. This functionality is useful for manual scheduling, or for manual modification of an (automatically) generated workforce schedule.

In option (3) CP is not used as the final algorithm, but ORTEC may capitalize on the

fast prototyping possibilities with Inovia's CP tools, translating customer requirements into a prototype in real time, providing a convincing demonstration of the possibilities that are actually included in Harmony.

6.3 Tactical Route Planning

Tactical Route Planning (TRP) aims to achieve a tactical transportation schedule that balances the workload and minimizes transportation costs for the repetitive delivery to groups of customers. The goal is to balance the periodic deliveries from a depot over a certain time period, such that all expected demand is delivered with a minimal fleet size, and furthermore cluster the required periodic deliveries geographically in order to minimize the required traveling distance.

TRP uses aggregate customer demand estimations in combination with actual required delivery frequencies. Translating delivery frequencies to fixed delivery schemes, for example deliver *every Monday and Friday*, allows balancing the demand over a period of time. A TRP solution is a 'static' plan that indicates for each customer the days it will be delivered, and the expected number of required resources. This static plan is the basis for operational delivery planning. TRP allows planners to manage the required fleet and workforce capacity much further in advance than operational planning.

Modeling approach

A solution to a TRP links every customer to a schedule, where a schedule indicates for every day the planned route and resources. There is a schedule for every geographical cluster, and the schedules respect the delivery schemes of each customer. The following global constraints are very useful for the various aspects of TRP:

- The *Balance* constraint is appropriate for balancing the required customer deliveries over the days of the week,
- The *Period* constraint is useful for modeling the required delivery frequency,
- The *Circuit* constraint can be applied for every cluster of customers, forcing a route in which every customer is visited exactly once.

This combinatorial problem requires cost optimization, which is hard for Constraint Programming in terms of speed. However, companies will typically perform such an analysis periodically, once every few weeks or even months. For this application, it does not matter that the CP approach is not fast, as long as the best solution is found.

Added value to ORTEC

TRP is currently only implemented in the OTD predecessor SHORTREC. In time, ORTEC intends to replace SHORTREC by OTD, requiring the integration of the TRP functionality in future versions of OTD. Constraint Programming certainly is an interesting option for the future integration of TRP in OTD.

6.4 Overview of global constraints

In this chapter we indicate possible successful applications of CP for ORTEC. We introduce several global constraints in the consecutive modeling approaches. This section briefly describes the global constraints that are introduced in this chapter.

GlobalCardinality

The input arguments for the GlobalCardinality constraint are a set of decision variables, a set of values, and for each of these values the required number of occurrences. The GlobalCardinality constraint makes sure that each value is assigned to the decision variables exactly as often as required.

Count

The required input arguments for the Count constraint are a set of decision variables, a value v, a number of occurrences n, and a comparison operator op ($\leq$, $\geq$, $=$, etc.). The Count constraint forces the actual number of occurrences of the value v in the set of decision variables to satisfy the bound that is indicated by the required number of occurrences n and the comparison operator op. For example: $Count(x_1, ..., x_4; 5; 2; \leq)$ makes sure that at most two variables from $\{x_1, ..., x_4\}$ are assigned the value 5.

Period

The Period constraint forces (or prevents) a repetition of variable values within a certain period. The input arguments are a set of decision variables, a period value p, and a comparison operator op ($\leq$, $\geq$, $=$, etc.). For example: $Period(x_1, ..., x_9; 3; =)$ forces a period of length 3 on the decision variables $\{x_1, ..., x_9\}$. A feasible assignment of values is for example $\{x_1, ..., x_9\} = \{4, 2, 9, 4, 2, 9, 4, 2, 9\}$.

Balance

The Balance constraint requires two input arguments: a set of decision variables, and a value v. The Balance constraint forces the difference in the number of occurrences of each individual variable value to be at most v. For example: the assignment of values

$\{x_1, ..., x_8\} = \{8, 1, 22, 8, 8, 22, 1, 8\}$ satisfies the global constraint $\textit{Balance}(x_1, ..., x_8; 2)$. The value 8 occurs four times, the values 1 and 22 occur two times. The difference between the number of occurrences does not exceed 2.

Circuit

The Circuit constraint works on a set of decision variables that represent nodes in a graph. The Circuit constraint forces a value assignment such that each node in the graph is visited exactly once, except for the starting node that is visited twice (start and finish of the tour). The Circuit constraint does not allow sub tours.

Chapter 7

Conclusions

In this report we applied Constraint Programming to a realistic load assignment problem. We have two goals: on the one hand increase ORTEC's knowledge of Constraint Programming, and on the other hand extend the current load assignment functionality in a new algorithm. Besides realistic functional extensions, ORTEC requires the new load assignment algorithm to be very fast, to be able to combine load and route optimization.

