Comparing Vehicle Routing approaches with Artificial Intelligence approaches

on solving instances of the
Logistics Planning Problem domain

Anne-Aimée Bun

Delft University of Technology

Comparing Vehicle Routing approaches with Artificial Intelligence approaches

on solving instances of the
Logistics Planning Problem domain

Master's Thesis in Computer Science

Algorithmics group
Department of Software Technology
Faculty of Electrical Engineering, Mathematics, and Computer Science
Delft University of Technology

Anne-Aimée Bun

1st July 2008

Author
Anne-Aimée Bun

Title
Comparing Vehicle Routing approaches with Artificial Intelligence approaches on solving instances of the Logistics Planning Problem domain

MSc presentation
23 July 2008
13h.30 Snijderzaal
Mekelweg 4, Delft

Graduation Committee

prof. dr. C. Witteveen	Delft University of Technology
ir. J.R. Steenhuisen	Delft University of Technology
em. prof. dr. ir. C. Roos	Delft University of Technology
dr. P.G. Kluit	Delft University of Technology

Abstract

Transportation problems are common, but complex. Therefore, it is not surprising that much effort is put into finding solving methods which generate high quality solutions in little time. Some of those solving methods are developed as planning systems and tested at the Artificial Intelligence Planning Systems (AIPS) or International Planning Competition (IPC). Others are developed as (rich) Vehicle Routing Problem (VRP) solvers and tested against some benchmarks.

It seems that developers of planning systems considering transportation problems and developers of VRP solvers are actually working on some similar, if not the same problems. Yet, it seems that cooperation between them is scarce. This can be explained by the different general goals they have. The Artificial Intelligence (AI) community focuses on creating general planners which can handle general, domain independent planning problems, while the main focus of the Operations Research community concerning transportation problems is solving a specific class of problems, which are based on Linear Programming, optimally. Nevertheless, the solutions sought for VRP's and some transportation planning problems are often solutions which are minimized in the number of movements, or the length of the movements, and in computation time. To show that some of those AI transportation planning problems and VRP variants are similar, we will show that a transportation problem used in the AIPS competitions can be written as a rich VRP.

We will compare the AI planners used in the mentioned competitions with a specific rich VRP solver and a Mixed Integer Program (MIP) solver. This will show that, considering the quality of the plans, the best results are generated by the rich VRP and the MIP solvers[1], followed by the AI planners FF and SGPlan. It will be shown that especially the rich VRP and the MIP solver are quite slow. Hence, for rapid results, the AI planners, especially YAHSP, are to be preferred, although the price is less plan quality. We will also show that when a decomposition method is applied to the AI planner YAHSP, the loss in plan quality will be considerably reduced, which causes it to be comparable to the plan quality with the other planners and the VRP solvers.

[1]Under assumption that the results of 50% of the tested problem instances, which are quite small, are nevertheless, representative.

“*Life is what happens to you while you’re busy making other plans.*”
– John Lennon(1940 - 1980) –
“Beautiful Boy”

Preface

After some courses on complexity theory, I became interested in NP-hard problems. Those that involved graphs, for example the Vehicle Routing Problem (VRP), are in my opinion, especially interesting. Therefore, I ended up doing my research assignment at the Algorithmics group at the faculty of Computer Science at the Delft University of Technology.

During this research project, I looked into logistic planning problems, and the so called Logistics Planning Problem domain in particular. I considered that this domain was almost similar to the VRP with Pick-up and Delivery (VRPPD), also known as the Pick-up Delivery Problem (PDP). Therefore, I was surprised to find that common solving algorithms for VRPDP were not used in the planners used to solve the problems in the Logistics Planning Problem domain. I was even more puzzled why VRP solving algorithms were not used when I found out that the largest benchmark of the Logistics Planning Problem domain had only about one tenth of the customers than the largest benchmark for the PDP, even with Time Windows (TW). On top of that, the PDPTW problems seem to be harder because of some extra features like TW, capacity limits for the used trucks, and more realistic distances, which are not present in the Logistics Planning Problem domain. Instead, this domain uses no TW at all, the trucks have a infinite capacity and all distances are set to one, so actually the number of movements is counted instead of measuring the distance.

Because of these observations, I decided to spend my master's thesis on comparing the performance of planners using AI algorithms with solvers using algorithms from the Operations Research community on the Logistics Planning Problem.

Acknowledgements

Special thanks to Stefan Røpke, who was so kind to provide me with a version of his PDPTW-solving program: ALNS-PDPTW. I also would like to thank Renze Steenhuisen and Hans Melissen for their assistance.

Anne-Aimée Bun

28th June 2008

Contents

Chapter 1

Introduction

Logistic problems are common. Every day, people are dealing with many logistic problems while sometimes not even noticing that it is a problem. For example, someone is going to work but finds a roadblock on his way. The logistic problem that needs to be solved now is: What is, in this situation, the best route to go to work? Another example of a logistic problem is when people do groceries: Before the letter can be posted, I have to buy a stamp and to spare my back, the heavy groceries will be picked up last, what will be the most efficient route? From the last example, it can be seen that some logistic problems are not only concerned with finding the fastest or shortest route, but also have to keep in mind certain extra demands.

One of the first to study logistics related problems was Hamilton (1805 - 1865) [30], after whom the Hamiltonian Path problem was named. Finding a Hamiltonian Path while minimizing the travelling distance is a well known logistic problem. It is called the Travelling Salesman Problem (TSP) which was first studied in the 1930's: The TSP is the problem of finding the cheapest way of visiting all of the cities and returning to your starting point, given a collection of cities and the cost of travel between each pair of them [12]. The more complex variant of this problem is the *Vehicle Routing Problem* (VRP), which basically introduces more vehicles (or salesmen) to the problem. The original VRP was formally introduced in 1959 by Dantzig and Ramser [15], and was later extended in many ways, of which examples can be found in Section A.2.

Especially larger logistic problems, for example problems like the 1000 customers benchmarks of Li and Lim [33], are difficult to solve optimally. In both the Planning Research and Operations Research communities, these problems have drawn much attention. Both fields of research have different overall goals in mind. Therefore, they use different approaches to solve logistic problems. Still, we are convinced that both fields of research are trying to solve the same problems. Therefore, we have modelled a logistic planning problem as a VRP

variant such that we are able to compare the solving methods of both fields of research. The subjects of our research is the Logistics Planning Problem from the Planning Research community and the Vehicle Routing Problem from the Operations Research community.

1.1 The Logistics Planning Problem

In 1998, the first *Artificial Intelligence Planning Systems* (AIPS) competition was organised [48]. During this competition several problems were presented, among which the Logistics Planning Problem. We will start with a description of the problem. A formal definition of the Logistics Planning Problem will be given in Section 3.4.1.

In a Logistics Planning Problem instance, there are several cities, each containing several locations, one of which is an airport. There are trucks, which can drive within a single city. The map of each city is a complete graph. There are airplanes, which can fly between airports. The cities are connected to each other by a complete graph of airways. It is assumed that all distances between locations are of equal length. The goal is to get some packages from various locations to various new locations. Unfortunately, this is an ambiguous description, which the hosts of the AIPS competition acknowledged [48]. Minimising either the number of parallel actions, or the total number of actions needed to solve the problem instances, should be an additional requirement. We have chosen for the latter requirement.

In the description of the Logistics Planning Problem, there are no fuel and capacity constraints (i.e., they are not considered in the problem). Those are assumed to be enough. As can be seen, there are trucks and airplanes. Of course, the airplanes cannot land at every location in a city, only at airports. Because the cities are only connected with each other by means of an airway, the trucks cannot go to every location too. Therefore, we can say that there are multiple maps: A different map for each city and the one for the planes.

The Logistics Planning Problem domain seems to be a bit artificial. Indeed, we can travel between cities without using an airplane. However, situations that fit the Logistics Planning Problem domain do occur in real life. Consider for example the postal service. When we send a letter to a friend, unless he lives nearby, we put the letter in a letterbox. The postal service will pick-up the letter, and will deliver it to your friend's letterbox. He will pick it up from there. If we consider the postal service as the airplane network, the postmen as airplanes, me and my friend as trucks and the postboxes as airports, we have an instance of the Logistics Planning Problem domain.

1.2 The Vehicle Routing Problem

As said previously, the basic VRP has formally been introduced in 1959 by Dantzig and Ramser [15]. Although the basic VRP is not the problem which resembles the Logistics Planning Problem the most, we will introduce it here, because all variants are based on it. The problem that has the most resemblance with the Logistics Planning Problem, will be discussed in Section 3.3. A brief description of other variants can be found in Appendix A.

In the literature, the VRP is also known as Vehicle Scheduling [31], Truck Dispatching [51], or the Delivery Problem [7]. The basic VRP contains several nodes, of which one node is special. This node is the depot, the others are customers. All customers have a known demand. There are several trucks and each truck has a fixed capacity. In the special case when only one truck exists, the VRP becomes a TSP. There is some kind of cost-function: it costs c_{ij} to travel from customer i to customer j. The basic VRP is to route all trucks along all customers, such that:

- every truck starts and finishes at the depot (it is not allowed to visit the depot in between),
- all customers are served according to their demand, and
- the total costs are minimised.

The basic VRP can be extended in multiple ways. Some do not essentially modify the problem, like adding travelling time or time windows. Other extensions do modify the problem, like having multiple depots, multiple goods, or different optimization goals [44, 54].

1.3 The Comparison of the Vehicle Routing Problem and the Logistics Planning Problem.

In order to make a fair comparison between the previously described problems, we have to show that it is indeed possible to make a polynomial reduction from the Logistics Planning Problem to a suitable variant of the VRP. When we started our project, we had the assumption that a VRP variant similar to the Logistics Planning Problem would exist. This assumption was based on the large amount of existing VRP variants, a few of those are presented in Appendix A.2. Unfortunately, after extensive search, this does not seem to be the case. Therefore, by modelling the Logistics Planning Problem as a VRP variant, we have created our own. Since VRP problems are traditionally described as a *Linear Program*, we have done the same. This description can be found in Section 3.4.1.

1.4 Versions of the Problem Instances

To compare the different approaches of the AI community and the OR community for solving the Logistics Planning Problem, our test set consists of the problem instances of this problem used in the AIPS competition of 1998 [48]. We have made some different versions of these problem instances. These versions are, besides the original problem instances, problem instances on which some preprocessing has been applied, and larger problem instances. The preprocessing effects are decomposition, filtering, and a combination of the previous two. We are interested in whether these preprocessing effects and enlarging the problem instances would make a difference in the performance of both the AI planners and the OR solvers. The performance will be measured in computation time and plan quality. The plan quality is based on the number of movements the vehicles have to do in order to have a solution to the problem instances. The plan quality decreases as the difference between the optimal solution and the found solution, measured in the number of movements, increases.

1.4.1 Decomposition of the Problem Instances

When a problem is difficult, it is not uncommon to search for ways to decompose the problem into smaller problems which are often easier to solve. For the Logistics Planning Problem, such a decomposition method exists. It is the depth-partitioning algorithm by Steenhuisen [66], which results in the Arbiter-0 method of Valk [72]. Since we are able to model the Logistics Planning Problem as a VRP variant, the same decomposition technique can be used for the VRP variant. It has already been shown by ter Mors [67], that a speed-up occurs, measured in computation time, when using this decomposition method to solve the problem instances of the Logistics Planning Problem of 2000. We are interested in whether the VRP solvers show the same effect.

The plan quality will also be measured. We expect a small change in plan quality due to the overhead introduced by the decomposition method. Large changes in plan quality would indicate that errors have been made, since a sudden rise in plan quality would indicate that infeasible plans were generated, and a sudden drop in plan quality would indicate that the number of movements was not minimized.

Of course, it is possible to use the decomposition method and subsequently remove the superfluous data. We assume that the results, measured in both computation time and plan quality, will resemble the combined results of removing the excess data and the decomposition of the problem instances.

1.4.2 Superfluous Data in the Problem Instances

Looking into the original problem instances of the Logistics Planning Problem used in the AIPS competition of 1998 [48], it occurred to us that the problem instances contained many superfluous data. For example, there were many packages in the problem instances, but only a few of them needed to be transported to another location. Hence, we will investigate what the effect of filtering out the superfluous data will be to the computation time and on the plan quality.

1.4.3 Enlarging the Problem Instances

The original Logistics Planning Problem instances were quite small, up to 57 packages, of which 44 were left after removing the unused packages. The largest of these Logistics Planning Problem instances is the only one which is comparable in size with the smallest benchmarks of the Solomon's test set, which are 100 customers large [26]. In these benchmarks, all customers have a certain demand, hence all customers have to be visited. Since the Solomon test set was designed for the Vehicle Routing Problem with Time Windows, only one customer is visited per demand. In the Logistics Planning Problem, at least two customers are visited per demand (package), the customer where the package is located and the customer who will receive the package. A maximum of two other customers can be added, because they are the customers the packages are passing. Even an additional four customers can be defended when we look at the number of orders. An order from the Logistics Planning Problem can be divided into at most three sub-orders, with two corresponding customers each. This would result in a maximum of six customers per order. It can be discussed whether any such intermediate customer is a real customer. An argument in favour is that it is a customer that has to be visited. An argument against it is that it is just an intermediate point. This intermediate point has to be on the correct route since, in this specific problem definition, it is the only one which connects the different subsets of locations. When multiple points could serve as transshipment-points, it would be unknown through which one the best route would be. This last argument will also make it difficult to divide the orders of the Logistics Planning Problem into sub-orders, since the intermediate points have to be known to do so. All in all, when we would like to compare the Logistics Planning Problem instances with Solomon's benchmarks, we have to multiply the number of packages by at least two, since one package represents at least one pick-up and one delivery demand.

It is more natural to compare the Logistics Planning Problem instances with the Pick-up and Delivery Problem with Time Windows, created by Li and Lim [33], because both problems have pick-up and delivery customers. The question whether the intermediate customers in the Logistics Planning Problem should be counted as customers still remains. The largest of the Pick-up and Delivery Problem with Time Windows have 1000 customers. To see how the

performance of the different AI and OR approaches would be affected, in time and plan quality, when solutions to larger problem instances need to be found, we created our own problem instances.

1.5 Research Questions

From the previous sections, several research questions can be deduced. Our main research questions are:

Can a VRP approach for solving the Logistics Planning Problem be competitive with an AI planning approach with respect to the computation time?
and
Can a VRP approach for solving the Logistics Planning Problem be competitive with an AI planning approach with respect to the plan quality?

In order to answer these questions, a reduction has to be made from the Logistics Planning Problem to a VRP variant. Since we could not find a suitable VRP variant to which we could make a reduction from the Logistics Planning Problem, the question becomes:

Is it possible to model the Logistics Planning Problem as a VRP variant, using Linear Programming?

As mentioned in Section 1.4, we are interested in the change of performance due to some pre-processing steps applied to the problem instances. The questions which address this subject are:

- What is the effect of decomposing the problem instances? Is it the same for the AI and OR approaches?
- What is the influence of filtering the problem instances? Is it the same for the AI and OR approaches?
- What is the combined effect of decomposing and filtering the problem instances? Is it the same for the AI and OR approaches?
- What is the effect of increasing the size of the problem instances? Is it the same for the AI and OR approaches?

The effects will be measured in computation time and plan quality. In this report, we will try to answer these questions.

1.6 Outline

We will first provide some background information on planning in Chapter 2. Then, in Chapter 3, we will go over some background on VRP. Because we use

a decomposition technique, which is also called a coordination method, we will give some general information on that in Chapter 4. The experiments will be discussed in Chapter 5, which will be followed by the conclusions in Chapter 6.

Chapter 2

Planning Background

The Logistics Planning Problem is a problem originating from the Planning Research community. In this chapter, we will give a definition of the term planning. We will also introduce the framework in which planning problems are formulated. We will end with some techniques used for solving many planning problems. However, we will start with some basic notions.

2.1 State Space Search

Let us start with some terminology. A *state* is a description of the condition the problem or world is currently in. A state can change when some input is given to which a *transition function* can be applied. The symbols that are allowed as input are called the *alphabet*. The *state space* is the set of states which can be reached by means of the transition functions [65].

It is possible to create a graph using the set of states and the transition functions. The nodes in the graph represent the states and when a transition function exist that transforms one state into another, a connection can be made between the two nodes. However, this is not always feasible since the state space is often much too large to generate and store in memory. *State space search* involves finding a path from the initial state to a goal state of a search problem, while keeping in memory as few states from the search space as possible. A search problem can be, amongst others, a planning problem. This is further elucidated in the next section. The general idea of finding a path by means of state space search is to build a search graph, starting from the initial state, the goal state or both. A state is expanded by applying all elements of the alphabet to that state, generating all of its successor states. These successors are in the next level down of the search graph. The order in which the states for expansion are chosen, is determined by the *search strategy*, for example depth-first search, breadth-first search or A^*-search. We will present the latter in Section 2.4.1. It is not unusual that different strategies result in very different behaviour [10].

The main difference of state space search versus traditional computer science search methods is that in state space search, the state space is kept implicit. Only the nodes which are explored, are generated. A solution to a search problem instance may consist of the goal state itself, or of a path from the initial state to the goal state [60].

2.2 Planning Definition

What is meant by Planning? Planning is making a plan, which is a sequence of actions that transforms the initial situation of a given planning task into a situation where the goal requirements are met [36]. Therefore, we can say that planning involves reasoning about future actions to sequence and generate a reasonable series of actions to be taken in order to achieve a goal [28]. It can be thought of as determining and sequencing all the small tasks that must be carried out in order to accomplish a bigger goal [17].

It is possible to plan automatically, by means of a planning algorithm. An abstract formulation of a planning problem is described in some formal language and consists of three parts: a description of *(i)* the world, *(ii)* the goal, and *(iii)* the possible actions that can be performed. The planners output is a plan, i.e., a sequence of actions that will achieve the goal, when executed in any world satisfying the initial state description. Note that these formulations are similar to the description of state space search in the previous section. By this abstract formulation, a class of planning problems can be defined, parametrised by the languages used to represent the world, goals, and actions [74].

A planning problem from the previous mentioned class of planning problems, can be denoted as $P(\Lambda, D, I, G)$, which is a 4-tuple, where Λ is the set of operators, D is a finite set of objects, I is the initial state, and G is the goal state. An operator definition may contain both variables and parameters, since an operator does not correspond to a single executable action but to a family of actions, which is different for each instantiation of the variables of the problem. An instance of an operator is often a set of preconditions and effects.

As said, the solution to a planning problem is a plan. This can be represented as a tuple $\langle S, O, L, B \rangle$ in which S is a set of plan steps which are instances of operators. As an example, if `load` is an operator, then `load package1 truck2` is an instance of the operator, which can be an element of S. O is a set of ordering constraints on the elements of S. Ordering constraints mean that a step S_i must occur sometime, not necessarily immediately, before step S_j. An ordering constraint is denoted by $S_i < S_j$ and usually induce only a partial ordering on the steps in S. Corresponding to some ordering constraints are causal links. A

causal link between steps exists when the effect of one step is a precondition of the other. In other words, causal links are triples $\langle S_i, c, S_j \rangle$, where S_i and S_j are elements of S and c is an effect of S_i and also a precondition for S_j[1]. L is a set of causal links representing the causal structure of the plan. It is important to keep track of a causal link $\langle S_i, c, S_j \rangle$ to ensure that no step S_k which would result in $\neg c$, can intervene between steps S_i and S_j. B is a set of binding constraints on the variables of the operator instances in S [49].

2.3 A Plan Formalism: STRIPS

In the previous section, we mentioned that the input of a planning problem is abstractly formulated in some formal language. *STRIPS* (Stanford Research Institute Problem Solver) was originally an automated planner developed by Richard Fikes and Nils Nilsson in 1971 [22]. The formal language used for the input of this planner is the base for most of the languages for expressing automated planning problem instances used today, and goes by the same name.

STRIPS represents the world as a set or first-order logic formulas [60]. Therefore, it does not make a distinction between more important and less important parts of the problem description. The problem space for STRIPS is defined by three entities [22]: *(i)* the present state of the world, *(ii)* a set of operators, their their effects and their preconditions, and *(iii)* a goal condition, each one described in well-formed formulas. Any conditions that are not mentioned in a state are assumed false, i.e., the *closed-world* assumption is used.

Mathematically, a STRIPS instance is a quadruple $\langle P, O, I, G \rangle$, in which P is a set of propositional variables and O is a set of actions [71]. Each action is defined by four sets of propositional variables: $\langle \alpha, \beta, \gamma, \delta \rangle$, which represent the pre- and postconditions of the action. The preconditions are represented as α and β, which are the sets of propositional variables that are required to be respectively true and false for the action to be executable. The postconditions are represented as γ and δ, which are the sets of propositional variables that are made respectively true and false by the action. I is the initial state, which is the set of propositional variables that are initially true. The goal state G is a pair of sets $\langle N, M \rangle$, which specify which propositional variables are respectively true and false in the goal state.

Changing a state into another can be done by means of a transition function, just like with state space search. As already mentioned, in STRIPS, states are represented by sets of propositional formulas. A state in STRIPS is a certain configuration of the set P, in fact, one of 2^P configurations, which represent all

[1] More generally, c represents a proposition that is the unification of an effect of S_i and a precondition of S_j.

possible states. Therefore, the transition function of a STRIPS instance can be written as the function: *succ* : $2^P \times O \rightarrow 2^P$. A transition function for a certain state C can therefore be defined as follows:

$$\begin{aligned} \textit{succ}(C, \langle \alpha, \beta, \gamma, \delta \rangle) &= C \backslash \delta \cup \gamma \text{ if } \alpha \subseteq C \text{ and } \beta \cap C = \emptyset && (2.1) \\ &= C \text{ otherwise.} && (2.2) \end{aligned}$$

A plan for a STRIPS instance is a sequence of actions. In order to be able to have plan with more than one action, it is needed that the function *succ* can be used recursively:

$$\textit{succ}(C, [\,]) = C \tag{2.3}$$

$$\textit{succ}(C, [a_1, a_2, \ldots, a_n]) = \textit{succ}(\textit{succ}(C, a_1), [a_2, \ldots, a_n]) \tag{2.4}$$

If executing all actions of the plan in order from the initial state results in the goal state, it is a valid plan. Formally, if the state $F = \textit{succ}(I, [a_1, a_2, \ldots, a_n])$ satisfies the conditions $N \subseteq F$ and $M \cap F = \emptyset$, then $[a_1, a_2, \ldots, a_n]$ is a valid plan for the goal $G = \langle N, M \rangle$ [71].

We would like to make two comments on the STRIPS language as presented. In practice, it can occur that actions have free variables. These free variables will be implicitly existentially quantified, which means that such an action represents all possible propositional actions that can be obtained by replacing each free variable with a value.

The second comment is on the assumption that the initial state is considered fully known in advance because of the closed-world assumption. The closed-world assumption is often a limiting assumption, as there are natural examples of planning problems which have partially known initial states. Variants of STRIPS have been developed to deal with those planning problems [71].

2.4 Used Techniques in Planning

The programs used for solving planning problems are often called planners. Most planners are specialized in one of the previously discussed planning problem categories. The way they try to solve them can differ. Some use satisfiability solvers and some use *heuristic algorithms* like weighted A* and A*-search algorithms. Tables or diagrams are often used, especially in combination with backward search [53]. In this section, we will go over these methods briefly.

2.4.1 GraphPlan

When GraphPlan was introduced in 1995, by Blüm and Furst [6], the time needed to compute solutions to planning problem instances, drastically decreased. The

algorithm this planner used, goes by the same name, of which the basic idea is presented in this section. The input of GraphPlan is a propositional planning problems, i.e., planning problems with no variables, for example in STRIPS format. It efficiently constructs a compact structure called a Planning Graph from the given problem statement [5].

A planning graph consists of a sequence of levels that correspond to time steps in the plan, where level 0 is the initial state. Each level contains a set of literals and a set of actions. It could be said that per level, depending on the actions executed at preceding time steps, the literals are those that could be true at that time step. Analogous, it could be said that per level, depending on which of the literals are actually true, the actions are all those actions that could have their preconditions satisfied at that time step. Because the planning graph records only some of the possible negative interactions among actions, it might be optimistic about the minimum number of time steps required for a literal to become true. Nonetheless, this number of steps in the planning graph provides a good estimate of the difficulty to reach a certain goal from the initial state [60].

In GraphPlan's planning graph, there are three types of edges between the nodes: *(i)* pre-conditional, which connects a literal and the action for which the literal is a precondition, *(ii)* post-conditional, which connects a literal and the action of which the literal is a result, and *(iii)* mutex, incompatible literals that cannot be true at the same time and incompatible actions that cannot be executed together are connected with mutex-links. By means of those mutex-links, the impossibility of certain choices become explicit.

The GraphPlan algorithm uses the planning graph in the following way. The algorithm has two alternating main steps. First, it checks whether all the goal literals are present in the current level without mutex-links between any pair of them. If such a link exists, the graph is expanded by adding the actions for the current level and the state literals for the next level. If no mutex-links exist in the current level, the algorithm tries to extract a solution, since it is possible that a solution exists within the current graph. The process is repeated until a solution is found or until it has found that no solution can exist [60].

Many planners are based on the GraphPlan algorithm, like the planner FF (Fast-Forward) [53], although often in combination with a heuristic search algorithm. An example of an heuristic search algorithm is A^*.

A^*-search

As explained in Section 2.1, planning problems can be written as a graph. It is possible to generate a state diagram in which the initial state is the first node and the goal state is (one of) the last, or vice versa. When a plan is build up

from the initial situation, to the goal situation, *forward* or *progression planning* is used. Planners that use forward planning are FF and YAHSP [53]. The major disadvantage of forward planning is the large search space, because with each step the number of possibilities grows, and all of them are considered equally likely to lead to a solution. To reduce the search space, GraphPlan is used first. When the starting point is the goal situation, and a plan is made while searching for the initial situation, *backward* or *regression planning* is used. This approach is often possible, because enough information is known to deduct from a partial description of a result state, a partial description of the state before an operator is applied. The advantage in comparison of the forward approach is that the search space is expanding less rapid.

Both forward and backward planning search for the optimal solution. The optimal solution is the shortest path from the initial state to the goal state and can be found with a graph traversal algorithm. The A^* algorithm, introduced by Hart, Nilsson and Raphael in 1968 [32], is an example of a graph traversal algorithm.

In the planner YAHSP (Yet Another Heuristic Search Planner) a variant of the A^* algorithm is used [73]: the weighted A^*. Since this is a derivative from the A^* algorithm, we will present only the A^* algorithm.

The A^* algorithm tries to find the shortest path from start s to finish t. The nodes s and t are part of a graph $\mathcal{G} = \{\mathcal{V}, \mathcal{E}\}$, in which $\mathcal{V}$ is the set of nodes and $\mathcal{E}$ is the set of edges. It makes use of a heuristic estimation $h(i)$ of the distance from node i to node t. Such an estimation needs to be admissible, i.e., it is never larger than the actual distance, in order for the A^* algorithm to be optimal when not used in a closed set. If used in a closed set, the triangle inequality should even hold for h. The actual distance between node i and j is represented as c_{ij}. Now, the A^* algorithm is as follows [54]:

Algorithm 1: The A^* algorithm [54].

Step 1: *Initialise*
Set $\mathcal{Q} = \{s\}$, set $\forall i \in \mathcal{V} \backslash \{s\} : \ell(i) = \infty$, $\ell(s) = 0$.

Step 2: *Select Node*
Let $i \in \mathcal{Q}$ be the node with $i = \arg\min\{\ell(j) + h(j) : j \in \mathcal{Q}\}$. Remove node i from $\mathcal{Q}$. The label of node i is now fixed. Quit the algorithm if $i = t$.

Step 3: *Adjust Labels*
Do $\forall (i, j) \in \mathcal{E}(i) : \ell(j) = \min\{\ell(j), \ell(i) + c_{ij}\}$. If $\ell(j)$ is modified in this step, then $\pi(j) = i$ and add j to $\mathcal{Q}$ unless j is already in $\mathcal{Q}$. If the label of j was fixed, then it will become unfixed because node j was added to $\mathcal{Q}$ again.

Step 4: Go to Step 2.

In this algorithm, $\mathcal{Q}$ stands for the set of candidate nodes which all have a label less than infinity. The label $\ell(i)$ stands for the temporal shortest path from source s to node i. The value of $\ell(i) + h(i)$ represents the temporal estimation of the shortest path from s to t via node i. For weighted A^*, the $h(i)$ in this equation is multiplied with a certain factor x, the weight, which should be larger or equal to one. The set $\mathcal{E}(i)$ is a subset of $\mathcal{E}$, an edge (u, v) is in subset $E(i)$ if $u = i$, hence they are the outward bound edges of node i. The operator $\pi(j)$ represents the predecessor of j. It is not sufficient to know only the length of the path to reconstruct the shortest path. It is also needed to know at least the predecessor of the node on the shortest path. The shortest path can be reconstructed by recursively calling $\pi(t)$ (i.e., $\pi\pi(t), \pi\pi\pi(t), enz$).

To summarise the algorithm: The A^* algorithm basically searches along nodes which are expected to be on the shortest path from s to t [54].

2.4.2 Satisfiability

Some planners, for example BlackBox[2] [53], transform a planning problem into boolean formulas, which is not hard to do since planning problems written in STRIPS are already composed of first-order logic formulas. A boolean formula consists of boolean variables, which can only assume the values “true” or “false”, and the logical operators “and”, “or”, and “not”. Often, the value “true” is represented as a 1 and the value “false” as a 0. A boolean formula is satisfiable if some assignment of 0’s and 1’s to the variables makes the formula evaluate to 1 [65]. Now, the Satisfiability problem (SAT) boils down to: Find a truth assignment to the variables of the formula that makes the formula evaluate to the logical value “true” [50].

When transforming a planning problem into boolean formulas, the literals are accompanied by the number of the time step, 0 representing the initial state. Because propositional logic has, in contrary, to STRIPS no closed-world assumption, the propositions that are not true need also to be specified. Only when propositions are unknown, they can be left unspecified. In the same manner, actions can be transformed into boolean formulas. The actions are also accompanied by a time step number, representing the time when the result of the action is visible. In addition to the literals and the actions, the preconditions need to be transformed as well as state constraints which prevent simultaneous actions only when they interfere with each other, which is detected as conflicting variables. The goal situation should also be associated with a time step. Since it is unknown when the goal can be reached, the time step of the goal is iteratively increased as long as no truth assignment can be found for which the boolean formulas of the planning problem evaluate to “true” [60]. By using these iterations, the search

[2]This planner was not used during the experiments because it was too slow to be competitive.

space is kept as small as possible.

2.5 Artificial Intelligence Planners used for our Experiments

In the section above, we already mentioned some planners which we used during our experiments. The choice of these planners was based on availability and on the diversity of the implemented algorithms. In this section we will highlight their specific features.

2.5.1 FF

FF was designed by Jörg Hoffmann [53]. It uses GraphPlan for graph building and plan extraction. The length of the plans that GraphPlan returns are used as an indication of the distance to the goal, and the actions that GraphPlan selects are used to guide the actual searching algorithm. The searching algorithm that is used is a combination of hill-climbing and systematic search. It was assumed that with respect to these algorithms, big plateaus or local minima do not occur very often, at least not in the benchmark planning problems. Indeed, plateaus caused by so called symmetries in planning domains, are handled well by FF [25].

2.5.2 YAHSP

YAHSP was designed by Vincent Vidal [73]. It is based on FF, but uses a lookahead strategy. In classical searching strategies, all actions that can be performed when in a certain state, are considered the same. The lookahead strategy used in YAHSP orders and evaluates the actions on a certain measure of so called helpfullness. The actions which are considered helpfull are the executable actions of a relaxed plan in a state. This lookahead strategie improves the computation time considerably [11].

2.5.3 SGPlan

SGPLan was designed by Chih-Wei Hsu, Benjamin Wah, Ruoyun Huang and Yixin Chen [38]. It partitions large planning problems into subproblems with their own sub-goals. It tries to resolve inconsistent solutions. By partitioning the large problems, some extra constraints are deduced. The searching algorithm that is used tends to get stuck in infeasible regions when the objective is too small or when the penalty values and/or constraint violations are too large. It gets resolved by backtracking. Like YAHSP, it uses helpfull actions to reduce the evaluated search space.

2.6 Summary

When defined in a formal language, like STRIPS, planning problems can be represented as state space search graphs. However, this representation is often too large to use in practice. Therefore, algorithms, like GraphPlan, are developed. GraphPlan compiles planning problems described in STRIPS into a different representation, which causes the search space to stay relatively small.

Chapter 3

Vehicle Routing Problem Background

Unlike in the previous chapter, there is not much of a discussion on the definition of a vehicle routing problem. This is mainly due to the fact that vehicle routing is an example of a planning problem. On the other hand, once in a while, the question "What makes a problem a VRP" is raised. After extensive search, we could not find a conclusive definition. We found many different problems, which were variants of the basic VRP with extra constraints or features, which were all called a VRP. The difference with de basic VRP was indicated in the name of the particular problem, like VRP with Time Windows (VRPTW). It seems that any problem for which the goal is to find the optimal path for every vehicle, under some constraints, such that the demand of all customers are served, can be called a VRP.

3.1 VRP Framework

We have already shown that the AI-community uses STRIPS to describe planning problems. In the OR-community, mathematical models are used, which exist of sets of equations. A special kind of equation is the linear equation. A linear equation is an equation in the variables $x_1, \ldots, x_n$ that can be written in the form $a_1x_1 + a_2x_2 + \ldots + a_nx_n = b$ where b and the coefficients $a_1, a_2, \ldots, a_n$ are real or complex numbers [45].

A *Linear Program* (LP) is most frequently used for mathematical models. If we can specify an objective, which has to be minimized or maximized given limited resources and competing constraints, as a linear function of certain variables, and if we can specify the constraints on resources as (in)equalities on those variables, then we have a LP problem [14]. In other words: LP problems involve the optimization of a linear objective function, subject to linear (in)equality constraints. If the unknown variables are all required to be integers, then the problem is called an Integer Program (IP) or Integer Linear Program (ILP) [14].

3.2 Used Techniques

LP's be solved using several techniques. For example the simplex algorithm, developed by George Dantzig in 1947 [15], the ellipsoid method, introduced by Leonid Khachiyan in 1979 [1], and interior point methods like Karmarkar's algorithm [42, 61]. VRP's are often solved with search or graph traversal algorithms, like branch and bound, branch and cut, A^* and (large) neighbourhood search [54]. In Section 2.4, we have already introduced the A^* algorithm. In this section, we will focus on two more of these algorithms, since they are used in the VRP solvers we will be used in our tests.

3.2.1 Simplex Algorithm

In this section, we will present the basic idea of the Simplex algorithm, not an elaborate description. For the latter, we refer to Cormen [14].

First, it is needed to have at least an intuition of the term simplex. A simplex is a convex[1] region, constructed with the constraints of a LP, in which all possible solutions of that LP are contained. Because the simplex is convex, the solution of an optimal solution to a problem will always occur at the rim of the simplex.

The input of the Simplex algorithm is a LP, and it returns an optimal solution. It starts at a certain vertex of the simplex, which is constructed by means of the LP, and performs a sequence of iterations. In each iteration, the algorithm tries to find an objective value at a neighbouring vertex, larger than or equal to the objective value at the current vertex. Neighbouring vertices are those vertices which are along an edge of the simplex next to the current vertex. When a local maximum is reached, the Simplex algorithm terminates. The local maximum is found at a vertex for which holds that it is a vertex from which all neighbouring vertices have a smaller objective value. Because the simplex is a convex region and the objective function is linear, this local optimum is actually a global optimum and therefore the optimal solution to the LP [14].

The major drawback of the Simplex algorithm is that in the worst case scenario, its running time is exponential. This has been shown using carefully constructed LP's, which do not occur often in real life. Usually, in practice, the Simplex algorithm is remarkably fast.

[1] An intuitive definition of a convex region is that if a line is drawn between any two points in the region, the line never crosses the rim of the region. As an example, in three dimensions: an hourglass is not convex, a cylinder is.

3.2.2 Large Neighbourhood Search

Large Neighbourhood Search (LNS) for VRP's was first presented by Shaw in 1998 [62]. The heuristic is quite like local search. In local search it is assumed that a (sub-)optimal route is found from the start node to and goal node. In planning the start node could represent the initial state, and the goal node could represent the goal state. Local search breaks a few branches of the current solution (route), resulting in sub-routes. Those sub-routes will be reattached to each other in a different order, if that order results in a better overall solution. Analogous, LNS removes several related customers from the current solution and reinserts them in a different order, if the overall solution improves.

Two choices have much influence on the performance of the heuristic. The first is the choice of the set of customers that need to be removed. The second is the algorithm used for reinserting the removed set of customers.

The choice of the set of customers that will be removed is generally based on how related the customers are. When and how much customers are related has to be properly defined and it should be chosen such that it results in good opportunities for the reinsertion to achieve improvement in the route. One obvious criterion would be that customers that are geographically close to one another will be more related than customers that are more distanced [62]. However, different relations can be thought of [57]. The size of the set which is to be removed is important too. Of course, a fixed number can be picked. Shaw decided to increase, starting from 1, the number of customers to be removed when a number of attempts did not improve the solution.

Shaw used a truncated branch and bound technique as an insertion algorithm, which is a near optimal insertion algorithm. Other insertion algorithms might also be used.

The pseudo-code for a minimizing LNS heuristic is shown in Algorithm 2 [57]. Like with local search, LNS assumes that an initial, (sub-)optimal solution s has been found, for example by a simple construction heuristic. The parameter q determines the how many customers will be removed from the current solution, which can be seen in line 5. In line 6, the removed customers are reinserted into the current solution. These two actions are responsible for the possible improvement of the current solution.

In lines 7 to 11, the best solution so far is updated and it is determined if the new solution should be accepted. A simple accept criterion would be to accept all improving solutions, but again other criteria can also be used. When some stop criterion is met, see line 11, the algorithm is terminated. Such a criterion can be, for example, when after several iterations no or insufficient improvements has been made.

Algorithm 2: LNS heuristic [57]

```
1  Function LNS(s ∈ {solutions}, q ∈ ℕ)
2  solution s_best = s
3  repeat
4      s' = s
5      remove q customers from s'
6      reinsert removed request into s'
7      if f(s') < f(s_best) then
8          s_best = s'
9      end
10     if accept (s', s) then
11         s = s'
12     end
13 until stop-criterion met ;
14 return s_best
```

3.3 VRP Variants

The basic VRP described in Section A.1 does not yet resemble the Logistics Planning Problem mentioned in Section 1.1. Fortunately, the VRP can be extended in various ways. All variants are gathered under the name of *rich VRP*. In this section, we will show the variant which has the most similarities with the Logistics Planning Problem. For a more extended overview of the different variants, we refer to Appendix A.2.

3.3.1 RPDPTW

The most elaborate version of a VRP can be found in the description of an Adaptive Large Neighbourhood Search heuristic for the Pick-up and Delivery Problem with Time Windows (ALNS-PDPTW) [57]. ALNS-PDPTW was designed by Røpke to handle more than one vehicle routing problem. The idea behind it is that in real life, transportation done by different companies is often different and thus calls for different types of vehicle routing problem solvers. The ALNS-PDPTW can solve five different vehicle routing problems. These are the Vehicle Routing Problem with Time Windows (VRPTW), the capacitated vehicle routing problem (CVRP), the multi-depot Vehicle routing problem (MDVRP), the site dependent vehicle routing problem (SDVRP) and the open vehicle routing problem (OVRP).

In the CVRP, goods have to be delivered to a set of customers with known demands on minimum-cost vehicle routes originating and terminating at a depot. The vehicles are assumed to be homogeneous and they have a certain capacity. Note that the CVRP is not that different from the basic VRP. Though the constraint of not exceeding the maximum capacity of the vehicles is not explicitly mentioned

in the basic VRP, it is often assumed to be the same as CVRP. In some versions of the CVRP, it is also required to obey route duration constraints that limits the lengths of the feasible routes. When time windows, the interval in which customers should be supplied, are added to the CVRP, the problem is called VRPTW. If it is not required to return a vehicle to a depot when the last customer of its route has been served, the route is open, hence OVRP. The MDVRP extends the CVRP by introducing multiple depots. The SDVRP generalises the CVRP, because it can be specified that certain customers only can be served by a subset of the vehicles. Furthermore, vehicles can have different capacities in the SDVRP. All problem types are transformed into a Rich Pick-up and Delivery problem with time windows (RPDPTW) [58].

The RPDPTW is the problem of serving a number of transportation requests, which involves moving some goods from a pick-up location to a delivery location, using a limited number of vehicles. The objective is to construct routes that visit all locations such that the pick-up and delivery of the same goods are placed on the same route and that the goods are picked up before they are delivered. Furthermore, the route of a vehicle should start at a depot, end at a given location, and service a number of requests such that the capacity of the vehicle is not exceeded in between. The routes must not violate time window and capacity constraints. A non-negative distance and travel time is assigned between any two locations. It is assumed that travel times satisfy the triangle inequality. This assumption implies that any removal of requests from a feasible route will keep the route feasible with respect to the imposed time windows [57, 58].

3.3.2 RPDPTW and the Logistics Planning Problem

RPDPTW is not usable as a model for the Logistics Planning Problem. In this section, we will present several reasons which result in this conclusion. However, let us start with a recollection of the Logistics Planning Problem.

The Logistics Planning Problem is the problem of serving a number of transportation requests using a limited number of vehicles which have infinite capacities. Each request involves moving a number of goods from a pick-up location to a delivery location using several different vehicles, hence, one request consists of a chain of pick-up and delivery sub-requests. The objective is to construct routes per vehicle such that every sub-request is performed in time with respect to the other sub-requests in the same chain. The pick-up and delivery of such a sub-request is placed on the same route and such that a pick-up is performed before the corresponding delivery. Furthermore, the route of a vehicle should start at a given location, service a number of (sub-)requests, and finally end at a certain location. Between any two locations, we have a uniform distance.

The first reason why RPRPTW is not usable as a model for the Logistics

Planning Problem concerns the time windows. Every task in an instance of the Logistics Planning Problem is actually a combination of three subtasks. Either the time windows are used for the original, or for the subtasks. In the first case, there is no way to guarantee that the subtasks are completed in the right order. In the second case, the sequence of the subtasks can be guaranteed to be correct. However, the time windows are not only valid for one sequence of subtasks, but for the whole system. By that we mean that time windows impose an order in which the tasks are served, while in the Logistics Planning Problem, the only concern is that the subtasks are done in the right order. Hence, it is far too restrictive to use time windows this way.

Looking at the Logistics Planning Problem, using multiple depots might not be such a bad idea. After all, the vehicles start at random locations within the problem, which can therefore, be seen as depots. Unfortunately, a vehicle has to return to a depot and can only return to a depot when finished, not between tasks. If we use multiple depots in the Logistics Planning Problem, it would imply that if one vehicle has to deliver a package to another location which happens to be also a depot, the vehicle will be unavailable for the rest of the problem. This is an undesirable effect. Of course, a work around can be thought of, which is an imaginary depot from which all vehicles start and where they all finish. The only drawback is that some post-processing is needed to find out how many movements are made extra, due to the introduction of this fake depot.

The last reason is of a more practical nature. The Site Dependent feature is actually exactly what we need to model the Logistics Planning Problem. Unfortunately, this feature is implemented such that a vehicle has to be able to perform the whole task, not just a subtask, see Appendix A.3. Of course we can divide every task in subtasks, but then again we cannot guarantee the order in which the subtasks are performed.

Clearly, the main problem we faced for creating our own model was a synchronisation problem. How to make sure that the transshipment of the packages is feasible? We will discuss our solution to this problem in the next section.

On the other hand, in case of only one city, no synchronization problem exists. When we apply the decomposition method we will present in Section 4.3.1, we can evade this synchronization problem. In that case, the RPDPTW can be used to model the Logistics Planning Problem.

3.4 Modelling the Logistics Planning Problem

Considering the whole Logistics Planning Problem, deciding whether a plan of a certain length exists, was proven to be NP-complete [34, 35]. Therefore, it should

be possible to make a reduction to any other NP-complete problem, though it might not be that straight forward.

We had the assumption that a VRP variant similar to the Logistics Planning Problem would exist. Unfortunately, after extensive search, this seems to be not the case. Several problems have some features that come close to the Logistics Planning Problem, like Site Dependency, which means that vehicles can be assigned to a sub-graph, but can only serve requests that start and finish in that sub-graph [57, 58, 59]. As described in the previous section, Site Dependency alone cannot be used to create a reduction, since there is no way to guarantee the precedence relations between the subtasks, which have to be specified explicitly.

In the literature concerning transportation with transshipment, like in [16], the focus is on determining the best location for the transshipment stations and as such, reaching optimal flow. That was not what we were looking for, hence, it appeared that we had to construct a model with transshipment locations which also respected precedence relations within a request ourselves.

3.4.1 The Mathematical Model for the Logistics Planning Problem

Since the problem described in the previous section did not match any known VRP variant, we designed our own LP model.