According to the literature, Constraint Programming is especially suitable for solving highly constrained problems to feasibility. We proposed a CP approach to solving the load assignment problem, and discussed and tested various efforts to improve the computational speed of the CP approach with a set of 12 test problems. We achieved computation time reductions up to 99% by replacing elementary constraints with new global constraints, and adding variable and value choice heuristics.

We developed an ILP benchmark approach to compare Constraint Programming to a well known technology. The results of the computation time comparison reveal that speed is a weakness of the CP approach. The CP approach does not meet the computation speed requirements, due to the fact that the CP approach does not handle the large domains of the decision variables fast enough. Therefore, we conclude that Constraint Programming is not suitable for solving the load assignment problem with the current computation time requirements.

The strength of Constraint Programming is the expressiveness of the global constraints, that allows users to focus on representing the problem, without worrying about how to solve it. The trade off is that CP solvers are not based on strong mathematical principles, and therefore the computational speed for solving large problems can be weak. In general, we do not advise to use Constraint Programming for solving standard problems that require a fast algorithm. Nor do we advise CP for applications that require little or

no modification over time: development of a dedicated fast heuristic is most likely worth the effort. The strengths of CP will not outweigh the weaknesses in these situations.

However, many complex business environments are subject to constant change, and require a flexible approach that allows modeling complex constraints. Also, for a lot of applications it is not a problem to wait a few seconds, minutes, or even hours for a good quality solution. In these cases Constraint Programming is a very suitable technology.

For ORTEC, it may be interesting to consider applying Constraint Programming to the load assignment problem with indivisible orders, for workforce scheduling, for Tactical Route Planning, and for any other problem that requires complex constraints and a flexible approach.

Bibliography

Abdelaziz, B., C. Roucairol, and C. Bacha (2002), Deliveries of liquid fuels to sndp gas stations using vehicles with multiple compartments, *2002 IEEE International Conference on Systems, Man and Cybernetics*, pp. 478–483.

Backer, B. D., P. Kilby, P. Prosser, and P. Shaw (2000), Solving vehicle routing problems using constraint programming and metaheuristics, *Journal of Heuristics*, *6*, 501–523.

Barták, R. (1999), Constraint programming: In pursuit of the holy grail, in *in Proceedings of WDS99 (invited lecture*, pp. 555–564.

Bessiere, C. (2006), Constraint propagation, in *Handbook of Constraint Programming*, edited by F. Rossi, P. van Beek, and T. Walsh, chap. 4, Elsevier.

Bessière, C., and P. V. Hentenryck (2003), To be or not to be...a global constraint, in *In Proceedings CP'03, Kinsale*, pp. 789–794, Springer-Verlag.

Burke, E. K., T. Curtois, G. Post, R. Qu, and B. Veltman (2008), A hybrid heuristic ordering and variable neighbourhood search for the nurse rostering problem, *European Journal of Operational Research*, *188*(2), 330–341.

Chajakis, E. D., and M. Guignard (2003), Scheduling deliveries in vehicles with multiple compartments, *J. of Global Optimization*, *26*(1), 43–78, doi: http://dx.doi.org/10.1023/A:1023067016014.

Corréa, A. I., A. Langevin, and L.-M. Rousseau (2007), Scheduling and routing of automated guided vehicles: A hybrid approach, *Computers & OR*, *34*(6), 1688–1707.

Fallahi, A. E., C. Prins, and R. W. Calvo (2008), A memetic algorithm and a tabu search for the multi-compartment vehicle routing problem, *Comput. Oper. Res.*, *35*(5), 1725–1741, doi:http://dx.doi.org/10.1016/j.cor.2006.10.006.

Fisher, M. L., and R. Jaikumar (1981), A generalized assignment heuristic for vehicle routing, *Networks*, *11*(2), 109–124.

Lambert, G. R. (2004), A tabu search approach to the strategic airlift problem.

Repoussis, P. P., C. D. Tarantilis, and G. Ioannou (2006), A hybrid metaheuristic for a real life vehicle routing problem., in *Numerical Methods and Applications*, *Lecture Notes in Computer Science*, vol. 4310, edited by T. Boyanov, S. Dimova, K. Georgiev, and G. Nikolov, pp. 247–254, Springer.

Semet, F., and E. Taillard (1993), Solving real-life vehicle routing problems efficiently using tabu search, *Ann. Oper. Res.*, *41*(1-4), 469–488.

Shaw, P. (1998), Using constraint programming and local search methods to solve vehicle routing problems, pp. 417–431, Springer-Verlag.

van Draat, L. F., G. F. Post, B. Veldman, and W. Winkelhuijzen (2006), Harmonious personnel scheduling, *Medium Econometrische Toepassingen 14*, Rotterdam.