In our model, we have a several groups of nodes. We call these groups cities. There are n cities. Each city consists of $m \geq 1$ nodes, and it is not necessary that all cities consist of the same amount of nodes. Every node within our model can be uniquely defined by the combination of the number of the city and the number of the node within that city. Such a tuple: $(city\ number, node\ number)$, we call a location. So, the set of all locations is:

$$\mathcal{L} = \{(i,j) \mid \ j = 1, \ldots, m_{(i)},\ i = 1, \ldots, n\}. \tag{3.1}$$

The sub-set of all nodes present in a city is defined by:

$$\mathcal{L}_i = \{(i,j) \mid \ j = 1, \ldots, m_{(i)}\} \subseteq \mathcal{L}. \tag{3.2}$$

In the Logistics Planning Problem there are some special locations: the airports. The airports are the only locations that form a sub-graph with locations of another city. Every city has at least one node, which in that case, has to be the airport. Generally, let us assume that the first node in a city is an airport. The sub-set consisting of all airports will then be defined by:

$$\mathcal{L}_0 = \{(i,1) \mid \ i = 1, \ldots, n\} \subseteq \mathcal{L}. \tag{3.3}$$

Note that previously we used the subscript of $\mathcal{L}$ to show the city number. Since the sub-set of all airports is unlike one of a city and there is only one such a sub-set,

we use a 0 to show this.

We do not take real distances or travel-times into account. Instead, travelling from one location to another takes always the same amount of time or distance. Hence, it is justifiable to say there are time discretizations, and the set of those time steps is defined by

$$\mathcal{N} \subset \{N \in \mathbb{N} \mid N \leq 2(\max\{m_{(i)} \mid i \in \mathbb{N}\}) + 2n\}. \tag{3.4}$$

Due to this equation, we facilitate that actions can be performed in parallel.

The inequality of Equation 3.4 can be explained as follows. If we make a round along all nodes, which can be done simultaneously and therefore takes as long as to visit all nodes in the largest city ($m_{(i)}$), we collect all packages. This can be done in such a way that the airport is the last location that is visited. There are n airports, if we visit them all once, all packages are again collected. If we visit all airports a second time, all packages are in the city of their destination, but not yet at the right destination. Therefore we have to make another round along all nodes again. Because we can assume that the first location that will be visited in the latter round is the airport, only one round along all locations is necessary. Now all packages are at their goal-location.

Of course, visiting locations is done by T vehicles. Each vehicle is assigned to a sub-set by means of the city number. The city number is again 0 to denote the airport sub-set, which suggests that the vehicles travelling between airports are airplanes. All vehicles travelling between other locations, can be thought of as trucks. All vehicles have a start-location. Of course, this start-location must be part of the sub-set the vehicle is assigned to.

$$\mathcal{T} = \{t_\ell \mid \ \ell = 1, \ldots, T,\ t_\ell = (a, b, c, \ell), (a, b) \in \mathcal{L}_c, c \in \{0, \ldots, n\}\}. \tag{3.5}$$

The start-location in itself is not enough information. A truck with a city-sub-graph, can start at the airport of that city. If no separate city number would be known in this situation, there would be no way to discern between trucks and airplanes. We also add the vehicle number to this definition to be able to distinguish between vehicles. If we do not add this number, it is impossible to have two vehicles starting at the same location and being restricted to the same city. The serial number of the vehicle is not used in any calculations.

We have a set of packages, this set is P packages large. All packages are tuples of locations, the first location is where the package starts, the second is where it has to go to.

$$\mathcal{P} = \{p_o \mid \ o = 1, \ldots, P,\ p_o = (a, b, c, d, o), \{(a, b), (c, d)\} \in \mathcal{L}\}. \tag{3.6}$$

Like with the vehicles, we add a serial number to the definition of the packages. It is only to distinguish between packages and it is not used in any calculations.

We have a tracking-variable for vehicles $C_n^t \in \mathcal{L}_{t_3}, t \in \mathcal{T}, n \in \mathcal{N}$ and a tracking-variable for packages $D_n^p \in \mathcal{L}, p \in \mathcal{P}, n \in \mathcal{N}$. These tracking-variables are actually locations, they represent the location the package p or vehicle t is at, at a certain time-step n. To ensure that a vehicle cannot travel beyond the locations in the sub-graph it was assigned to, the tracking-variable for that vehicle is also bound to the same sub-graph. To accomplish that, the sub-graph for which the variable is valid, is extracted from the definition of the truck it belongs to. The third variable of the definition of a truck, t_3, represents that sub-graph, which explains the notation of the sub-graph in the definition of the tracking-variable C_n^t. If we want to know the route a vehicle or package follows, we just have to put the corresponding tracking-variables in increasing order of the time-slots. Hence, the route a vehicle follows is denoted as

$$\forall t \in \mathcal{T}, (C_1^t, C_2^t, \ldots, C_N^t) \tag{3.7}$$

and the route a package follows is denoted as

$$\forall p \in \mathcal{P}, (D_1^p, D_2^p, \ldots, D_N^p). \tag{3.8}$$

The start of the routes equals the start-location of the vehicles

$$\forall t \in \mathcal{T}, (C_1^t)_1 = t_1, (C_1^t)_2 = t_2, \tag{3.9}$$

or the packages

$$\forall p \in \mathcal{P}, (D_1^p)_1 = p_1, (D_1^p)_2 = p_2. \tag{3.10}$$

Of course, it is possible that, for example, a packages reaches its end-location before all time-steps are finished. In that case, the tracking-variable corresponding to that package simply registers the location the package is at, though the location will stop changing. For example, if a package has to go from location $(2, 2)$ to location $(2, 3)$, and the number of time-steps equals four, the route of that package could be: $((2, 2), (2, 3), (2, 3), (2, 3))$.

A package can only change its location if there is a vehicle that is making the same transition. Therefore, we first want to know whether a package and a vehicle are at the same location. To this purpose, we introduce a binary variable $S_{ijk} \in \{0, 1\}$. We would like to have $S_{ijk} = 1$ iff[2] $C_k^i = D_k^j, i \in \mathcal{P}, j \in \mathcal{T}, k \in \mathcal{N}$. In other words, the difference between the location of package i and the location of vehicle j is equal to 0 iff they are at the same location. Unfortunately, this is impossible to put this decision in a LP. However, the formula $(C_k^i - D_k^j)S_{ijk} = 0$ will do because of the maximization in the end. We will explain this later in more detail. In LP notation the latter formula will be:

$$M(S_{ijk} - 1) \leq (C_k^j)_1 - (D_k^i)_1 \leq M(1 - S_{ijk}), \tag{3.11}$$

[2]We use the word "iff" as a shorthand for "if and only if".

$$M(S_{ijk} - 1) \leq (C_k^j)_2 - (D_k^i)_2 \leq M(1 - S_{ijk}) \tag{3.12}$$

where M is a sufficiently large number.

As said, a package can only move if a vehicle makes the same move: $\forall i \in \mathcal{P}, k \in \mathcal{N}, D_k^i \neq D_{k+1}^i \exists j \; S_{ijk} = 1 \wedge S_{ij(k+1)} = 1$. To put this in our LP, we introduce an auxiliary binary variable $Z_{ijk} \in \{0, 1\}$, which stands for whether a package i at time k makes the same move as vehicle j: $Z_{ijk} = 1$ iff $S_{ijk} = 1 \wedge S_{ij(k+1)} = 1, i \in \mathcal{P}, j \in \mathcal{T}, k \in \mathcal{N}$. In our LP, this is accomplished by:

$$Z_{ijk} \geq S_{ijk} + S_{ij(k+1)} - 1, \tag{3.13}$$

$$Z_{ijk} \leq S_{ijk}, \tag{3.14}$$

$$Z_{ijk} \leq S_{ij(k+1)}, \tag{3.15}$$

$$-M \sum_j^T Z_{ijk} \leq (D_k^i)_1 - (D_{k+1}^i)_1 \leq M \sum_j^T Z_{ijk}, \tag{3.16}$$

$$-M \sum_j^T Z_{ijk} \leq (D_k^i)_2 - (D_{k+1}^i)_2 \leq M \sum_j^T Z_{ijk}. \tag{3.17}$$

The next binary variable we use indicates whether a package is at its end-location at a certain time: $E_{ij} = 1$ if $(D_j^i)_1 = i_3 \wedge (D_j^i)_2 = i_4, i = (i_1, i_2, i_3, i_4, i_5) \in \mathcal{P}, j \in \mathcal{N}$. In LP, this will be

$$M(E_{ij} - 1) \leq (D_j^i)_1 - i_3 \leq M(1 - E_{ij}), \tag{3.18}$$

$$M(E_{ij} - 1) \leq (D_j^i)_2 - i_4 \leq M(1 - E_{ij}). \tag{3.19}$$

It seems that this does not ensure that a package reaches its end-location and stays there. Indeed, it does allow a package to leave its end-location again. If a package leaves its end-location directly after it arrived at time j, the variable $E_{i(j+1)} = 0$ again. Because of the maximization in the end, removing packages from their end-location, will not occur if it has a negative influence on the solution. This also implies that if the number of movements is minimised in the objective function, the packages will not leave their end-location, since it would unnecessarily increase the objective. Like Equations 3.11 and 3.12, E_{ij} is undefined if a package is at its end-location. Again this is not a problem. We will explain this later.

The last auxiliary binary variable is $Q_j \in \{0, 1\}$, which indicates whether at time j all packages have arrived at their end-locations:

$$(P - \sum_i^P E_{ij}) \leq M(1 - Q_j). \tag{3.20}$$

Finally, the objective function is:

$$\max \sum_{j}^{N} Q_j. \tag{3.21}$$

Note that this objective implies that the number of steps needed overall to solve the problem, also known as the time span, is minimised. Of course, it can be modified if some other criterion is to be measured. For example, the total number of movements made by the vehicles. In that case, the following has to be added. First we introduce another binary variable $F_{jk} \in \{0, 1\}$, which keeps track of which vehicle j changes its position at time k: $F_{jk} = 1$ if $C_k^j \neq C_k^j + 1$. This is accomplished by

$$-M \cdot F_{jk} \leq (C_k^j)_1 - (C_k^j + 1)_1 \leq M \cdot F_{jk}, \tag{3.22}$$

$$-M \cdot F_{jk} \leq (C_k^j)_2 - (C_k^j + 1)_2 \leq M \cdot F_{jk}, \tag{3.23}$$

and the objective function will become:

$$\min \sum_{j}^{T} \sum_{k}^{N} F_{jk} + N - \sum_{j}^{N} Q_j. \tag{3.24}$$

In this objective function, we minimise the number of movements of all vehicles over time, represented by $\sum_j^T \sum_k^N F_{jk}$. However, it is important to include in the objective the requirement that all packages have to be at their end locations: $\sum_j^N Q_j$. Otherwise, the optimal solution is that the vehicles do not move at all. Because this time the objective function is a minimising function, instead of the maximising function of Equation 3.21, we have to rewrite the constraint that all packages have to be at their end locations as a minimising function too. This can be accomplished by subtracting $\sum_j^N Q_j$ from the total number of packages that should be transported: N. Minimising $N - \sum_j^N Q_j$ will eventually evaluate to zero, hence, the objective function will show the number of movements. Calculating the actual routes can be done by means of the solution, which consists the instances of all variables of the solution.

Let us get back to Equations 3.11 and 3.12. They are derived from $(C_k^i - D_k^j)S_{ijk} = 0$. It is obvious that if $(C_k^i - D_k^j) \neq 0$, then S_{ijk} has to be equal to 0. The other way around, hence if $(C_k^i - D_k^j) = 0$, then the value of S_{ijk} is undefined, it can be 0 or 1. It is not a problem since we maximize $\sum_j^N Q_j$ (Equation 3.21). To maximize $\sum_j^N Q_j$, all packages have to be as soon as possible at their end-locations. Hence, Q_j has to become 1 as soon as possible and keep that value. To get $Q_j = 1$, every package has to be at its end-location. This means that for all packages the corresponding variable E_{ij} (Equation 3.20) has to be

equal to 1 and preferably keep that value for every following time-step. So if it is possible to have the variable $E_{ij} = 1$, this will occur. Although it is allowed to move a package once it has reached its end-location, doing so will cause the corresponding variable E_{ij} to drop to 0 again and thus Q_j too. Therefore, it is unlikely that a package will be removed from its end-location.

The variables E_{ij} are (implicitly) dependent of Z_{ijk}. The variable Z_{ijk} allows the variable D_k^j, the tracking-variable of a package, to change (Equation 3.16 and Equation 3.17). Changing the variable D_k^j is necessary to be able to change variable E_{ij} (Equation 3.18). This means that we have to move packages in order to get them from their start-location to their end-location. the variable Z_{ijk} is obviously dependent of S_{ijk}, see Equations 3.13, 3.14, and 3.15. We have shown that in order to move packages, $Z_{ijk} = 1$ has to occur often. Only if $S_{ijk} = 1$ and $S_{ij(k+1)} = 1$, which is a possible outcome from Equations 3.12 and 3.12 when a package is at the same location as a vehicle, then $Z_{ijk} = 1$. Summarizing, any ambiguities occurring in the presented model are solved due to maximizing.

Finally, we have to be able to construct a valid plan by means of the output-variables of the model. The most important output-variables of this model are the sets of D and C. These variables represent the location where a vehicle or package is located at a certain time. These variables can be sorted per vehicle or per package and in increasing order of time. Each time the location at time t is unequal to the previous location, at time $t - 1$, the vehicle or the package has moved. Constraints 3.13, 3.14, 3.15, 3.16 and 3.17 ensure that a package can only be moved iff a vehicle makes the same move. So, every time a package changes from location, it was apparently loaded into a truck, moved, and unloaded. It is easy to see that by means of the sets of variables C and D, a valid plan can be constructed.

Example

To give a concrete example of an application of this model, we will use the following example. We have two cities, one with three locations and one with four locations (Figure 3.1). As said, we number all cities, we have a city number 1 and a city number 2, and all locations within a city, starting from 1. Now we can create the set of all nodes ($\mathcal{L}$), and the sub-graphs ($\mathcal{L}_i$): $\mathcal{L} = \{(1,1)(1,2)(1,3)(2,1)(2,2)(2,3)(2,4)\}$, $\mathcal{L}_1 = \{(1,1)(1,2)(1,3)\}$, $\mathcal{L}_2 = \{(2,1)(2,2)(2,3)(2,4)\}$, and the special case, the airports $\mathcal{L}_0 = \{(1,1)(2,1)\}$. Because the largest city has four locations, $\max\{m_{(i)} \mid i \in \mathbb{N}\} = 4$, and the number of cities equals 2, ($n = 2$), the set of $\mathcal{N}$, which represents the computation time, evaluates to the subset of natural number with a maximum of $2(\max\{m_{(i)} \mid i \in \mathbb{N}\}) + 2n = 2 * 4 + 2 * 2 = 12$.
Every city needs to have at least one truck and there has to be at least one airplane. Let us, for simplicity assume that we have no more vehicles than required, ergo, we

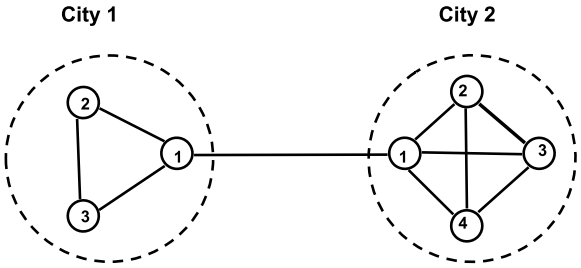


Figure 3.1: The graphical representation of the example we use to explain the our model. Reusing location-numbers is not a problem when adding the city-number for identification: Both airports have location-number 1, that is location (1,1) and location (1,2).

have three vehicles. The vehicles can start at a random location within their area, but let us assume that both trucks start at the third location, and the airplane starts in the first city. We can construct the set $\mathcal{T}$: $\mathcal{T} = \{(1, 1, 0, 1), (1, 3, 1, 2), (2, 3, 2, 3)\}$.

Let us assume that we have one package in each city that needs to go to the other city. Let us say that the first package starts in city 1 at location 2 and needs to go to city 2 location 2: $\mathcal{P}_1 = (1, 2, 2, 2, 1)$. The second package starts in city 2 location 3 and needs to go to city 1 location 4: $\mathcal{P}_2 = (2, 3, 1, 4, 2)$. Hence, set $\mathcal{P} = \{(1, 2, 2, 2, 1), (2, 3, 1, 4, 2)\}$.

The set C keeps track of the location where a vehicle has been at each moment in time. In our example, it is a 3×12 matrix because we have three vehicles and $N = 12$. Every row in this matrix represents the route which the vehicle follows. The first column in this matrix is filled with the starting locations of the vehicles, hence, $C_1^1 = (1, 1)$, $C_1^2 = (1, 3)$ and $C_1^3 = (2, 3)$. The other locations will be filled in while solving the LP.

The set D is similar to the set C, only this set keeps track of the packages. In our example, it is a 2×12 matrix because we have two packages and $N = 12$. Every row in this matrix represents the route which the package follows. The first column in this matrix is filled with the starting locations of the packages, hence, $D_1^1 = (1, 2)$ and $D_1^2 = (2, 3)$. The other locations will be filled in while solving the LP.

Until this point, we have described the initial state. Now, the LP has to be solved. The sets C and D will be filled in, as well as the three dimensional matrices, $|packages| \times |vehicles| \times N = 2 \times 3 \times 12$ each, that represent the auxiliary variables S and Z, the two dimensional matrix E and the set Q.

When the problem is solved the variables will look like:

$$C = \begin{bmatrix} (1,1) & (2,1) & (1,1) & (2,1) & (2,1) & \cdots \\ (1,3) & (1,2) & (1,1) & (1,4) & (1,4) & \cdots \\ (2,3) & (2,1) & (2,1) & (2,1) & (2,2) & \cdots \end{bmatrix}$$

$$D = \begin{bmatrix} (1,2) & (1,2) & (1,1) & (2,1) & (2,2) & \cdots \\ (2,3) & (2,1) & (1,1) & (1,4) & (1,4) & \cdots \end{bmatrix}$$

$$E = \begin{bmatrix} 0 & 0 & 0 & 0 & 1 & \cdots \\ 0 & 0 & 0 & 1 & 1 & \cdots \end{bmatrix}$$

$$Q = \begin{bmatrix} 0 & 0 & 0 & 0 & 1 & \cdots \end{bmatrix}.$$

Let us take one package, say the first package, the corresponding variable

$$S = \begin{bmatrix} 0 & 0 & 1 & 1 & 0 & \cdots \\ 0 & 1 & 1 & 0 & 0 & \cdots \\ 0 & 0 & 0 & 1 & 1 & \cdots \end{bmatrix}$$

and the corresponding variable

$$Z = \begin{bmatrix} 0 & 0 & 1 & 0 & 0 & \cdots \\ 0 & 1 & 0 & 0 & 0 & \cdots \\ 0 & 0 & 0 & 1 & 0 & \cdots \end{bmatrix}.$$

It is easy to see that the second and third variable in the second row of S should indeed be equal to one. The second row in S corresponds to the second vehicle. This vehicle is at this time (the second and third column), at the same location as the first package. Hence, moving the package is valid, and indeed happens as the second variable in the second row of Z shows.

3.5 Summary

The RPDPTW, can only model simplified Logistics Planning Problems, like those problems in which only one city exists. The LP model presented in previous section, can model all Logistics Planning Problems. An instance of this model can be solved by, for example, a solver using the simplex method.

Chapter 4

Coordination Background

In Chapter 2, we looked at planning problems as if only one party, program or agent would solve it. Of course, multiple agents can work together to solve the same problem. To have several agents working on the same problem at the same time, it is often needed to decompose the problem into smaller problems which are easier to solve. Finding a suitable decomposition technique for a problem, is often not trivial. Even if it is possible to create sub-problems by means of a decomposition technique, solving the sub-problems separately from each-other does not always end up in a feasible solution for the original problem. Coordination is needed to manage the partial solutions such that a feasible solution to the original problem will appear, especially when the parties, or agents are unwilling, or even unable to cooperate.

4.1 What is Coordination

In order to facilitate the construction of a feasible solution to an original problem from the partial solutions of the corresponding decomposed problem, sometimes a set of rules is added. This is to prevent the different participants to hinder each other. Adding those rules is part of coordination: “Coordination is the process of placing restrictions on the autonomy of individual parties, to ensure that every party can pursue his intended goals, in spite of, or thanks to the actions of other parties” [68].

Besides previous definition of coordination, there are many others in use. The one of Crowston and Malone often suits best [46]: “Coordination is the act of managing interdependencies between activities.” Coordination is needed as soon as multiple participants (or agents) are involved, and who are in one way or another interdependent. It includes choosing and temporally ordering the actions, which might be interrelated, such that the overall task is accomplished by the agent [17]. Coordination is also needed if multiple interdependent tasks have to be performed by one agent.

Another definition is from Jennings [41]: "Participation in any social situation should be both simultaneously constraining, in that agents must make a contribution to it, and yet enriching, in that participation provides resources and opportunities which would otherwise be unavailable" [27]. Coordination, the process by which an agent reasons about its local actions and the (anticipated) actions of other agents to try to ensure the community acts in a coherent manner, is the key to achieving this objective.

4.2 Coordination and Planning

In Section 4.1, we have seen that "Coordination is the act of managing interdependencies between activities" [46]. Interdependencies exist between the actions of agents when others have the necessary expertise, or the right resources, or an important piece of information. Other dependencies are caused by global constraints, like a fixed maximum budget. All agents have to coordinate their actions such that the budget is not exceeded. It is possible to distribute the maximum budget evenly among the agents. However, when an agent hardly uses its budget, another can use it without violating the global constraint. This shows another dependency: the actions of one agent influence the actions of another agent [41, 52].

Above the afore mentioned reasons to cooperate, Nwana and Jennings [41, 52] identify two general reasons. Those general reasons are the prevention of chaos or anarchy, and the possible increase of efficiency. The first of these two means that because agents only have an overview of their own activities, coordination is needed to prevent conflicts with other agents. The second points out that agents working together can often solve problems faster. Due to the first reason, interdependencies have to made explicit to preserve a feasible solution to the original problem, when it is decomposed.

When the participating agents are selfish, they do not wish to share information, not even information of the interdependencies they are subject to. In that case, the above reasons are not sufficient to persuade them to cooperate. According to Decker and Lesser [19], the only reason for a selfish agent to cooperate is: if its own performance is affected. Performance is affected if either *(i)* an agent has a choice of actions, or *(ii)* by the order in which activities are carried out, or *(iii)* by the time at which actions are executed. Although, both Decker and Lesser, and Nwana and Jennings identify reasons why coordination is needed, nothing is mentioned about when decomposed planning problems have to be coordinated.

Coordination of the decomposed planning problem can be performed *before*, *during*, or *after* the actual sub-plans are made, hence, it is called pre-planning

coordination, during-planning coordination, and post-planning coordination. During-planning coordination is sometimes seen as a repetition of planning and pre- or post-planning coordination, and is, therefore, sometimes also called interleaved coordination [72]. This division is independent of the particular coordination technique used, and focuses on coordination as a process. It is obvious that these categories are orthogonal and span the space of solutions for multi-agent coordination situations.

- **Pre-planning coordination**: In some situations all potential conflicts can be prevented by using a coordination method before any planning has taken place. If that is the case, the coordination mechanism facilitates the agents to plan and execute those plans without having to worry about interference from other agents [68].

- **During-planning coordination**: While creating plans, agents decide which actions will realize their goals. This decision is influenced by the amount of coordination that is needed for each choice. For instance, some actions lead to conflicts with other agents, others do not [68].

- **Post-planning coordination**: At first, agents are given complete freedom in the planning process. As soon as all sub-plans have been made, the plans are coordinated. This implies that the agents have to revise their initially developed plans. Post-planning coordination methods usually try to maintain the initial set of plans, therefore, successful post-planning coordination is only possible if no irresolvable conflicts can exist between plans of different agents [68].

4.3 Pre-planning Coordination Techniques

It has been shown that planning can be coordinated at three different moments. The coordination method we have used, is a pre-planning coordination method. Two pure[1] pre-planning coordination strategies are *Social Laws* and *Filtering* [68].

- **Social Laws**: These are extra rules which decrease in advance, the need of communication between agents as much as possible. It prevents common conflicts, by giving rules of how to behave. An example is the convention of driving on the right lane, at least on the main land of Europe. If every one obeys these social laws, conflicts can only arise in special occasions. In principle, if for every situation, a social law exists, agents can make plans without being afraid of interference from other agents. However, a social law is only useful if it is not too strict. It has to leave agents enough freedom

[1] A pure pre-planning coordination strategy is an coordination strategy which is applied after decomposition and before planning. Even though decomposition happens before planning, we do not call the coordination strategies applied at that stadium, pre-planning coordination.

to find plans that will accomplish their goals. Therefore, social laws are generally hand-crafted for the problem at hand, which is a downside. Shoham and Tennenholtz [63], however, provide a general model for social laws in multi-agent systems.

- **Filtering**: Filtering is a strategy designed for agents in dynamic environments [21]. This strategy filters out the options in the search space of the agent that are incompatible with its goal or goals. When multiple agents are involved, also options that would be incompatible with other agents' goals are filtered out. Removing options for the sake of other agents reduces the number of possible actions for the filtering agent. That does raise the question that Ephrati, Pollack and Ur [21] ask themselves: "Should rational agents employ a filtering strategy?" Fortunately, if an option looks promising but it does interfere with some goals, an override mechanism can be used. Whether an agent considers this interfering option depends on a threshold, or how bold the agent is [21]. Unfortunately, no proof is given yet that filtering is the dominant strategy. However, this does not mean that filtering can not ever be useful. Geographical filtering is often a good choice.

4.3.1 Arbiter-0 and the Depth-Partitioning Algorithm

The pre-planning coordination technique we have used during this research project is based on the *Arbiter-0* method proposed by Valk [72], which was generalised as the *depth-partitioning algorithm* by Steenhuisen [66]. The general idea of these methods is twofold. The first is to enable the decomposition of an original problem into a set of sub-problems in such a way that all agents can solve their assigned sub-problem independently from the other agents. In other words, when agents are generating sub-plans, they are completely autonomous. The second is that it facilitates the sub-plans generated by the individual agents to be easily be joined together to form a joint plan for the total planning problem, without having to revise a sub-plan [69].

In the Arbiter-0 method and the depth-partitioning algorithm tasks are units of work that an agent can perform by itself. It is also assumed that dependencies between tasks exist, which, without loss of generality, can be specified as a precedence relation. A precedence relation between tasks exist if it is necessary that one task is performed before the other. Such a precedence relation imposes a partial order on the tasks. A sequence of tasks with precedence relations between them, is called a complex task. When tasks are distributed among agents, it is possible that parts of a complex task are assigned to different agents. The precedents relations between tasks assigned to the same agent are called *intra-agent* constraints, those between tasks assigned to different agents are called *inter-agent* constraints. Regardless of which particular planning method is used by the agent, it is also assumed that all agents generate plans which take the original precedence

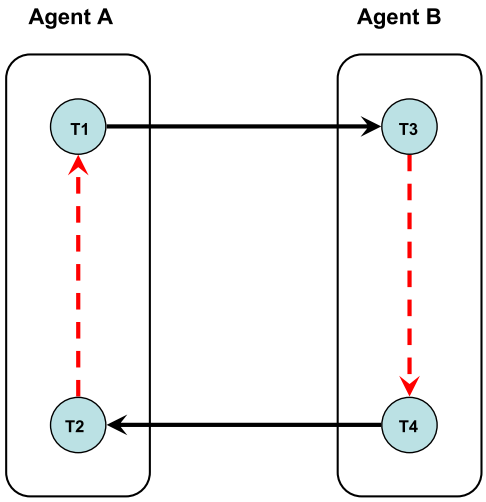


Figure 4.1: The dots represent tasks, the solid arrows represent precedence relations, which in combination with the tasks form complex tasks. The dotted arrows represent the precedence relations added due to a free choice of the individual agents, which causes a deadlock to occur.

relations into account. In addition, the agents are allowed to specify extra ordering constraints [69].

The depth-partitioning method of Steenhuisen [66], is a method to find extra precedence constraints which have to be added to a decomposed planning problem before any planning is done by the individual agents, and which prevent deadlocks. Deadlocks can easily occur when the requirements of autonomy and not having to revise the sub-plans while constructing the joint plan, have to be met, due to the freedom of choice of an agent to perform one task before another. For example, two agents, agent A and agent B, each having to perform two tasks. Agent A has to perform task 1 and 2, agent B has to perform task 3 and 4. Task 1 and 3 form a complex task, for which task 1 has to precede task 3. Also, task 2 and 4 form a complex task, for which task 4 has to precedes task 2. If agent A chooses to perform task 2 before task 1, and agent B chooses to perform task 3 first, a deadlock would occur: 2 is waiting for the completion of 4, is waiting for 3, is waiting for 1, is waiting for 2,see Figure 4.1. To prevent such cycles, additional constraints have to be added [69].

The depth-partitioning algorithm is composed of five simple steps, of which we will give an intuitive description. For the precise algorithm we refer to the third section of [66]. The first step is to number each task in a complex task, starting from 0 and each task has a higher number than all of its preceding tasks. This number represents the depth of the task, i.e., how many tasks has to be performed before this task can be executed. The second step is to distribute the tasks among

the agents, while taking into account the existing precedence relations. The third step is to sort for each agent, the tasks in increasing order of depth. Then, step four, we add for every agent precedence constraints which dictate that all tasks at a lower depth have to precede all tasks of a higher depth within an agent. Hence, only intra-agent constraints are added. The final step is to add all extra constraints, which we have found in the previous steps, to the planning problem. Note that this algorithm can also be used as a decomposition method. All tasks of the same depth can be assigned to the same agent, though this is not necessary.

Applying the depth-partitioning algorithm to the Logistics Planning Problem, results in the Arbiter-0 method and boils down to the following. Every transportation task is a complex task. Every complex task contains a truck-transport task, an airplane-transport task, and another truck-transport task, in that order. Any two of these tasks are allowed to be empty, resulting in only one transport task. Also, either one of the truck-transport tasks may be empty too, resulting in only an airplane-transport and one truck-transport task. Whether all tasks are non-empty or not, all complex tasks can be decomposed into a pre-flight, in-flight, and post-flight tasks. The pre- and post-flight tasks are assigned to truck-agents, and the in-flight tasks are assigned to airplane-agents. The depth partitioning algorithm introduces to the truck-agents, the constraint that all pre-flight tasks (depth 0) have to be performed before any of the post-flight tasks (depth 2). The only existing precedence constraints that are left are inter-agent constraints. These constraints enforce that the pre-flight tasks are performed before the in-flight tasks, and those have to be performed before the post-flight tasks. However, the agents do not have to keep these constraints in mind while planning. They can plan autonomously. Only when the plans are assembled into a joint plan, these constraints have to be taken into account [69].

Summarising, decomposing the Logistics Planning Problem into pre-, in-, and post-flight phases, is a suitable decomposition technique which requires little coordination. Agents only need to know when it is safe to start each phase, and they can solve each sub-problem by itself without further communication. The benefit of this technique is that we end up with smaller, and because of that often easier, problems. A drawback is that the solution of the decomposed problem might be sub-optimal. However, this loss in quality is limited. It has been proven that the performance ratio is only 1.25, i.e., it is at most 1.25 times worse than the optimal solution, which is measured in movements [70].

Chapter 5

Experiments

In Section 3.4, we have shown that the Logistics Planning Problem can be modelled as a VRP. Therefore, we can compare the programs used for solving these problem instances of both the AI community and the OR community. The results of the comparison we made, are presented in this chapter.

5.1 Setup

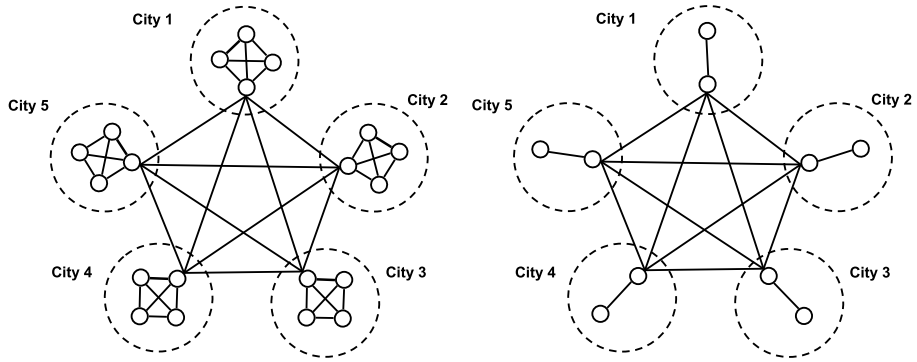
The Layout of the Problem Instances

The Logistics Planning Problem has been used in both the AIPS competition of 1998 and of 2000 [47, 48]. The problem instances of the Logistics Planning Problem of the AIPS competition in 1998 were different from those of the AIPS competition in 2000. In 2000, the graphs were of the following form: A complete graph with one path extra at each node which leads to a location within that city. In other words, the cities only contained two locations, one of which was an airport (see Figure 5.1(b)).

A more challenging graph was used in 1998, which not only used a complete graph to connect the cities, but the cities themselves were also modelled as complete graphs of more than two locations (see Figure 5.1(a)). Since we will also use the depth-partitioning algorithm to decompose the problem instances, which in case of the problem instances of the 2000 AIPS competition, results in trivial sub-problems, we will use the problem instances from the 1998 AIPS competition.

Plan of Evaluation

In these experiments, we try to answer the main questions raised in Section 1.5. The main differences between AI planners and OR solvers, are investigated by finding answers to the sub-questions raised in that same section. These answers can be deduced from the results of the experiments on the several versions of the problem instances of the 1998 AIPS competition. These versions are the



(a) Layout in 1998. (b) Layout in 2000.

Figure 5.1: The layout of the graphs used in the AIPS competition of 1998 (Figure 5.1(a)) and 2000 (Figure 5.1(a)). The small circles are locations, the solid lines are the connections between the locations. The dashed circles are only to show that a collection of locations is called a city. In each city, only one location is connected with other locations in different cities, such location is an airport.

problem instances which are the result of pre-processing, which is decomposition and filtering. We have decomposed the original problem instances into a pre-flight, in-flight and post-flight problem instance. We have filtered each of the original problem instances, as well as the corresponding decomposed problem instances, see Figure 5.2. If a solver needed a different format, we have generated equivalent problem instances for them. Details on how we have generated these problem instances can be found in Appendix B.1, B.2, and B.3. To see whether a difference in performance would occur between the AI planners and the OR solvers when the problem instances would become larger, we created our own set of problem instances, using the same layout of the problem instances of the 1998 AIPS competition. We used these self-generated instances in our second set of experiments. Details on how we created these larger problem instances, can be found in Appendix B.4.

The effect of the pre-processing steps we applied to the problem instances (decomposition, filtering, and both), appear when we compare the problem instances before and after pre-processing. The effect of enlarging the problem instances become apparent because we have generated our own sequence of problem instances from small instances to large instances. The largest problem instances are up to approximately three times the size of the largest problem instance of the first data set.

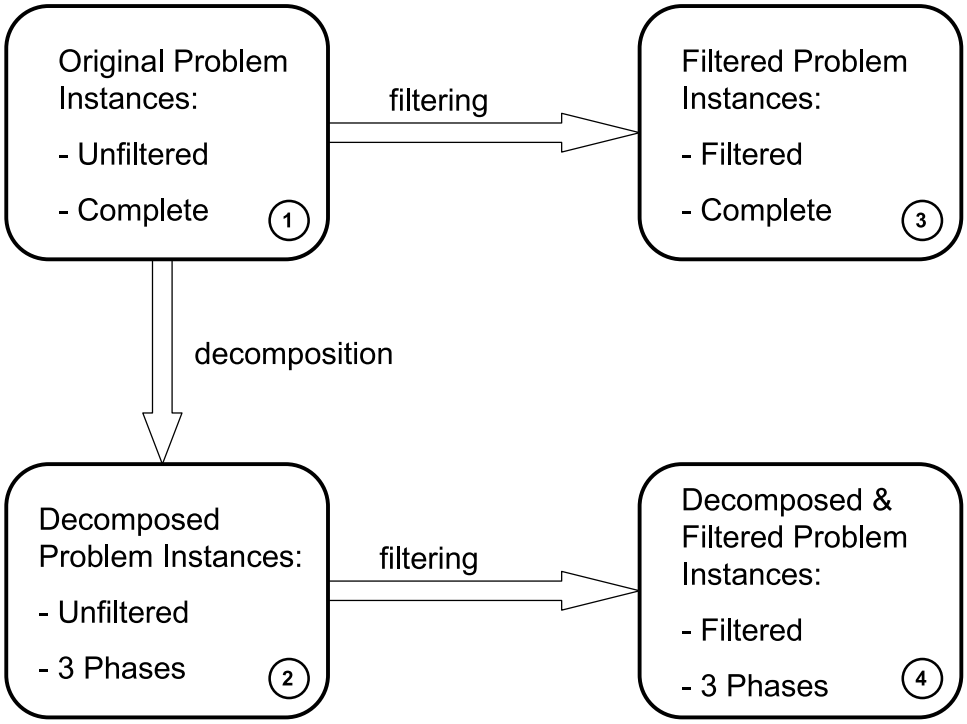


Figure 5.2: An overview of the problem instances we use. Starting from the original problem instances, we have filtered them to get the filtered complete problem instances. Decomposing the original problem instances resulted in unfiltered decomposed problem instances. Filtering the latter results in filtered decomposed problem instances.

Evaluation Criteria

We have compared the output of our experiments on computation time and plan quality. Plan quality is based on the total number of movements made by all vehicles, i.e., airplanes and trucks, in the plans, which result in a solution is to the problem instance. The less movements needed, the better is the plan quality. computation time is important because no-one wants to wait a long time for the solution of a problem. Plan quality is also important because a company often wants to minimise transportation costs. Transportation costs are dependent on the means of transportation. Although in our problem, we have vehicles and airplanes, the costs of using either of them, is equal. Therefore, the costs of which type of transportation is used, is not an element in our problem. Another factor on which transportation costs are dependent is the distance over which goods are to be transported. Therefore, it is important to make as little detours as possible and thus to have routes which have together the best plan quality.

Environment

The tests have been run on a computer with 16Gb RAM, an Intel Xeon E5345 processor, with 2.33 GHz and 4096 kb cache. The AI planners we have used are YAHSP (version 1.1), FF (version 2.3) and SGPlan (version 2), their main properties are covered in Section 2.4. CPLEX [39] (version 11.0) has been used as the OR representative and in case of the decomposed problem instances, ALNS-PDPTW [57, 58, 59] by Stefan Røpke has been used too. The main properties of these two solvers, were covered in Section 3.2.

Concluding, with the exception of ALNS-PDPTW, all planners and solvers are used in the experiments. As explained in Section 3.3.2, the RPDPTW is not fit to model complete Logistics Planning Problem instances. Complete Logistics Planning Problem instances are the original and the corresponding filtered problem instances. However, it could be used to model both filtered and decomposed problem instances, as explained in Section B.3. In case of the unfiltered decomposed instances, we have only removed the superfluous packages from the problem instances for ALNS-PDPTW. Otherwise, we could not represent the problem instances in the format required by ALNS-PDPTW. Fortunately, this does not make our comparisons unfair, since the AI planners also do not take the superfluous packages into account for any problem instance.

We also adapted the manner in which we let CPLEX solve the problem instances. In Section 3.4.1, we showed that the maximum number of movements needed to solve a problem instance was $2 * (|locations| + |airports|)$. We implemented this constraint in the model in CPLEX, however, it turned out to cause CPLEX to run very slow: CPLEX did not stop when an optimal solution was found, but when the maximum time was reached. We decided to use an iterative approach, similar to the approach used in Satisfiability solvers. We started with a tight time constraint and we increased it by one, each time CPLEX could not find a solution. The effect of this iterative approach is that CPLEX was given a harder objective to calculate: minimising the number of movements in a minimal time-span.

5.2 Expectations

We have seen in Section 1.2, that the OR community has been working on VRP problem instances much longer than the AI community has worked on logistic problems. Therefore, we expect that specialised OR solvers to outperform the AI solvers in both computation time and plan quality.

However, CPLEX is not a specialised solver for the Logistics Planning Problem. Therefore, we expect that CPLEX will need the most computation time of all, because, unlike the AI planners, it does neither use look-ahead algorithms, nor any

pre-processing step to reduce the search-space. Therefore, CPLEX will evaluate the most states in the search-space, which will probably cause it to be rather slow. However, if it will get an answer, it will be optimal, hence, it will generate the best plan-quality.

ALNS-PDPTW is expected to generate also better plan quality then the AI planners, though this assumption is mainly based on the previously mentioned seniority of the field of research. Since this solver was not designed for the Logistics Planning Problem, certain assumptions made in this solver do not hold, which is expected to hamper the performance. One of these assumptions will be discussed in Section B.4. Details on the specifications of the model ALNS-PDPTW was designed for, can be found in Appendix A.3.

Of the AI planners, we expect YAHSP to be the fastest, since this would be conform the conclusions of Coles and Smith [11]. However, we consider the possibility that this high speed comes at the cost of lower plan quality. FF and SGPlan are expected to perform between YAHSP and the OR solvers on both evaluation criteria.

With respect to the pre-processing steps we performed on the problem instances (decomposition, filtering, and both), we do not expect either the AI planners or the OR solvers to benefit more than the others from these pre-processing effects. Decomposing the problem instances is expected to cause a large speed-up, while its effect on plan quality is expected to be small, a neither very positive nor very negative effect. The effect of filtering is not expected to cause a large speed-up, though, it might be the cause of generating plans with higher quality.

Since VRP solvers are presently tested on problem instances up to 1,000 nodes, and because ALNS-PDPTW is record holder on some of these problem instances [33], we expect ALNS-PDPTW to overtake YAHSP with respect to the performance measured in time when we let it run on the larger problem instances to them.

5.3 Results

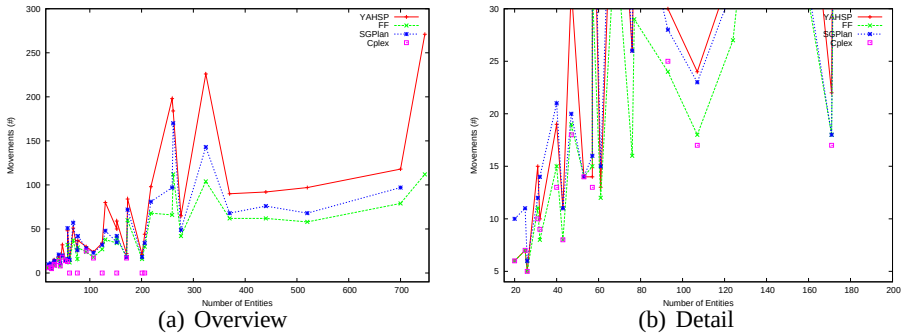
Summarising Section 5.1, we have run tests on YAHSP, FF, SGPlan, CPLEX, and ALNS-PDPTW. We tested four different kinds of problem instances: *(i)* the original or complete unfiltered problem instances, *(ii)* the decomposed unfiltered problem instances, *(iii)* the complete filtered problem instances, and *(iv)* the decomposed filtered problem instances. CPLEX had, as expected, much trouble on many problem instances. Therefore, we have little measurements of this solver, approximately 50% of the first test set. For larger graphs, we refer to Appendix C.

5.3.1 Results without Pre-Processing the Problem Instances

Before we compare the results of the problem instances before and after decomposition, filtering, and both, we present the results in the initial situation.

Plan Quality

In Figure 5.3(b), we see that CPLEX not always generates plans with the best plan quality, i.e., plans with the least number of movements. However, this was expected, since it is an optimal solver. We will get back on the reason for these unexpected sub-optimal results in the next section. It is followed by FF, SGPlan and YAHSP, see Figure 5.3(a). It also seems that the plan quality of FF is near optimal. CPLEX shows also some values of 0 movements. The explanation to the result of 0 movements is that CPLEX had completed at least one round with a tight time constraint without finding a solution. In that case, we also cut off the computation, because it took too long.



(a) Overview (b) Detail

Figure 5.3: The number of movements needed to solve the original problem instances.

Computation Time

In Figure 5.4, we see that the computation time needed to solve the problem instances is mainly small. Only CPLEX seems to have trouble with, from left to right, problem number 35, 16, 4, 6, 24 and 13, which we cut of after approximately three hours. The other planners also show spikes, although these spikes do not correspond to the same problem instances CPLEX has such trouble with. The spikes especially shown by FF represent, from left to right, problem number 29, 22, and 28. The reason why these latter three spikes occur can be found when looking at the graphs of absolute number of movements, Figure 5.3. In these graphs, the spikes reappear, corresponding to the same three problem instances. Obviously, there is a connection between the length of the path and the time it takes to calculate it.