Appendix A

Test Problems

code	orders	order size	compartments	compartment capacities	ullage rate rule	different products
p_1_1	32	5 - 20	7	15 - 33	35 - 65 95 - 99	4
p_1_3	32	15 - 60	7	45 - 99	35 - 65 95 - 99	4
p_1_10	32	50 - 200	7	150 - 330	35 - 65 95 - 99	4
p_2_1	18	10 - 60	6	40 - 50	40 - 60	3
p_2_3	18	30 - 180	6	120 - 150	40 - 60	3
p_2_10	18	100 - 600	6	400 - 500	40 - 60	3
p_3_1	15	10 - 100	5	50 - 150	45 - 75	3
p_3_3	15	30 - 300	5	150 - 450	45 - 75	3
p_3_10	15	100 - 1000	5	500 - 1500	45 - 75	3
p_4_1	15	20 - 90	3	110	50 - 60	3
p_4_3	15	60 - 270	3	330	50 - 60	3
p_4_10	15	200 - 900	3	1100	50 - 60	3

Table A.1: Test problem parameters

Appendix B

AQL Load Assignment model

Algorithm 3 Parameter tables

```
//  parameter tables:
AQL_TABLE Orders WITH_TITLE
(
   Id          NUMAUTO,
   Product     CHAR,
   Amount      NUMBER,
   PickupTime  NUMBER,
   DeliverTime NUMBER
) ;
AQL_TABLE OrderCombinations WITH_TITLE
(
   Number         NUMAUTO,
   PickupTime     NUMBER,
   DeliverTime    NUMBER
) ;
AQL_TABLE  Compartments WITH_TITLE
(
   CompartmentId    NUMBER,
   Capacity         NUMBER,
   ConfigurationId  NUMBER,
   Rule             NUMBER,
   SingleOrder      BOOLEAN
) ;
AQL_TABLE  Configurations WITH_TITLE
(
   ConfigurationId NUMBER,
   Capacity        NUMBER
) ;
AQL_TABLE  Axles WITH_TITLE
(
   AxleId    NUMBER,
   MaxWeight NUMBER
) ;
```

Algorithm 4 Parameter and duplication tables

```
AQL_TABLE  UllagerateRules WITH_TITLE
(
   RuleId     NUMBER,
   LowerBound REAL,
   UpperBound REAL
) ;
AQL_TABLE  AxleWeightContr WITH_TITLE
(
   CompartmentId    NUMBER,
   AxleId           NUMBER,
   Contribution     REAL
) ;
AQL_TABLE Compatibility WITH_TITLE
(
   Product          CHAR,
   CompartmentId    NUMBER
) ;
//  duplication tables:
AQL_DUP_TABLE SplitOrders FROM Orders
(
   OrderId         FROM Id NUMBER,
   Product         CHAR,
   Amount          NUMBER,
   PickupTime      NUMBER,
   DeliverTime     NUMBER,
   SplitOrder      DUPLICATION sup(0, count(Compartments) - 1),
   CompartmentId   NUMBER SET SplitOrder + 1,
   CompCapacity    NUMBER SET Compartments[CompartmentId].Capacity,
   ConfigurationId NUMBER SET Compartments[CompartmentId].ConfigurationId
) ;
AQL_TABLE Amounts FROM_EMPTY
(
   AffAmount    DUPLICATION FOR_EACH ?o FROM Orders { Orders[?o].Amount  }
) ADD_LINE "0" ;
//  table with decision variables:
AQL_TABLE result LINK SplitOrders WITH Amounts WITH_TITLE
(
   OrderId AS SplitOrders.OrderId,
   Product AS SplitOrders.Product,
   CompartmentId AS SplitOrders.CompartmentId,
   ConfigurationId AS SplitOrders.ConfigurationId,
   CompartmentCapacity AS SplitOrders.CompCapacity,
   //  To be chosen by solver:
   AffAmount AS Amounts.AffAmount
) AS "Order[%OrderId][%CompartmentId][%Product]";
```