In case of the larger problem instances, a difference between the AI planners presents itself. YAHSP is still fast, FF is the slowest, the performance of SGPlan is in between the previous two. This is exactly the opposite of the performance measured in plan quality, where the worst performance was that of YAHSP and FF was the best, at least of the AI planners.

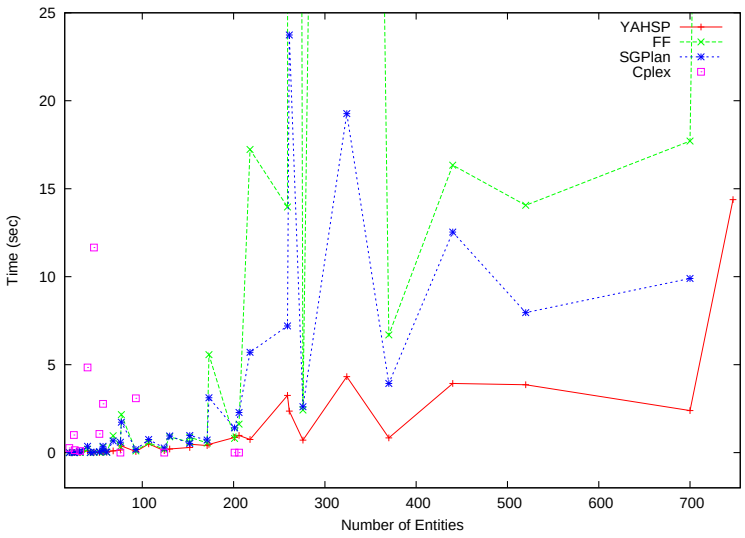


Figure 5.4: The computation time that was needed to solve the original problem instances.

Summary

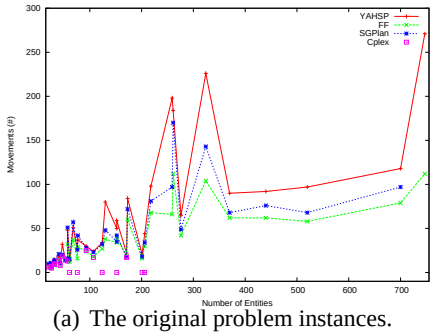
We can conclude that a trade-off exists between the plan quality and the computation time. FF and CPLEX generate the best plans, though use the most computation time. We already expected CPLEX to have a bad performance measured in time, since no preprocessing methods are used contrary to the AI planners, see Section 5.2. We recommend further research to investigate whether a preprocessing method like GraphPlan would improve the performance measured in time of CPLEX.

5.3.2 The Effect of Decomposing the Problem Instances

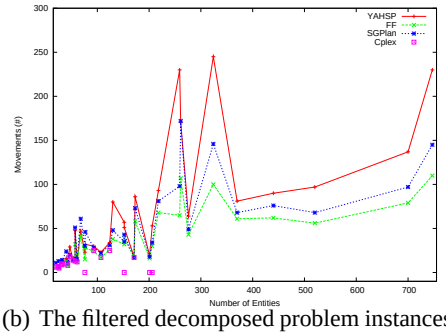
The problem instances we have decomposed are the original problem instances. Recalling Figure 5.2, we present in this section the effect of filtering by comparing problem instances from box (1) with problem instances from box (2). We have used the decomposition method presented in Section 4.3. In Appendix B.2, we describe how we applied the decomposition method to the problem instances.

Plan Quality

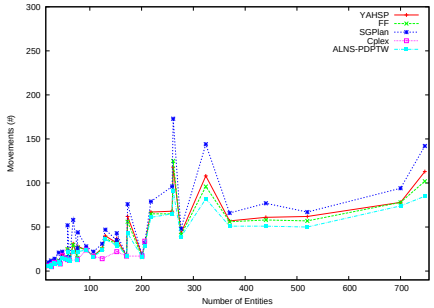
In Figure 5.5, we can see the effect of decomposition to the number of movements when we compare the Figure 5.5(a) with Figure 5.5(c). In the lower graphs, we have combined the plan quality of the pre-, in-, and post-flight phases, i.e., we added the results from the decomposed problem instances and compared this with the result of the original problem instances. Except for YAHSP, which shows a large improvement in plan quality, the effect of decomposition on the plan quality generated by the other programs seems not that large. We can also see that in case of the decomposed problems and in the absence of a full series of results of CPLEX, ALNS-PDPTW generates the best plans. These graphs also show that CPLEX is able to solve more problem instances after decomposition.



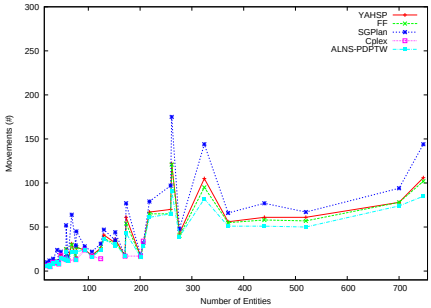
(a) The original problem instances.



(b) The filtered decomposed problem instances.



(c) The unfiltered decomposed problem instances.



(d) The filtered decomposed problem instances.

Figure 5.5: The number of movements to find a solution to the problem instances. The upper graphs show the complete problem instances, the lower graphs show the decomposed problem instances, the left graphs show the unfiltered problem instances, and the right graphs show the filtered problem instances.

In Figure 5.6[1], we show the relative effect of the plan quality of the original problem instances to the combined plan quality of the corresponding in-, pre- and post-

[1]Instead of the first and third quartile, we show the standard deviation in these box-and-whisker plots.

flight phases. As we already concluded from Figure 5.5, YAHSP generates much better plans after decomposition in comparison to the other AI planners. The other AI planners do on average generate better plans, but not as much, since their plan quality was rather good to begin with.

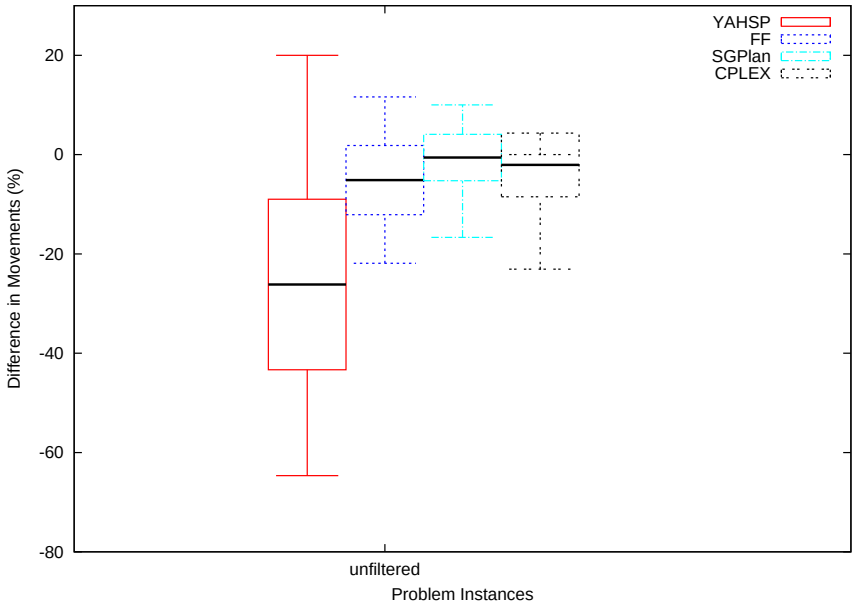
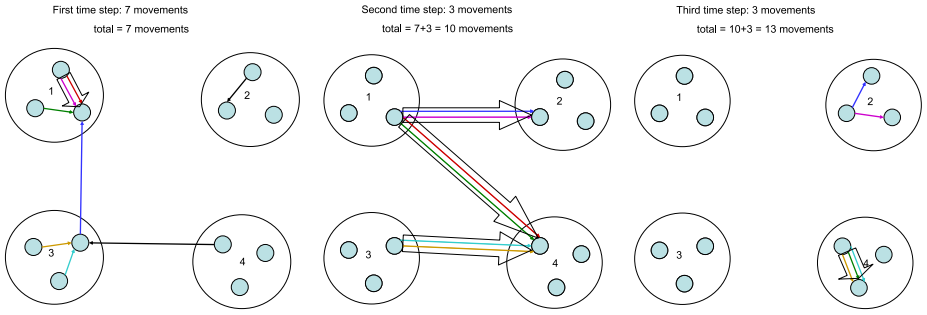


Figure 5.6: The relative effect of decomposition on the number of movements. Negative values mean improvement in plan quality and in computation time.

The objective function for CPLEX was to minimise the number of movements in a minimal time span. Therefore, the plans generated by CPLEX are sometimes sub-optimal, due to the following effect. As described in Section 5.1, we let CPLEX run iteratively. Running CPLEX iteratively caused that a solution was found after, for example, three iterations, hence, after the minimal time span of three. However, if one extra time step would have been used, more packages could have shared a vehicle, and therefore, reducing the number of movements needed to solve the problem. In Figure 5.7 and Figure 5.8, we present this anomaly as it occurred in problem 17 of our test set. This anomaly is the cause of the small improvement shown for CPLEX in Figure 5.6, and also of the sub-optimal results in Figure 5.3. Otherwise, the graph for CPLEX would have shown a line at zero. This is also true for the overview of the filtered problem instances, only this time, the reason for the distortion is a cut-off computation.

Computation Time

In Figure 5.9(a), we show the difference in seconds between the original problem instances and the corresponding decomposed problem instances. In Figure 5.9(b),

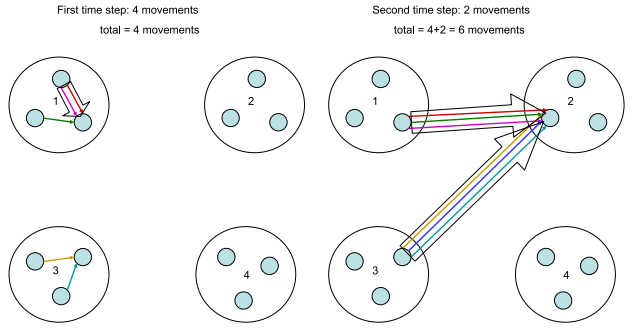


(a) The movements made during the first time step.

(b) The movements made during the second time step.

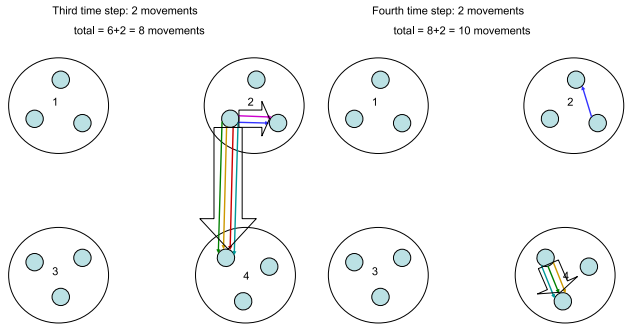
(c) The movements made during the final time step.

Figure 5.7: The plan per time step for problem 17, using three time steps. Black arrows represent empty vehicles, coloured arrows represent vehicles transporting one package, block arrows represent vehicles transporting multiple packages.



(a) The movements made during the first time step.

(b) The movements made during the second time step.

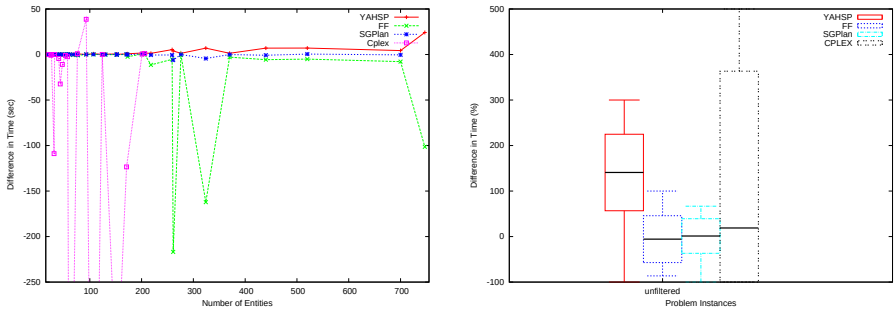


(c) The movements made during the third time step.

(d) The movements made during the final time step.

Figure 5.8: The plan per time step for problem 17, using four time steps. Coloured arrows represent vehicles transporting one package, block arrows represent vehicles transporting multiple packages.

we show the relative difference between the original problem instances and the corresponding decomposed problem instances. In case of SGPlan, it occurred once that the complete problem instance took 0 seconds to be solved and after decomposition, a larger computation time was measured. Since this is only one result, we left it out when constructing Figure 5.9(b). We have not left out the worst results of CPLEX, since those were actual measurements.



(a) The effect in seconds. (b) The relative effect.

Figure 5.9: The effect of decomposition on computation time of the original problem instances. Negative values mean that the decomposed problem instances are solved in less seconds than the original problem instance.

We can see that both speed-up and slow-down occur. Looking at Figure 5.9 it shows that YAHSP performs constantly worse. We assume that the initialization phase in YAHSP gets in the way of good results because YAHSP is quite fast in solving the problem instances. This means that, for example, the complete problem is solved in 0.01 seconds, but the individual phases also. Hence, it seems that YAHSP slows down considerably, while it might be that this time is just needed to load the problem. The results of CPLEX suffer from some large computation times, before and after decomposition, which is the cause of the large standard deviation. If somehow, those extreme long computation times could be prevented, it might be possible that decomposition would cause a considerable improvement to the computation time needed for CPLEX to solve the problem instances.

Unfortunately, from the statistics in Figure 5.9, no trend becomes visible. We guess that the small computation times before and after decomposition (compare Figure 5.12(a) with Figure 5.12(c)) and therefore the small differences (see Figure 5.9(a)), are responsible for the extreme values in Figure 5.9(b).

Summary

The effect of decomposing the problem instances is that especially YAHSP generates on average plans with better quality than is the case when no decomposition is used. In case of the other programs, the effect of decomposition on the plan qual-

ity is almost negligible. As mentioned, decomposition of these problem instances does not show a clear effect on the computation time, which is probably the cause of the small computation times before and after decomposition.

5.3.3 The Effect of Filtering the Problem Instances

The problem instances we have filtered are the original problem instances. Recalling Figure 5.2, we present in this section the effect of filtering by comparing problem instances from box (1) with problem instances from box (3). For the criteria on which we have filtered, we refer to Appendix B.1.

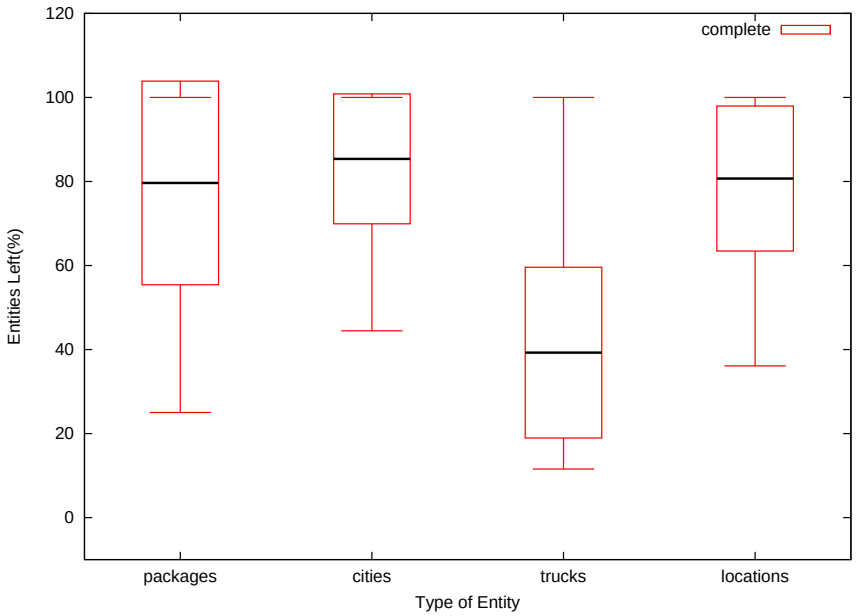


Figure 5.10: The percentages of the entities left after filtering the original problem instances: 0 meaning nothing is left, 100 means everything is left.

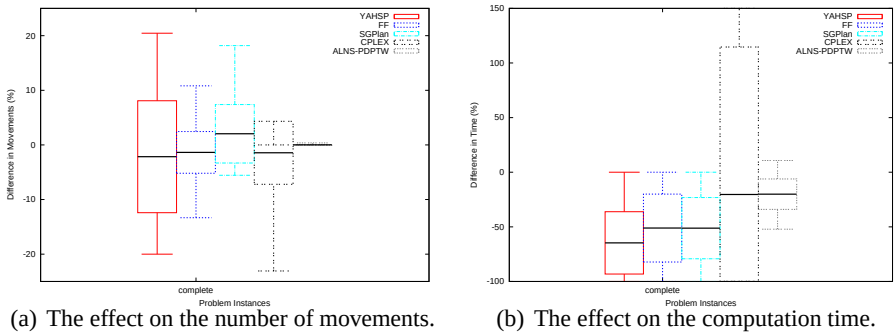
In Figure 5.10, we show the percentage of entities, packages, cities, trucks and locations per city, that is left after filtering the unfiltered problem instances. It shows that in the problem instances, especially trucks are present in an excessive amount. Also, many packages can be removed: In 23% of the problem instances, 40% or more of the packages in the original problem instance, is removed.

Plan Quality

With respect to the plan quality, we did not expect to see large differences between the unfiltered and filtered problem instances. Indeed, when we compare Figure 5.5(a) with Figure 5.5(b), no large differences are visible. Even Figure 5.11(a), which presents the relative difference in percentages between the unfiltered and

filtered problem instances on the number of movements, shows little differences on average. This figure also shows that the plan quality occasionally improves, and occasionally deteriorates. The plan quality of FF tends to improve after filtering, just as the plan quality of YAHSP. The plan quality of SGPlan tends to deteriorate. The plan quality of CPLEX stays the same in the worst case, most often the plan quality improves.

Since we do not explicitly divide the problem instances of ALNS-PDPTW into several phases, these phases are defined within the problem instance, we have included the effect of filtering on ALNS-PDPTW problem instances in Figure 5.11. ALNS-PDPTW does not respond at all to filtering, with respect to the number of movements.



(a) The effect on the number of movements. (b) The effect on the computation time.

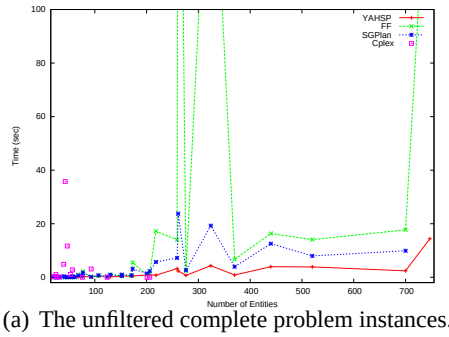
Figure 5.11: The relative effect of filtering the original problem instances. Negative values mean improvement in plan quality and in computation time.

We can think of an explanation of the occurrence of both deteriorations and improvements. When we think of the problem as a set of equations, it is possible that superfluous data obscures the underlying problem. When this superfluous data is removed, think of it as simplifying the set of equations. When a set of equations are simplified, the problem at hand often becomes easier, which causes the performance to improve. On the other hand, when we think of the problem as a riddle, removing superfluous data could be seen as removing clues. It is possible that the superfluous data are used to exclude certain possibilities, hence when they are removed, the problem becomes harder.

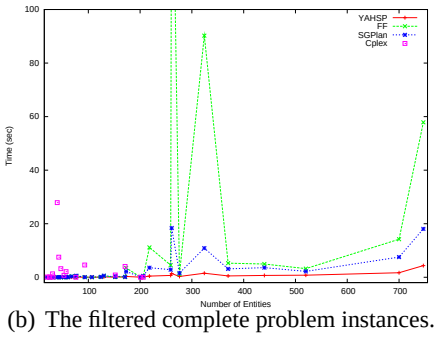
Computation Time

If we compare Figure 5.12(a) with Figure 5.12(b), we can see the effect of filtering the problem instances on the computation time. We can see that in general, the computation times become shorter after filtering.

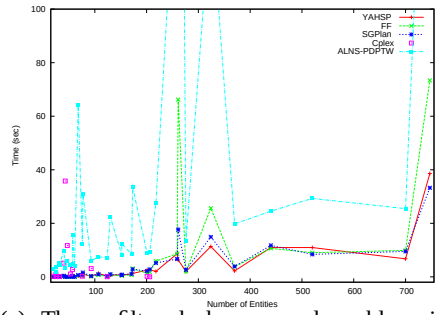
In Figure 5.11(b), the relative effect of filtering with respect to the computation



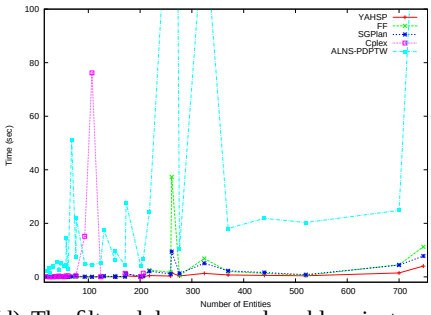
(a) The unfiltered complete problem instances.



(b) The filtered complete problem instances.



(c) The unfiltered decomposed problem instances combined.



(d) The filtered decomposed problem instances combined.

Figure 5.12: The computation time in seconds that was needed to solve each problem instance. In case of a decomposed problem instance, we added the computation time of the in-flight phase to those of the post- and pre-flight phases.

time is shown. We can see that all AI planners show considerable speed-up after filtering. For ALNS-PDPTW the graph shows that occasionally a small slow-down occurs, though mainly the computation time becomes shorter. Especially CPLEX shows some slowing-down. CPLEX solves the problem instances either rather fast, or extremely slow for which we have cut-off the computation. Especially the extremely long computation times cause the average measurements and standard deviation to become this large. As already mentioned in Section 5.3.2, if somehow these large computation times could be avoided, it could be possible that CPLEX will show a considerable speed-up.

Summary

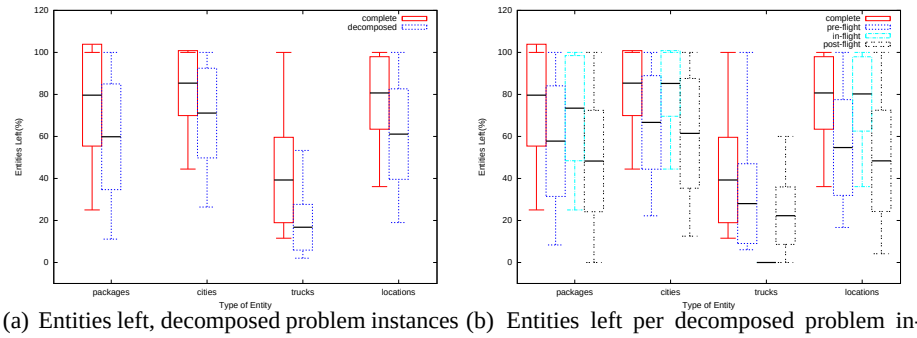
Filtering the problem instances has little effect on the plan quality, it neither improves nor deteriorates much. At least for the AI planners, filtering largely improves the needed computation time. The computation time needed by ALNS-PDPTW generally improves too, though not as much compared to the AI planners. The plan quality of ALNS-PDPTW is not at all sensitive to filtering. Keeping in mind the small test set used for CPLEX, filtering seems to have a mixed effect on the computation time it needs. We assume that it is caused by the extreme long

computation times which CPLEX seems to need to find an answer to the harder objective function used by CPLEX. We recall that this harder objective function is to minimise the number of movements in a minimal make-span. For CPLEX, filtering seems to have little to no effect on the plan quality.

5.3.4 The Combined Effect of Decomposing and Filtering the Problem Instances

To measure the combined effect of decomposing and filtering the problem instances, we compare the original problem instances with the corresponding filtered and decomposed problem instances. Hence, we compare problem instances of box (1) with those of box (4) in Figure 5.2.

To create the problem instances of box (4), we have filtered the problem instances of box (2). In Figure 5.13, we show the minima, maxima, averages and standard deviation in percentages of packages, cities, trucks and locations which are left after filtering. Figure 5.13(a) shows that in the decomposed problem instances, all entities are even more abundantly present than in the original problem instances. This is not surprising since each of the decomposed problem instances is a smaller problem instance than the original problem instance. To create each of the decomposed problem instances, only the start and end locations of the packages are adapted. Other data remains in the problem instance, see Appendix B.2, even when it might have become irrelevant, for example trucks in the in-flight phase.



(a) Entities left, decomposed problem instances combined.

(b) Entities left per decomposed problem instance.

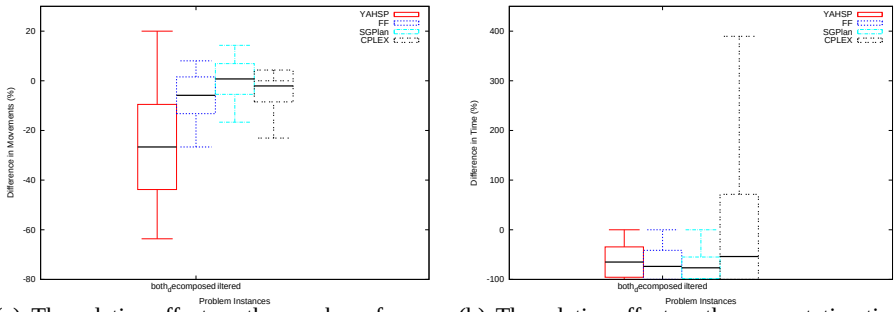
Figure 5.13: Entities left after filtering the (decomposed) problem instances in percentages: 0 meaning nothing is left, 100 means everything is left.

When we compare the in-flight phase with any of the other phases, see Figure 5.13(b), we notice that the in-flight phase benefits the least from the filtering. The statistics of the in-flight phase are more similar to those of the complete problem instances. Of course, all packages present in the complete problem instance

are present in the in-flight phase, save for a few packages which have their destination within the same city as where they start. After all, all packages which have their destination in another city, need to be brought there by airplane. Because, in the in-flight phase, all cities are reachable, all cities have to exist. As explained in Appendix B.1, we could have removed all locations except the airports. However, since those locations are not reachable, because all trucks have been removed, we considered that solving algorithms would not consider visiting them. Hence, we supposed it would not matter if we left all locations in the in-flight phase, which is the reason for the relative large amount of locations that is left in the problem instances.

Plan Quality

We can see the effect of both decomposition and filtering on plan quality when we compare Figure 5.5(a) with Figure5.5(d). If we compare Figure5.5(c) with Figure5.5(d), it can be suspected that the combined effect of decomposition and filtering, is mainly determined by decomposition. Namely, when comparing these figures, we see the effect of filtering the decomposed problem instances. Since the two figures are quite alike, the conclusion that filtering contributes little seems likely to be true.



(a) The relative effect on the number of movements.

(b) The relative effect on the computation time.

Figure 5.14: The relative effect of decomposition and filtering combined with respect to the original problem instances. Negative values mean improvement in plan quality and in computation time.

In Figure 5.14(a), we show the combined relative effect of decomposition and filtering on the plan quality. This figure has a striking resemblance with Figure 5.6, which shows the relative effect of only decomposition. We have seen in Figure 5.11(a) that filtering has little effect on the plan quality. These two comparisons consolidate our suspicion that decomposition is the main contributor to the combined effect, with respect to number of movements.

Of all effects, we have calculated a ratio by dividing the average results of the pre-processed problem instances by the results of the original problem instances. These ratios are summarised in Table 5.1, and are accompanied by a valuation. In this table, we also show a theoretical effect, which is calculated by multiplying the effect of decomposition with the effect of filtering. From this table, we can conclude that filtering has no significant effect on the plan quality, which is conform the suspicion we mentioned earlier in this section. Decomposition has a minor positive effect on the plan quality of FF, and a larger positive effect on the plan quality of YAHSP, which is conform our conclusions in Section 5.3.2. Filtering the decomposed problem instances has no additional effect on the plan quality. Otherwise, the ratio of the combined effect would differ from the theoretical effect.

		Filtering	Decomposition	Combined	Theoretical
YAHSP	ratio	0.98	0.74	0.73	0.72
	valuation	0	+	+	+
FF	ratio	0.99	0.95	0.94	0.94
	valuation	0	0/+	0/+	0/+
SGPlan	ratio	1.02	0.99	1.01	1.01
	valuation	0	0	0	0
CPLEX	ratio	0.99	0.98	0.98	0.97
	valuation	0	0	0	0

Table 5.1: The ratios and valuation of the effect of filtering, the effect of decomposition, the the effect of the combination, and the theoretical combined effect on the plan quality.

Computation Time

We can see the effect of both decomposition and filtering on plan quality when we compare Figure 5.12(a) with Figure 5.12(d). In Figure 5.11(b), we have seen that filtering generally has a positive effect on the computation time. When both effects on computation time of decomposition and filtering are combined, as in Figure 5.14(b), the effect resembles the effect of filtering the most.

In Table 5.2, we use the same ratio as described in the previous section, except that it is calculated with respect to the computation time. The first conclusion we can draw from this table, is that filtering always has a positive effect. At best, decomposition has hardly an effect on the computation time. In case of YAHSP, we have already mentioned in Section 5.3.2, we suspect that this apparent deterioration on computation time is due to the effects of the extremely small computation times we have measured. The combined effect is much more positive than could be expected based on the theoretical effect. Not only are all negative

effects caused by decomposition removed by filtering, the interaction of both effects contributes positively to the combined effect. Looking at Figure 5.13(a) this is not surprising, since this figure shows that filtering the decomposed problem instances is much more effective.

		Filtering	Decomposition	Combined	Theoretical
YAHSP	ratio	0.35	2.41	0.35	0.85
	valuation	+++	–	+++	+
FF	ratio	0.49	0.94	0.26	0.46
	valuation	++	0/+	++++	++
SGPlan	ratio	0.49	1.01	0.23	0.49
	valuation	++	0	++++	++
CPLEX	ratio	0.80	1.19	0.46	0.95
	valuation	+	-	++	0/+

Table 5.2: The ratios and valuation of the effect of filtering, the effect of decomposition, the the effect of the combination, and the theoretical combined effect on computation time.

Summary

The effect of decomposition and filtering combined, resembles the effect of filtering when looking at the criterion of computation time. When looking at the criterion of plan quality, the combined effect resembles the effect of filtering. From Table 5.1, we can also conclude that decomposition and filtering do not interact on plan quality, since no extra effect is visible. In Table 5.2, an extra effect is visible, hence some interaction on computation time between decomposition and filtering exists. This effect is apparent in Figure 5.13(a), since it shows that after decomposition, much more entities can be filtered out. In addition, we can conclude that, with respect to the computation time, decomposition is only beneficial in combination with filtering. All in all, we can conclude that all planners benefit much from decomposition and filtering.

5.3.5 The Effect of Enlarging the Problem Instances

We generated also larger problem instances, of which the details can be found in Appendix B.4. To be able to make fair comparisons when we investigate the change in performance when these larger problem instances are used, we made sure we kept the same proportions between the different variables, i.e., the packages, the trucks, the cities, the locations, and the airplanes. To be able to show the effect of scaling we made also small problem instances with these proportions. After generating these new problem instances, we have repeated the same pre-processing steps, tests and comparisons. Unfortunately, due to boundaries on the maximum number of constants in some of the planners and

also due to limited memory on the computers the maximum size of the problem instances was 180 packages, 126 cities, a total of 126 times 8 is 1008 locations, 162 trucks and 18 airplanes.

In the graphs presented in this section, we will see abrupt endings and erratic curves of lines appear when the problem instances become too large for a planner to solve. For example, in Figure 5.15(d) we see that FF has difficulties solving problem instances of 150 and up. The fact that some results are visible is because we show decomposed problem instances. Hence, the presented results are partial results, since some of the decomposed problem instances could be solved.

Plan Quality

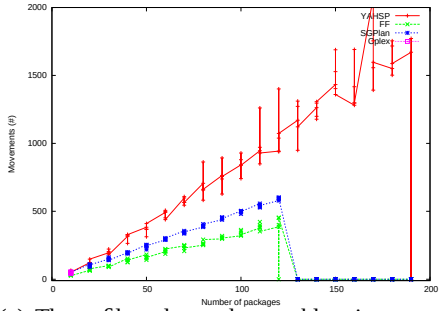
Figure 5.15 shows linear curves, indicating that the problem instances linearly increase in size. Furthermore, we can see by comparing the upper and lower graphs of Figure 5.15 that decomposition greatly improves the plan quality of YAHSP, while the curves of FF and SGPlan stay the same. This is the same as what we already had seen in Section 5.3.2. Also, we can see by comparing the left hand graphs with the right hand graphs in this same figure, that filtering causes the planners FF and SGPlan to be able to solve more problem instances. We can see that the larger the problem instance, the more movements it takes to find a solution.

CPLEX had much difficulties solving even the smallest problem instances (see also Figure 5.16), therefore, we have mostly results of the filtered and decomposed problem instances. In Figure 5.15(d), we can see two measurements of CPLEX, which are clearly below the other curves. These measurements are caused by CPLEX not having solved one of the phases of the decomposed problems. ALNS-PDPTW is still very slow in computing solutions to the problem instances. In case of the unfiltered decomposed problem instances, we have cut-off the computation time after approximately 4000 seconds.

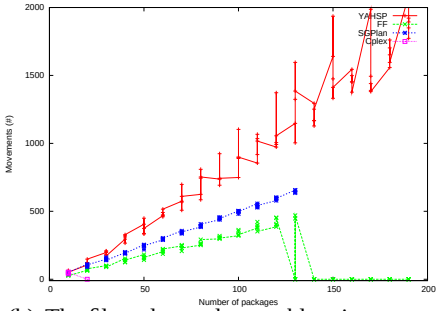
In Figure 5.15(d), we can see that if CPLEX solves a filtered and decomposed problem instance, it can compete with the other planners with respect to the plan quality. Again, ALNS-PDPTW generates plans of the highest quality. The conclusions we have drawn from the experiments we covered earlier in this section seem to hold.

Computation Time

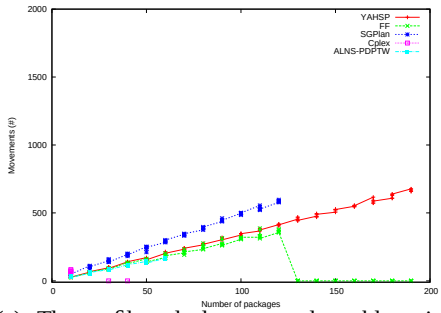
In Figure 5.16, we can see that when the problem instances get larger, it takes, as expected, more time to find a solution. However, FF is extremely sensitive to an increase in size, while YAHSP is not. Indeed, we can see that the curve of FF is steepest on the logarithmic scale. However, not even FF shows a log-linear curve.



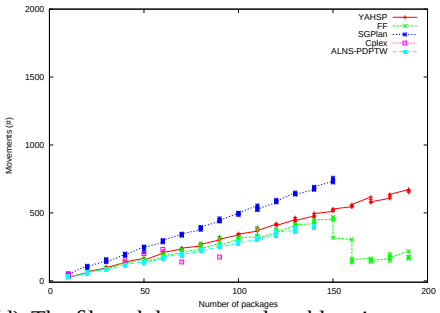
(a) The unfiltered complete problem instances.



(b) The filtered complete problem instances.



(c) The unfiltered decomposed problem instances combined.

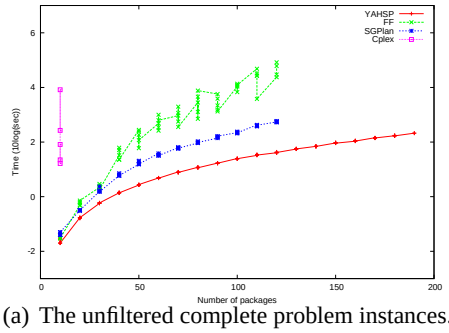


(d) The filtered decomposed problem instances combined.

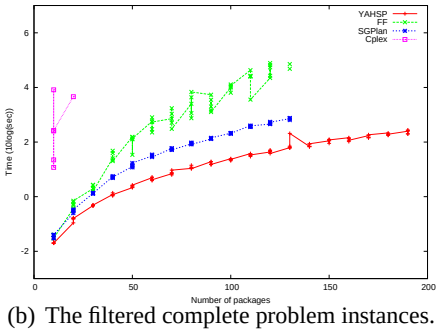
Figure 5.15: The calculated plan quality in the number of move actions from vehicles needed to solve each of the larger problem instance.

The curve of ALNS-PDPTW is positioned highest in the graph. Indeed, it took much computation time to solve the problem instances. However, it seems that the larger the problem instances the more the curve flattens, more than FF or SGPlan. The same effect can be seen with YAHSP. CPLEX solved too few problem instances, hence, we cannot deduce a trend from these figures. If we compare Figure 5.16(a) with Figure 5.16(b), in which we show respectively the unfiltered complete problem instances and the corresponding filtered problem instances, we can see that filtering causes SGPlan to be able to solve a few problem instances more.

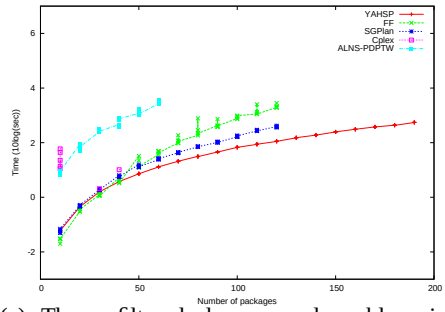
We can think of an explanation why ALNS-PDPTW is this slow. ALNS-PDPTW is based on LNS. As described in Section 3.2.2, LNS selects some nodes to remove and reinsert into the problem to find a better solution. This selection is made on a certain measurement of relatedness. A natural choice is to consider nodes that are geographically close to each other, more related than those further away. Unfortunately, as described in Section B.3, we have “abused” the x- and y-coordinates and replaced them by a city number and a location-number. If these locations would be plotted, a perfect grid would appear, in which every column would represent a city (Figure 5.17). A node in another city might be geographically closer and therefore



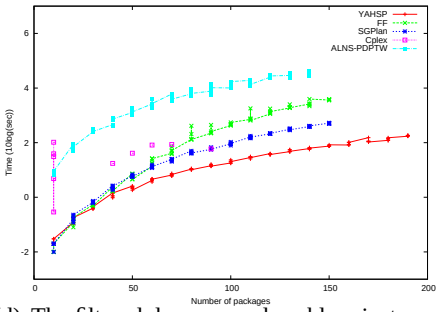
(a) The unfiltered complete problem instances.



(b) The filtered complete problem instances.



(c) The unfiltered decomposed problem instances combined.



(d) The filtered decomposed problem instances combined.

Figure 5.16: The computation time in seconds against a logarithmic scale.

a natural choice for LNS, it is not a good choice in order to find rapidly a solution to the problem we are looking at. It would be interesting to repeat our tests when we can modify the selection-criteria of the LNS algorithms in ALNS-PDPTW to our special case.

Summary

The conclusions we have previously drawn with respect to computation time and plan quality still hold. We expected that the OR solvers would show better performance measured in time when solutions to larger problem instances had to be computed. This seems not to be the case.

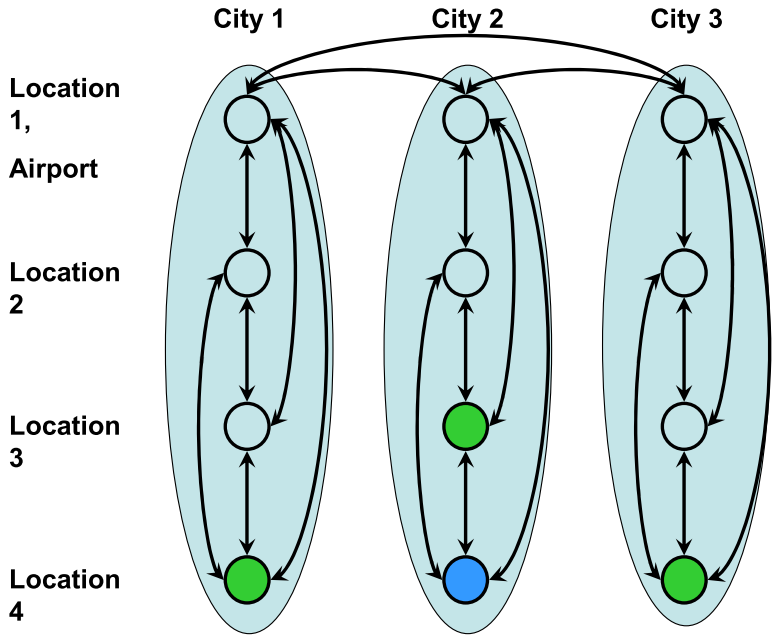


Figure 5.17: The green nodes are geographically close to the blue node. However, the real distance between the blue node and the green node in, for example, city 1 is 3 steps.

Chapter 6

Conclusions and Future Work

In this chapter, we will present the conclusions we have drawn from Section 5.3. We will also discuss the questions raised during our research, which might be subject to further investigation.

6.1 Conclusions

In Section 1.5, we have presented our research questions. In this section, we will answer them, one by one. Before we answer the main research questions, we will go over the sub-questions which address the change of performance due to the pre-processing steps applied to the problem instances. From the answers to those sub-questions, the answers to the main questions can be deduced. We will start with the question which was actually a precondition to our research.

The question which was actually a precondition to our research was: Is it possible to create a VRP variant which is equal to the Logistics Planning Problem? It was widely believed that it should be possible, and indeed it is, as we have shown in Section 3.4.1.

We have applied two pre-processing steps: decomposition and filtering. We have also combined these two pre-processing steps. Decomposition was the first pre-processing step we applied to the problem instances. In Section 5.3.2, we showed that especially the plan quality of YAHSP improves after decomposition. Anticipating on the answers of the main questions, we might say that YAHSP exchanges speed for accuracy, since it is the fastest in solving the problem instances compared with the other planners and solvers used during this research. Decomposition has no clear effect on the computation time.

The second sub-question concerns the change of performance after filtering. We have removed some of the superfluous data from the problem instances. In Section 5.3.3, we have shown that filtering has little effect on the plan quality of

the found solutions by AI planners. The plan quality of the OR solvers does not change at all. With respect to the computation time needed to find a solution to the problem instances, we can conclude that filtering has a large positive effect. This conclusion also holds in case of the in-flight phase, although a smaller amount of packages, cities, and above all, locations are removed from this phase compared to the pre- and post-flight phases.

The third sub-question concerns the change of performance when decomposition and filtering are combined. As shown in Section 5.3.4, the effect of the combined pre-processing steps is the same as the best of each. We have concluded that filtering has mainly an effect on the computation time, and decomposition has at least an effect on YAHSP regarding plan quality. We see the same pattern when we combine decomposition and filtering: as for plan quality, the effect is similar to decomposition, as for the computation time, the effect is similar to filtering.

The last sub-question concerns the behaviour of the planners and solvers when the problem instances are enlarged. We have seen in Section 5.3.5, that YAHSP solved the most problem instances. The other AI planners, used too much memory, which caused them to fail. CPLEX and ALNS-PDPTW also used too much memory, which caused them to run very slow. With respect to the computation time needed to solve the problem instances, the planner FF was most sensitive to enlarging, while YAHSP was the least sensitive. We have also seen that the number of vehicle movements increases linearly with respect to the problem instances. The problem instances are designed such that the increase in size is also linear, while keeping the layout of the problem the same. We assume that, under afore said conditions, the optimal solution will also increase linearly. Under this assumption and previously mentioned conditions, it could be said that the plan quality stays constant when the problem instances are enlarged.

The answer to the question whether VRP approaches and AI approaches show the same sensitivities to the previous pre-processing steps is generally “yes”. Of course, some are more sensitive than others. For example, YAHSP is, with respect to the plan quality, quite sensitive on decomposition. CPLEX benefits obviously from decomposition too, since even the smallest complete problem instances took too long to be solved by CPLEX, which could be solved after decomposition.

Finally we can answer the main questions. Secondly, we can answer the most important question: “Can a VRP approach for solving the Logistics Planning Problem be competitive with an AI planning approach?”, with respect to both computation time and plan quality. It is possible to solve problem instances of the Logistics Planning Problem domain by using (rich) VRP approaches. Although, as far as we have shown, it is not advisable to do so. Both CPLEX and ALNS-PDPTW are very slow. On the other hand, ALNS-PDPTW and CPLEX

generate generally shorter routes and have therefore better plan qualities. YAHSP is fast, but seldom generates the most efficient route.

Summarising, whether a VRP approach can be competitive with an AI approach depends on what measurement is considered most important, computation time or plan quality. For now, a VRP approach cannot compete if effectiveness is measured in computation time, although we have to keep in mind that at least ALNS-PDPTW was not designed for these kind of problems. If effectiveness is only measured in plan quality, ALNS-PDPTW and CPLEX can compete.

6.2 Future Work

The work presented in this thesis, can be continued in several ways. First of all, some modifications can be made to our research, based on some observations we made during the experiments.