Algorithm 5 constraints I

```
//  constraints:
AQL_SELECT SUCH_AS
{
   // These constraints are independent of the orders on board
   GROUP SHOW "0 - Assign total amount "
   {
      FOR_EACH ?o FROM Orders
         {
            sum( FOR_EACH ?r FROM result
                 VERIFYING result[?r].OrderId = Orders[?o].Id
                 { result[?r].AffAmount }
               )
               =   Orders[?o].Amount ;
         } ;
   };
   GROUP SHOW "1 - Compatibility product-compartment"
   {
      FOR_EACH ?S FROM SplitOrders
      VERIFYING 0 = sum( FOR_ONE ?C FROM Compatibility
                         VERIFYING ( Compatibility[?C].CompartmentId = SplitOrders[?S].CompartmentId )
                         AND ( SplitOrders[?S].Product = Compatibility[?C].Product) {1} )
      {
         result[?S].AffAmount = 0;
      };
   } ;
   GROUP SHOW "2 - Compatibility product-product"
   {
      FOR_EACH ?p1 FROM SplitOrders
      {
         FOR_EACH ?p2 FROM SplitOrders
         VERIFYING SplitOrders[?p1].Product <> SplitOrders[?p2].Product
         AND SplitOrders[?p1].CompartmentId = SplitOrders[?p2].CompartmentId
         {
            ( result[?p1].AffAmount = 0 ) OR ( result[?p2].AffAmount = 0 );
         };
      };
   };
   GROUP SHOW "3 - SingleOrderProperty"
   {
      FOR_EACH ?p1 FROM SplitOrders
      {
         FOR_EACH ?p2 FROM SplitOrders
         VERIFYING ( (SplitOrders[?p1].CompartmentId = SplitOrders[?p2].CompartmentId )
         AND         ( Compartments[ SplitOrders[?p1].CompartmentId ].SingleOrder )
         AND         ( SplitOrders[?p1].OrderId <> SplitOrders[?p2].OrderId ) )
         {
            ( result[?p1].AffAmount = 0 ) OR ( result[?p2].AffAmount = 0 );
         };
      };
   } ;
```

Algorithm 6 Constraints II

```
// These constraints are dependent of the orders on board
   GROUP SHOW "4 - Capacity "
   {
      FOR_EACH ?O FROM OrderCombinations
      {
         FOR_EACH ?c FROM Compartments
         {
            sum( FOR_EACH ?s FROM result
                 VERIFYING SplitOrders[?s].CompartmentId = Compartments[?c].CompartmentId
                 AND       SplitOrders[?s].PickupTime  <= OrderCombinations[?O].PickupTime
                 AND       SplitOrders[?s].DeliverTime >= OrderCombinations[?O].DeliverTime
                 { result[?s].AffAmount } ) <= Compartments[?c].Capacity ;
         } ;
         FOR_EACH ?k FROM Configurations
         {
            sum( FOR_EACH ?s FROM result
                 VERIFYING result[?s].ConfigurationId = Configurations[?k].ConfigurationId
                 AND SplitOrders[?s].PickupTime  <= OrderCombinations[?O].PickupTime
                 AND SplitOrders[?s].DeliverTime >= OrderCombinations[?O].DeliverTime
                 { result[?s].AffAmount } ) <= Configurations[?k].Capacity
         };
      };
   };
   GROUP SHOW "5 - Ullagerates "
   {
      FOR_EACH ?O FROM OrderCombinations
      {
         FOR_EACH ?c FROM Compartments
         {
            FOR_ONE ?u FROM UllagerateRules
            VERIFYING Compartments[?c].Rule = UllagerateRules[?u].RuleId
            {
               ( sum( FOR_EACH ?s FROM result
                      VERIFYING SplitOrders[?s].CompartmentId = Compartments[?c].CompartmentId
                      AND SplitOrders[?s].PickupTime  <= OrderCombinations[?O].PickupTime
                      AND SplitOrders[?s].DeliverTime >= OrderCombinations[?O].DeliverTime
                      { result[?s].AffAmount })
                 <= ( UllagerateRules[ ?u ].LowerBound * Compartments[?c].Capacity )
               )
               OR
               ( sum( FOR_EACH ?s FROM result
                      VERIFYING SplitOrders[?s].CompartmentId = Compartments[?c].CompartmentId
                      AND SplitOrders[?s].PickupTime  <= OrderCombinations[?O].PickupTime
                      AND SplitOrders[?s].DeliverTime >= OrderCombinations[?O].DeliverTime
                      { result[?s].AffAmount })
                 >= ( UllagerateRules[ ?u ].UpperBound * Compartments[?c].Capacity )
               );
            } ;
         } ;
      } ;
   } ;
   GROUP SHOW "6 - Axleweights "
   {
      FOR_EACH ?O FROM OrderCombinations
      {
         FOR_EACH ?A FROM Axles
         {
            sum( FOR_EACH ?C FROM AxleWeightContr
                 VERIFYING AxleWeightContr[?C].AxleId = Axles[?A].AxleId
                 { FOR_EACH ?r FROM result
                   VERIFYING result[?r].CompartmentId = AxleWeightContr[?C].CompartmentId
                   AND SplitOrders[?r].PickupTime  <= OrderCombinations[?O].PickupTime
                   AND SplitOrders[?r].DeliverTime >= OrderCombinations[?O].DeliverTime
                   { result[?r].AffAmount * AxleWeightContr[?C].Contribution }
                 })
            <= Axles[?A].MaxWeight ;
         };
      } ;
   } ;
} ;
```