The in-flight phase was considered the hardest phase to solve in comparison with the pre- and post-flight phases. Although we constructed the in-flight phase such that all "not airport" locations were present but not reachable, and should therefore not disturb the performance measured in time, it might have caused the planners and solvers to search for a solution in a wrong direction. It would be interesting to see if the unreachable locations indeed "distract" the planners and solvers. To answer this question, the performance, measured in computation time, of planners and solvers on problem instances with and without unreachable locations should be compared.

It might be possible to create a more efficient model of the Logistics Planning Problem. It would be interesting to see whether this would improve the computation time needed to solve the problem instances. Finally, we recommend further research to investigate whether a preprocessing method like GraphPlan would reduce the computation time of CPLEX.

ALNS-PDPTW was not designed for the Logistics Planning Problem. As explained at the end of Section 5.3.5, it would be interesting to repeat our tests when we can modify the selection-criteria of the LNS algorithms in ALNS-PDPTW to our special case. It might improve the computation time needed to solve the problem instances.

It would also be interesting to see how a specially designed solving algorithm for the Logistics Planning Problem would perform. Neither CPLEX nor ALNS-PDPTW are specialized solvers for this problem domain. CPLEX is a general mixed integer problem solver, ALNS-PDPTW is not fit for this problem domain.

Another interesting and natural successor of this research project is to change the direction of this subject. In this research project, we took an AI benchmark and we tried to convert it into an OR problem. That is, to take the Solomon benchmarks used by the OR community and try to rewrite them using PDDL notation. The Solomon benchmarks are commonly used in the OR community to test new solvers on their performance. If these benchmarks can be rewritten in PDDL, the performance of the AI planners can be graded to their measurements.

Bibliography

[1] Ilan Adler and Peter A. Beling. Polynomial algorithms for linear programming over the algebraic numbers. In *STOC '92: Proceedings of the twenty-fourth annual ACM symposium on Theory of computing*, pages 483–494, New York, NY, USA, 1992. ACM.

[2] C. Archetti, R. Mansini, and M.G. Speranza. The split delivery vehicle routing problem with small capacity. *Technical Report 201*, 2001.

[3] S. Baptista, R.C. Oliveira, and E. Zúquete. A period vehicle routing case study. *European Journal of Operational Research*, 139:220–229, 2002. Elsevier.

[4] J.F. Bard, L. Huang, M. Dror, and P. Jaillet. A branch and cut algorithm for the vrp with satellite facilities. *IIE Transactions*, 30:821–834, 1998.

[5] Avrim Blum, Merrick Furst, and John Langford.

[6] Avrim L. Blüm and Merrick L. Furst. Fast planning through planning graph analysis. *Artificial Intelligence*, 90:281300, 1997.

[7] Raymond K. Cheung, Dongsheng Xu, and Yongpei Guan. A solution method for a two-dispatch delivery problem with stochastic customers. *Journal of Mathematical Modelling and Algorithms*, 6(1):87–107, March 2007.

[8] N. Christofides, A. Mingozzi, and P. Toth. Contributions to the quadratic assignment problem. *European Journal of Operational Research 4*, 1980.

[9] N. Christofides, A. Mingozzi, and P. Toth. Exact algorithms for the vehicle routing problem, based on spanning tree and shortest path relaxations. *Mathematical Programming*, 20(1):255–282, December 1981. ISSN 0025-5610 (Print) 1436-4646 (Online).

[10] Dr. Vic Ciesielski. http://www.cs.rmit.edu.au/AI-Search/Courseware/Slides1/, 10th January 2002.

[11] A. I. Coles and A. J. Smith. Generic types and their use in improving the quality of search heuristics. In *Proceedings of the 25th Workshop of the UK Planning and Scheduling Special Interest Group (PlanSIG 2006)*, December 2006.

[12] William Cook. Traveling salesman problem. http://www.tsp.gatech.edu/index.html, Januari 2007.

[13] J.-F. Cordeau, G. Desaulniers, J. Desrosiers, M.M. Solomon, and F. Soumis. Vrp with time windows. *In P. Toth and D. Vigo (eds.): The Vehicle Routing Problem SIAM Monographs on Discrete Mathematics and Applications*, 9:157–193, 2002. Philadelphia, PA,.

[14] Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest, and Clifford Stein. *Introduction to Algorithms, Second Edition*. The MIT Press, September 2001.

[15] Richard Cottle, Ellis Johnson, and Roger Wets. George B. Dantzig (1914-2005). *Notices of the AMS*, 54(3), March 2007.

[16] T. G. Crainic, K. H. Kim, C. Barnhart, and G. Laporte. *Transportation Handbooks in Operations Research and Management Science*, chapter Intermodal transportation. North-Holland (Elsevier), Amsterdam, Netherlands, December 2005.

[17] Soumita Datta. Constraint programming approach to the teams scheduling problem. Master's thesis, Vanderbilt University, May 2006.
[18] Bernabé Dorronsoro Daz. The VRP web. `http://neo.lcc.uma.es/radi-aeb/WebVRP/`.
[19] Keith S. Decker and Victor R. Lesser. Designing a family of coordination algorithms. *In Proceedings of the Thirteenth International Workshop on Distributed Artificial Intelligence (DAI-94)*, 1994.
[20] M. Dror, G. Laporte, and P. Trudeau. Vehicle routing with split deliveries. *Discrete Appl. Math.*, 50:239–254, 1994.
[21] Eithan Ephrati, Martha E. Pollack, and S. Ur. Deriving multi-agent coordination through filtering strategies. *In Proceedings of the Fourteenth International Joint Conference on Artificial Intelligence (IJCAI-95)*, August 1995. Morgan Kaufmann Publishers.
[22] Richard E. Fikes and Nils J. Nilsson. *STRIPS: a new approach to the application of theorem proving to problem solving*, pages 429–446. American Association for Artificial Intelligence, Menlo Park, CA, USA, 1995.
[23] M. L. Fisher and R. Jaikumar. A decomposition algorithm for large-scale vehicle routing. Technical Report 78-11-05, Department of Decision Sciences, University of Pennsylvania, Philadelphia, 1978.
[24] M. L. Fisher and R. Jaikumar. A generalized assignment heuristic for vehicle routing. *Networks*, pages 11:109–124, 1981.
[25] Maria Fox, Derek Long, and Julie Porteous. Abstraction-based action ordering in planning. *International Joint Conference on Artificial Intelligence.*, 2005.
[26] Herman Gehring. Extended solomon's vrptw instances.
[27] E. M. Gerson. On quality of life. *American Sociological Review*, 41, 1976.
[28] M. Ghallab, D. Nau, and P. Traverso. *Automated Planning: theory and practice*. Morgan Kaufmann Publischers, 2004. ISBN 1-55860-856-7.
[29] I. Gribkovskaia, Ø. Halskau, and K. N. Bugge Myklebost. Models for pick-up and deliveries from depots with lasso solutions. *Proceedings of the 13th Annual Conference on Logistics Research - NOFOMA 2001*, 2001.
[30] Ralph P. Grimaldi and Rose-Hulman. *Discrete and Combinatorial Mathematics; An Applied Introduction*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 4th edition edition, 1985.
[31] Hadjar, Marcotte, and Soumis. A branch-and-cut algorithm for the multiple depot vehicle scheduling problem. *Operations Research*, 54(1):130149, 2006.
[32] P. E. Hart, N. J. Nilsson, and B. Raphael. A formal basis for the heuristic determination of minimum cost paths in graphs. *IEEE Trans. on Systems Science and Cybernetics*, 4(2):100–107, July 1968.
[33] Dr. Geir Hasle. Benchmarks - vehicle routing and travelling salesperson problems. `http://www.top.sintef.no/`. SINTEF.
[34] Malte Helmert. On the complexity of planning in transportation and manipulation domains. Master's thesis, Albert-Ludwigs-Universität Freiburg, March 2001.
[35] Malte Helmert. On the complexity of planning in transportation domains. *Proceedings of the Sixth European Conference on Planning (ECP 2001)*, 2001.
[36] Malte Helmert. Complexity results for standard benchmark domains in planning. *Artificial Intelligence*, 2003.
[37] C. Hjorring. *The Vehicle Routing Problem and Local Search Metaheuristics*. PhD thesis, The University of Auckland, 1995.
[38] Chih-Wei Hsu, Benjamin W. Wah, Ruoyun Huang, and Yixin Chen. New features in sgplan for handling preferences and constraints in pddl3.0*. *ICAPS 2008*, 2008.

[39] ILOG. Ilog-opl-cplex developement studio. http://www.ilog.com/products/oplstudio/.

[40] C. Jacobs-Blecha and M. Goetschalckx. *The vehicle routing problem with backhauls: properties and solution algorithms*. Atlanta; Georgia, Januari 1992. Presented at the National Transportation Research Board.

[41] N.R. Jennings. Coordination techniques for distributed artificial intelligence. *Foundations of Distributed Artificial Intelligence*, 1996.

[42] Narendra Karmarkar. A new polynomial-time algorithm for linear programming. *Combinatorica*, 4(4):373–396, 1984.

[43] G. Laporte and F.V. Louveaux. Solving stochastic routing problems with the integer l-shaped method. *In T.G. Crainic and G. Laporte (eds.): Fleet Management and Logistics*, pages 159–167, 1998.

[44] E.L. Lawler, J.K. Lenstra, A.H.G. Rinnooy Kan, and D.B. Shmoys. *The Traveling Salesman Problem. A guided tour of combinatorial optimization*. John Wiley and Sons, 1985.

[45] David C Lay. *Linear Algebra and Its Applications*. Addison Wesley, second edition, 1997.

[46] Thomas W. Malone and Kevin Crowston. The interdisciplinary study of coordination. *ACM Computing Surveys*, 26(1), March 1994.

[47] Drew McDermott. AIPS98 planning competition results. ftp://ftp.cs.yale.ed/pub/mcdermott/aipscomp-results.html.

[48] Drew McDermott. The 1998 AI planning systems competition. *AI Magazine*, 2000.

[49] Atif M. Memon, Martha E. Pollack, and Mary Lou Soffa. Hierarchical GUI test case generation using automated planning. *Software Engineering*, 27(2):144–155, 2001.

[50] Bernard M. Moret. *The Theory of Computation*. Addison Wesley, 1998. ISBN 0-201-25828-5.

[51] Steven Michael Morris. *Truck Dispatching and Fixed Driver Rest Locations*. PhD thesis, Georgia Institute of Technology, December 2007.

[52] H. S. Nwana, L. Lee, and N. R. Jennings. Coordination in software agent systems. *BT Technology Journal*, 14(4), October 1996.

[53] http://www.cs.toronto.edu/aips2000/systemdescriptions.html.

[54] H.N. Post. Transport, routing- en schedulingproblemen. Lecturenotes, May 2006.

[55] T.K. Ralphs, L. Kopman, W.R. Pulleyblank, and L.E. Trotter Jr. On the capacitated vehicle routing problem. *Math. Program.*, pages 343–359, 2003. Springer-Verlag Preliminary version.

[56] G. Righini. Approximation algorithms for the vehicle routing problem with pick-up and delivery. Note del Polo - Ricerca 33, Polo Didattico e di Ricerca di Crema, Università degli Studi di Milano, 2000.

[57] Stefan Røpke and David Pisinger. An adaptive large neighborhood search heuristic for the pickup and delivery problem with time windows. Technical report, DIKU - Department of Computer Science, University of Copenhagen, Universitetsparken 1, DK-2100 Copenhagen, Denmark., 8th August 2005.

[58] Stefan Røpke and David Pisinger. A general heuristic for vehicle routing problems. Technical report, DIKU - Department of Computer Science, University of Copenhagen, Universitetsparken 1, DK-2100 Copenhagen, Denmark., 25th February 2005.

[59] Stefan Røpke and David Pisinger. A unified heuristic for a large class of vehicle routing problems with backhauls. *European Journal of Operational Research*, 171(3):750–775, 2006.

[60] Stuart Russel and Peter Norvig. *Artificial Intelligence: a modern approach*. Prentice Hall International Editions, Upper Saddle River, NJ, USA, 1995. ISBN 0-13-360124-2.
[61] A. Schrijver. A course on combinatorial optimization. Lecture Notes, 1994.
[62] P. Shaw. Using constraint programming and local search methods to solve vehicle routing problems. *Proceedings of the Fourth International Conference on Principles and Practice of Constraint Programming (CP '98)*, pages 417–431, 1998. M. Maher and J.-F. Puget (eds.), Springer-Verlag.
[63] Yoav Shoham and Moshe Tennenholtz. On social laws for artificial agent societies: Off-line design. *Artificial Intelligence*, 1995.
[64] M. Sigurd, D. Pisinger, and M. Sig. The pickup and delivery problem with time windows and precedences. Technical report, University of Copenhagen, 2000.
[65] Michael Sipser. *Introduction to the Theory of Computation*. PWS Publishing Company, 1997. ISBN 0-534-94728-X.
[66] Jan Renze Steenhuisen, Cees Witteveen, and Yingqian Zhang. Plan-coordination mechanisms and the price of autonomy. In *Proceedings of the 8th Workshop on Computational Logic in Multi-Agent Systems (CLIMA)*, Berlin, Germany, sep 2007.
[67] Adriaan ter Mors. Coordinating autonomous planning agents. Master's thesis, Delft University of Technology, 2004.
[68] Adriaan ter Mors. Coordination mechanisms for autonomous agents: A survey of the multi-agent coordination literature. Literature study, April 2004.
[69] A. W. ter Mors, J. M. Valk, and C. Witteveen. Decomposition in task-based multi-agent planning systems. In Kenneth N. Brown, editor, *Proceedings of the 23rd Workshop of the UK Planning and Scheduling Special Interest Group (PlanSIG)*, pages 169–180. Cork Constraint Computation Centre, dec 2004.
[70] A. W. ter Mors, J. M. Valk, and C. Witteveen. Task coordination and decomposition in multi-actor planning systems. In G. Sutschet, editor, *Proceedings of the Workshop on Software-Agents in Information Systems and Industrial Applications (SAISIA)*, pages 83 – 94, Karlsruhe, february 2006. Fraunhofer Institut Informations- und Datenverarbeitung IITB, Fraunhofer IRB Verlag.
[71] unknown. Strips. `http://en.wikipedia.org/wiki/STRIPS`.
[72] Jeroen Valk. *Coordination in Multi-Agent Systems*. PhD thesis, Delft University of Technology, 2005.
[73] Vincent Vidal. A lookahead strategy for solving large planning problems. In *IJCAI*, pages 1524–1525, 2003.
[74] Daniel S. Weld. An introduction to least commitment planning. *AI Magazine*, 15(4):27–61, 1994.

Appendix A

The Basic VRP and Variants

Since the introduction of the basic Vehicle Routing Problem (VRP) in 1959 by Dantzig and Ramser [15], several different notations for the same problem have been made, which are all useful in their own way. That is, every notation has a solving-method that uses the structure by which the problem is written down. Below, we will present the different notations of this NP-hard problem [44, 54].

A.1 Notations of the Basic VRP

In the notations below, variable c_{ij} is the costs for moving from customer i to customer j, there are n customers in total. Variable k is a truck, there are m trucks in the problem. Variable Q_k is the capacity of truck k and q_i is the demand of customer i.

Definition 1 ***VRP notation 1 by Fisher and Jaikumar, 1978, 1981 [23, 24]***. *Let*

$$x_{ijk} = \begin{cases} 1 & \textit{if truck } k \textit{ visits customer } j \textit{ immediately after customer } i, \\ 0 & \textit{else,} \end{cases}$$

and

$$y_{ik} = \begin{cases} 1 & \textit{if truck } k \textit{ visits customer } i, \\ 0 & \textit{else.} \end{cases}$$

Minimise

$$\sum_{i,j} c_{ij} \sum_{k} x_{ijk} \tag{A.1}$$

under following preconditions.

$$\sum_{k} y_{ik} = \begin{cases} 1 & \textit{if } i = 2, 3, \ldots, n, \\ 0 & \textit{if } i = m. \end{cases} \tag{A.2}$$

$$\sum_i q_i y_{ik} \leq Q_k, k = 1, 2, \ldots, m. \tag{A.3}$$

$$\sum_j x_{ijk} = \sum_j x_{jik} = y_{ik}, 1 \leq i \leq n, 1 \leq k \leq m. \tag{A.4}$$

$$\sum_j x_{ijk} \leq \mid S \mid -1, S \subseteq \{2, 3, \ldots, n\}, 1 \leq k \leq m \tag{A.5}$$

$$y_{ik} \in \{0, 1\}, 1 \leq i \leq n, 1 \leq k \leq m. \tag{A.6}$$

$$x_{ijk} \in \{0, 1\}, 1 \leq i, j \leq n, 1 \leq k \leq m. \tag{A.7}$$

Elucidation of the above: Constraint A.2 guarantees that every customer is assigned to exactly one truck, and that the route of every truck contains the depot. Constraint A.3 negotiates the limited capacity of the trucks. The constraints A.4 ensure that every truck visits and leaves every to him assigned customer. Finally, constraint A.5 prevents sub routes in the TSP for truck k.

Definition 2 ***VRP notation 2 by Christofides, Mingozzi and Toth, 1980 [8]***.
We assume that all admissible routes of truck 1, the truck with maximum capacity, are known and numbered from $r = 1$ to $r = \hat{r}$. Let M_r be the set of customers in route r, and d_r the costs of this route. Let N_i be a subset of previous routes that contain customer i, i.e. $N_i = \{r : i \in M_r$. For every r, $1 \leq r \leq \hat{r}$, let K_r be the with r corresponding cargo, i.e. $K_r = \sum_{i \in M_r} q_i$. We assume that routes are numbered such that $K_1 \geq K_2 \geq \cdots \geq K_{\hat{r}}$. For every truck k let r_k be the smallest value of r, such that $K_r \leq Q_k$. That implies that truck k has sufficient capacity for route r iff[1] *$r \geq r_k$. It is obvious that $r_1 = 1$. Let $r_{m+1} = \hat{r} + 1$. Now let*

$$y_r = \begin{cases} 1 & \textit{if route } r \textit{ is used} \\ 0 & \textit{else.} \end{cases}$$

The basic VRP then boils down to minimising

$$\sum_{r=1}^{\hat{r}} d_r y_r \tag{A.8}$$

[1] We use the word 'iff' as short for 'if and only if'.

under following preconditions.

$$\sum_{r \in N_i} y_r = 1, 2 \leq i \leq n. \tag{A.9}$$

$$\sum_{r=1}^{r_{k+1}-1} y_r \leq k, r_k \neq r_{k+1}, 1 \leq k \leq m. \tag{A.10}$$

$$\sum_{r=1}^{\hat{r}} y_r = m. \tag{A.11}$$

$$y_r \in \{0, 1\}, 1 \leq r \leq \hat{r}. \tag{A.12}$$

Elucidation of the above: Constraint A.9 ensures that every customer is visited only once. Constraint A.11 guarantees that the number of routes is equal to the number of trucks. Finally, constraint A.10 ensures that no truck is assigned a route that exceeds its capacity.

Both previous two mathematical notations are integer programming problems. The next notation is more dynamic.

Definition 3 ***VRP notation 3 by Christofides, Mingozzi and Toth, 1981 [9]***
Let $N = \{2, 3, \ldots, n\}$ be the set of all customers. For every $T \subseteq N$ let $f(k, T)$be the minimal costs for serving the customers in T by the first k trucks. Further, let $v(T)$ be the minimal costs for solving the TSP defined by the depot, the customers in T and $q(T) = \sum_{i \in T} q_i$. The dynamic programming recursion formula is initialised for $k = 1$ by $f(1, T) = v(T), \forall T$, and for $k \geq 2$ further defined by:

$$f(k, T) = \min_{S \subset T}(f(k-1, T-S) + v(S)) \tag{A.13}$$

under preconditions:

$$q(T) - \sum_{k=1}^{k-1} Q_k \leq q(S) \leq Q_k, \tag{A.14}$$

$$\frac{1}{m-k} q(N-T) \leq q(S) \leq \frac{1}{k} q(T), \tag{A.15}$$

for which the left hand inequality in (14) is only valid if $k \neq m$. Further, the sets T have to satisfy:

$$q(N) - \sum_{k=k+1}^{m} Q_k \leq q(T) \leq \sum_{k=1}^{k} Q_k. \tag{A.16}$$

Elucidation of the above: Constraint A.16 is in the second inequality dependent on constraint A.14. Constraint A.13: S is the set of customers that is assigned to truck k in such manner that the resulting costs are minimal. The constraints A.14, A.15, and A.16 ensure that $f(T)$ and $v(T)$ are not calculated for situations that can only lead to exceeding of one or more capacities. The right hand inequality in A.14 ensures that truck k is not overloaded, while the left hand inequality guaranties that the first $k-1$ trucks are not overloaded. Constraint A.15 expresses that the load of truck k is at least equal to the average load of the first k trucks and at least equal to the average load of the last $m-k$ trucks, i.e., the average load of the first k trucks $\geq q(S)$ = the load of truck $k \geq$ the average load of the last $k-m$ trucks. It only serves to limit the number of subsets S. Finally, the right hand inequality in A.16 implies that the first k trucks together can serve the customers in T, and the left hand inequality implies that the remaining customers can be served by the remaining $m-k$ trucks. It is assumed that the first k trucks together serve the customers in T.

A.2 VRP Variants Overview

The basic VRP can be extended in several ways. Below we have some examples as found in [18]. The list presented here is far from conclusive. Many other problems can be found in the literature, although perhaps under different names. Like the basic VRP, some problems are also known by different names. Also, some names represent different problems in the literature. As an example, the pick-up and delivery problem, which is also known as VRP with pick-up and delivery. It is also a name under which the VRP with backhauls can be found. In some pick-up and delivery problems it is allowed to interchange goods between customers, some do not.

- **Capacitated VRP** (CVRP) [55]: CVRP is a VRP in which the travelling costs will be minimized while having a fixed number vehicles with uniform capacity which must serve known customer demands. The demands are a certain amount of uniform goods are which are initially located at a depot. Summarising, the additional constraint that every vehicles must have uniform capacity is the difference between a CVRP and a VRP.

- **VRP with time windows** (VRPTW) [13]: The additional restriction which is the difference between a VRP and a VRPTW is that in VRPTW a time window is associated with each customer $v \in V$, defining an interval $[e_v, e_l]$ in which the customer has to be served. The interval $[e_0, l_0]$ at the depot is called the scheduling horizon or the time window.

- **Multiple Depot VRP** (MDVRP) [37]: It is not uncommon that a company has several distribution centres or depots from which it serves its customers. If the customers are clustered around depots, it is possible to model this

distribution problem as a set of basic VRP's. However, if it is less clear which customers are served from which depot, a Multiple Depot Vehicle Routing Problem should be solved. A number of vehicles is based at each depot. Each vehicle originates from one depot, services the customers assigned to that depot, and returns to the same depot. Again, serving all customers while minimising the number of vehicles and travel costs, is the objective.

- **Periodic VRP** (PVRP) [3]: In classical VRP's, the period for which a planning is made, is usually a single day. The PVRP extends this planning period to M days.

- **Split Delivery VRP** (SDVRP) [2]: In the classical VRP's a demand from a customer is entirely met by one vehicle, no partial deliveries are allowed. SDVRP is a relaxation of the VRP in which several vehicles can serve the same customer if it reduces overall costs. This relaxation facilitates the customers to have demands equal to or even larger than tha capacity of a vehicle. In [20] it is concluded that it is more difficult to obtain the optimal solution in the SDVRP than in the VRP.

- **Stochastic VRP** (SVRP) [43]: SVRP are VRP's in which one or more random components exist. Three examples of SVRP are: *Stochastic customers*: Each customer v_i is part of the problem with a probability p_i, and absent otherwise. *Stochastic demands*: Each customer has a random demand q_i. *Stochastic times*: Service times δ_i and travel times t_{ij} are randomly determined. In SVRP, two stages are made for getting a solution. A first solution is determined before knowing the values of the random variables. In a second stage, when the values of the random variables are known, a corrective action can be taken.

- **VRP with Pick-up and Deliveries** (VRPPD) [29, 56]: In the VRPPD, it is possible for customers to return some goods. Of course, returning goods may not cause the capacity of the vehicles to exceed. This restriction make the planning problem more difficult, it can lead to vehicles capacities being used sub-optimal, longer travelling distances or that more vehicles are needed. Therefore, it is common that restricted situations are considered where all delivery demands start from the depot and all pick-up demands shall be brought back to the depot, so no interchanges of goods between the customers are possible. Another alternative is relaxing the restriction that all customers have to be visited exactly once. Another usual simplification is to consider that every vehicle must deliver all the commodities before picking up any goods.

- **VRP with Backhauls** (VRPB) [40]: The VRPB is a VRP in which customers can demand or return some commodities. So in VRPB it's needed to take into account that the goods that customers return to the deliver vehicle must fit into it. The critical assumption in that all deliveries must be made on

each route before any pick-ups can be made. This arises from the fact that the vehicles are rear-loaded, and rearrangement of the loads on the tracks at the delivery points is not deemed economical or feasible. The quantities to be delivered and picked up are fixed and known in advance. VRPB is similar to VRPPD with the restriction that in the case of VRPB all deliveries for each route must be completed before any pick-ups are made.

- **Vehicle Routing Problem with Satellite Facilities** (VRPSF) [4]: An important aspect of the VRP that has been largely overlooked is the use of satellite facilities to replenish vehicles during a route. When possible, satellite replenishment allows the drivers to continue making deliveries until the close of their shift without necessarily returning to the central depot. This situation arises primarily in the distribution of fuels and certain retail items.

Combinations of the above are also possible. Some examples of those combinations are Split Delivery VRP with Time Windows (SDVRPTW), Capacitated VRP with Pick-up and Deliveries and Time Windows (CVRPPDTW), Periodic VRP with Time Windows (PVRPTW) and Multiple Depot VRP with Time Windows (MDVRPTW).

A.3 The RPDPTW Model

In the RPDPTW there are a number of requests to be carried out by a fixed set of vehicles. Each request consists of picking up a quantity of goods at one location and delivering it to another location. The objective of the problem is to find a feasible set of routes for the vehicles so that all requests are serviced, and such that the overall travel distance is minimized. A feasible route of a vehicle should start at a given location, service a number of requests such that the capacity of the vehicle is not exceeded, and finally end at a given location. A pick-up or delivery should take place within a given time window. Furthermore, each request has an associated pick-up precedence number, and a delivery precedence number. A vehicle must visit the locations in non-decreasing order of precedences (see e.g. Sigurd et al. [64] for more information on other uses of precedence constraints). Since not all vehicles may be able to service all requests (e.g. due to their physical size or the absence of some cooling compartments) we need to ensure that every request is serviced by a given subset of vehicles. Between any two locations we have an associated, non-negative distance and travel time. It is assumed that travel times satisfy the triangle inequality. This assumption implies that any removal of requests from a feasible route will keep the route feasible with respect to the imposed time windows [58].

A.3.1 A Formal Description of RPDPTW

A more formal description of the RPDPTW is as follows [57]: A problem instance of the pick-up and delivery problem contains n requests and m vehicles. The

problem is defined on a graph, $P = \{1, \ldots, n\}$ is the set of pick-up nodes, $D = \{n+1, \ldots, 2n\}$ is the set of delivery nodes. Request i is represented by nodes i and $i+n$. K is the set of all vehicles, $|K| = m$. One vehicle might not be able to serve all requests, as an example a vehicle like an airplane cannot land but on an airport. K_i is the set of vehicles that are able to serve request i. $P_k \subseteq P$ is the set of pick-ups and $D_k \subseteq D$ is the set of deliveries that can be served by vehicle k. Thus for all i and $k : k \in K_i \Leftrightarrow i \in P_k \wedge i \in D_k$. Requests where $K_i \neq K$ are called *special requests*. Define $N = P \cup D$ and $N_k = P_k \cup D_k$. Let $\tau_k = 2n + k, k \in K$ and $\tau_0' k = 2n + m + k, k \in K$ be the nodes that represents the start and end depot, respectively, of vehicle k. The graph $G = (V, A)$ consists of the nodes $V = N \cup \{\tau_1, \ldots, \tau_m\} \cup \{\tau_1', \ldots, \tau_m'\}$ and the arcs $A = V \times V$. For each vehicle we have a sub-graph $G_k = (V_k, A_k)$, where $V_k = N_k \cup \{\tau_k\} \cup \{\tau_k'\}$ and $A_k = V_k \times V_k$. For each edge $(i, j) \in A$ we assign a distance d_{ij} and a travel time t_{ij}. Distances and times are assumed to be non-negative: $d_{ij} \geq 0$, $t_{ij} \geq 0$. It is also assumed that the times satisfy the triangle inequality: $t_{ij} \leq t_{il} + t_{lj}$ for all $i, j, l \in V$. To simplify modelling the elimination of sub tours and the pick-up-before-delivery constraint ,it is also assumed that $t_{i,n+i} + s_i > 0$.

Each node $i \in V$ has a service time s_i, which represents the time needed for loading and unloading, and a time window $[a_i, b_i]$ in which the visit should take place. A vehicle is allowed to arrive early at location, i.e., before the start of the time window, it has to wait until the start of the time window before the visit can be performed. For each node $i \in N$, l_i is the amount of goods that must be loaded onto the vehicle at the particular node, $l_i \geq 0$ for $i \in P$ and $l_i = -l_{i-n}$ for $i \in D$. The capacity of vehicle $k \in K$ is denoted C_k.

Four types of decision variables are used in the mathematical model of Røpke [57]. x_{ijk}, $i, j \in V$, $k \in K$ is a binary variable which is one if the edge between node i and node j is used by vehicle k and zero otherwise. S_{ik}, $i \in V$, $k \in K$ is a non-negative integer that indicates when vehicle k starts the service at location i. L_{ik}, $i \in V$, $k \in K$ is a non-negative integer that is an upper bound on the amount of goods on vehicle k after servicing node i. S_{ik} and L_{ik} are only well-defined when vehicle k actually visits node i. Finally z_i, $i \in P$ is a binary variable that indicates if request i is placed in the request bank. The variable is one if the request is placed in the request bank and zero otherwise.

A.3.2 The Mathematical Model of RPDPTW

The following mathematical model is, unlike ours in Section 3.4.1, not as integer linear program. The authors considered it unnecessary since they solve the problem instances heuristically [57]. The mathematical model is:

$$\min \alpha \sum_{k \in K} \sum_{i,j \in A} d_{ij} x_{ijk} + \beta \sum_{k \in K} \left(S_{\tau_k', k} - a_{\tau_k} \right) + \gamma \sum_{i \in P} z_i \tag{A.17}$$

Subject to:

$$\sum_{k \in K_i} \sum_{j \in N_k} x_{ijk} + z_i = 1 \qquad \forall i \in P \tag{A.18}$$

$$\sum_{j \in V_k} x_{ijk} - \sum_{j \in V_k} x_{j,n+i,k} = 0 \qquad \forall k \in K, \forall i \in P_k \tag{A.19}$$

$$\sum_{j \in P_k \cup \{\tau'_k\}} x_{\tau_k,j,k} = 1 \qquad \forall k \in K \tag{A.20}$$

$$\sum_{i \in D_k \cup \{\tau_k\}} x_{i,\tau'_k,k} = 1 \qquad \forall k \in K \tag{A.21}$$

$$\sum_{i \in V_k} x_{ijk} - \sum_{i \in V_k} x_{jik} = 0 \qquad \forall k \in K, \forall j \in N_k \tag{A.22}$$

$$x_{ijk} = 1 \Rightarrow S_{ik} + s_i + t_{ij} \leq S_{jk} \qquad \forall k \in K, \forall (i,j) \in A_k \tag{A.23}$$

$$a_i \leq S_{ik} \leq b_i \qquad \forall k \in K, \forall i \in V_k \tag{A.24}$$

$$S_{ik} \leq S_{n+i,k} \qquad \forall k \in K, \forall i \in P_k \tag{A.25}$$

$$x_{ijk} = 1 \Rightarrow L_{ik} + l_j \leq L_{jk} \qquad \forall k \in K, \forall (i,j) \in A_k \tag{A.26}$$

$$L_{ik} \leq C_k \qquad \forall k \in K, \forall i \in V_k \tag{A.27}$$

$$L_{\tau_k k} = L_{\tau'_k k} = 0 \qquad \forall k \in K \tag{A.28}$$

$$x_{ijk} \in \{0,1\} \qquad \forall k \in K, \forall (i,j) \in A_k \tag{A.29}$$

$$z_i \in \{0,1\} \qquad \forall i \in P \tag{A.30}$$

$$S_{ik} \geq 0 \qquad \forall k \in K, \forall i \in V_k \tag{A.31}$$

$$L_{ik} \geq 0 \qquad \forall k \in K, \forall i \in V_k \tag{A.32}$$

The objective function minimizes the weighted sum of the distance travelled (α), the sum of the time spent by each vehicle (β), and the number of requests not scheduled (γ).

Constraint A.18 ensures that each pick-up location is visited or that the corresponding request is placed in the request bank. Constraint A.19 ensures that the delivery location is visited if the pick-up location is visited and that the visit is performed by the same vehicle. Constraints A.20 and A.21 ensure that a vehicle leaves every start terminal and a vehicle enters every end terminal. Together with constraint A.22 this ensures that consecutive paths between τ_k and τ'_k are formed for each $k \in K$.

Constraints A.23, A.24 ensure that S_{ik} is set correctly along the paths and that the time windows are obeyed. These constraints also make sub tours impossible. Constraint A.25 ensures that each pick-up occur before the corresponding delivery. Constraints A.26, A.27 and A.28 ensure that the load variable is set correctly along the paths and that the capacity constraints of the vehicles are respected.

Using this general model, several rich VRP's can be described. We refer to [58] for detailed descriptions of these transformations.

Appendix B

Pre-processing and Enlarging the Problem Instances

In this chapter, we will describe the problem instances in detail. We will describe on which criteria the filtering of the problem instances was done and we will explain why some further filtering is left out. Furthermore, we will explain how we decompose the problem instances. We will end this chapter with describing the way we have enlarged the problems after the description of the transformation of an original planning problem instance to a VRP problem instance.

As mentioned in Section 5.1, will use two VRP solvers, CPLEX and ALNS-PDPTW. In order to do experiments on CPLEX, we had to make a mathematical model of the Logistics Planning Problem. This was described in Section 3.4.1, along with how to modify the data of instance of Logistics Planning Problem accordingly. Because of this model we can compare a general LP optimizer with general planners.

Before we could compare the performance of the ALNS-PDPTW solver with the planners, we needed to modify the problem instances. ALNS-PDPTW cannot deal with packages that have to be served by several vehicles to get to their end-locations. Therefore, ALNS-PDPTW can only be compared with the planners on the coordinated or the, with the Arbiter-0 method, decomposed problems. Also, the ALNS-PDPTW solver requires the input in a different format, and above that, we found out that much data in the original problem instances was superfluous. In this chapter, we will discuss how we implemented all three file conversions.

B.1 Filtering the Problem Instances

The problem instances are described in STRIPS, which is explained in Section 2.3. When we looked into the STRIPS problem instances, it was noted that much data was superfluous. For example, ten packages were generated, but only two had to

reach a certain destination. The consequence of that was that eight packages could be deleted. This meant also that some trucks did not have to drive, which means that many locations are not visited. If not even a airplane starts at the airport, the whole city can be dismissed. In this section we will discuss the conditions on which data was filtered out, but first we will introduce the a problem instance in STRIPS format.

Because it is easier to explain the filtering process from an example, we will present a small problem in STRIPS, see Table B.1. The file starts with a section in which all used variables are gathered. The type of the variable is established in the `init` section, which is the second section. In the `init` section, the relations between the variables are described also. Every line can be read as a kind of normal sentence if the second word is read before the first, and the third at last. For example: `city14-3` is `in-city city14`. Finally, there is a `goal` section, which states the destinations of the packages.

B.1.1 Filtering of Unused Packages

As can be seen in the example, two packages, `package1` and `package2` do not occur in the `goal` section. Therefore, there is no need to keep them in the input file. So every entry referring to these variables can be deleted. In some rare occasions the destination of the package is the same as the start location. These packages can be removed also.

B.1.2 Filtering of Unused Cities and Locations

If we look at the cities where the packages need to be picked up or delivered, we see that the cities 2, 3, 6, 7 and 14 are not used. However, we cannot remove every entry of that city yet. At some city airports, there is an airplane, in the example at `city1`, `city10`, and two at `city11`. Since we do not want to remove airplanes, we only remove the cities that do not occur in the start and end locations of packages and the start locations of airplanes. In the example, the cities 2, 3, 6, 7, and 14 plus all references to them can be removed, because no airplanes start from the airport of those cities. Of course, when the existence of the city only depends on the fact that an airplane starts at its airport, it is not needed to keep all other locations in that city.

This method of removing cities and locations is quite rude and certainly does not remove all superfluous locations. We will get back to the choice of not filtering any further in Section B.1.4.

B.1.3 Filtering of Unused Trucks

To find the trucks that can be removed, it is better to find the trucks that are needed first. In principle, every package needs two trucks, one for preflight transportation and one for post-flight transportation. Because of the layout of the city being a complete graph with equal distances, packages that start at the same location can share the same truck, even if they do not have the same destination. The number of trucks is in this case minimized, while the travelling distance stays the same. When in the future, for example, euclidean distances will be used, packages cannot this easily be combined in one truck, so then we will still need a truck per package.

If a package starts at the airport and has to go to another city, it does not need a truck in the starting city. However, if its destination is within the same city, it does need one. Of course, if a package is delivered at an airport after the in-flight phase and that airport is its destination, no truck is needed for this package in the post-flight phase.

In order to minimize the number of trucks and the travelling distance, we decided to add to the post-flight transportation phase the packages that have their destination within the same city as their start location and this start location is the airport. The other packages that need only within-city transportation are added to the preflight transportation phase. The idea behind it is that in the preflight phase, a truck might be present at that location and therefore most suitable to pick-up this package. If, on the other hand, the package would be served in the post-flight phase, the truck might have been moved to the airport and should go back to pick-up the package, which costs one movement extra. If a package is already at the airport, it is often the case that a truck needs to go to the airport in the preflight phase, to deliver a package. It would be a waste of movements and trucks if that truck would not wait at the airport until other packages would arrive that need post-flight transportation. Then it can perform all deliveries in one go.

After using these filtering rules we end up with the problem instances that is shown in Table B.2. As can be seen, the cities 2, 3, 6, 7, and 14 are removed, as are the packages 1 and 2. Since in this example only one truck exist per city, only the trucks belonging to the unused cities are removed.

B.1.4 Further Filtering

Some further filtering can be done. It was chosen not to, because most of the data that can be removed now are based on the properties of a complete graph with equal distances between nodes. An example of this is removing all locations other than the ones used. Because all routes are of the same length, it is best to use direct routes. However, using different weights for each path, it might be better to take a route via another location (Figure B.1).

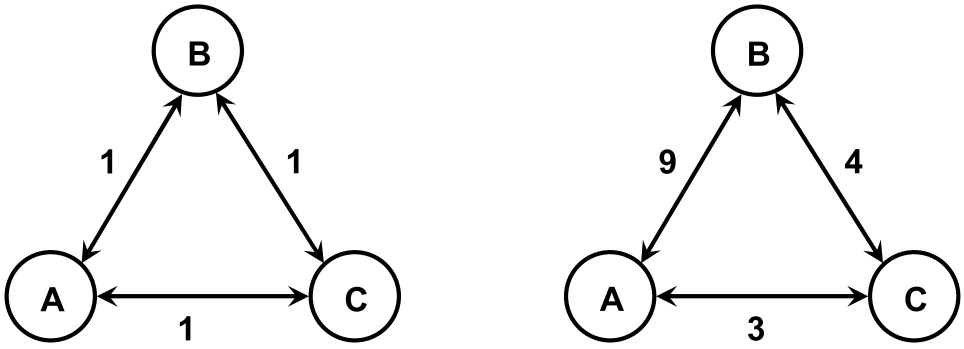


Figure B.1: If all distances between locations are uniform, than the most efficient way to travel from location A to location B is directly. If the distances between locations are not uniform, it might be beneficial to travel via C.

B.2 Decomposing the Problem Instances

For every original input file, we created three smaller problem instances, according to the Arbiter-0 method [72], or more generally, the depth-partitioning algorithm of Steenhuisen [66]. Those methods are explained in Section 4.3. We only made a slight adaptation of this method, which cannot generally be used, but is specific for the Logistics Planning Problem domain.

B.2.1 Adaptation of the Arbiter-0 Method

The Arbiter-0 method, proposed by Valk [72], generalised in the depth-partitioning algorithm of Steenhuisen [66], is a pre-planning coordination technique which, in short, orders the tasks such that all tasks for which the preconditions are met, are carried out first. We differ a bit from the Arbiter-0 method. If we follow the Arbiter-0 method to the letter, all tasks that can be performed, should be performed. However, some tasks are independent from others. In case of the Logistics Planning Problem, those are the tasks of transporting a package which start and end locations are within the same city. According to the Arbiter-0 method, these task are performed in the preflight phase. Because these tasks are independent, there is actually no reason why they can not be performed in the post-flight phase. Therefore, we chose to put in the post-flight phase the tasks starting at the airport and going a location within the same city, for the reasons already mentioned in Section B.1.3.

B.2.2 The Implementation of Arbiter-0

To explain how the Arbiter-0 method was implemented, we will go further on the example of Table B.1. There are seven packages left. Let us take package number 9 as an example of the general case. Its start location is `city4-2` and its goal location is `city11-2`. These locations are not of the same city, so it has to

go by airplane. Therefore, the package has to visit the airport of `city4` which is at location `city4-3` and that of `city11` which is at location `city11-3`. From this we can construct the start and end locations of the package in each phase. In the preflight phase it starts at location `city4-2` and ends at location `city4-3`. In the in-flight phase it starts at location `city4-3` and ends at location `city11-3`. In the post-flight phase it starts at location `city11-3` and ends at location `city11-2` and then the package is where it should be.

Sometimes, either the preflight phase or the post-flight phase is not needed, or even both phases can be omitted. For package 3, the latter is the case. It has to go from `city12-3` to `city1-3`, which are both airports. Therefore, there will only be an entry for package 3 in the in-flight file.

Package 6 does not need to go by airplane. Its end location is in the same city as its start location. Since the start location is not an airport, the entry for this package will occur in the pre-flight phase. This is also a nice example to illustrate the choice explained in Section B.1.3. Because we put this package in pre-flight transport, it can share the truck with package 8 that conveniently starts at the same location and needs to go to the airport too. If package 6 was put in the post-flight phase, the truck would be at the airport because of package 8. The truck would have to go back and forth to transport package 6, which is a waste of movements. If, to make the argument complete, the start and end locations of package 6 were reversed, hence it would start at the airport, it is obvious that it is more efficient to put it in the post-flight phase.

We made three copies of the file in Table B.1, and replaced the start and end locations according to whether it would become the pre-, post-, or in-flight file. That resulted in the files in Table B.3, B.4 and B.5 respectively.

Simply replacing the start and end location in the file of Table B.1 is quite rude and introduces many superfluous data. Fortunately, we had just explained our filtering method, which we will also use to clean up these three files, resulting in Table B.6, B.7 and B.8.

B.3 Practical Mapping or Transformation

Before we could test the performance of both planners and ALNS-PDPTW, we had to make the problem instances fit for ALNS-PDPTW. We will use Table B.10 as an example. We will first explain the format of the file, later we will point out some problems that occurred because of this format.

We create the problem instances for ALNS-PDPTW using the data from files like those in Table B.6, B.7 and B.8. Due to the size of the problem, we will use a different problem as an example. The most interesting data in the filtered and coordinated problem instances are the starting locations of the packages, airplanes and trucks, and of course the end-locations of the packages. The data of the planning problem instances we use to explain the creation of the problem instances for ALNS-PDPTW, is gathered in Table B.9.

At the first line in Table B.10 there are two numbers. The first represents the number of requests. The number of requests is equal to the sum of the number of packages in the planning problem instances. In this case, there is a total of four requests. The second is the number of vehicles. The number of vehicles is equal to the sum of the number of airplanes and trucks in the planning problem instances. In this case, there is also a total of four vehicles. It is a coincidence that the number of requests and vehicles are equal, most often they are not.

On the second line are some variables used intern by ALNS-PDPTW. We refer to Røpke [57] for the precise meaning of these variables.

In the next block of lines, the vehicles are defined. Every line has the same format: `<vehicle id> <capacity> <start x coord.> <start y coord.> <end x coord.> <end y coord.> <time window begin> <time window end>`. The first number is just an identification number of the vehicle. It is not used in any calculation. The next number is the maximum capacity of the vehicles. The maximum capacity should be larger or equal than the total number of packages in each transportation phase to simulate unlimited capacities. We have derived this number from the total unique packages in the planning problem instances.

The following two numbers represent the start location of the vehicle. Originally, these are the x- and y-coordinates of a node. We do not represent nodes with coordinates, but with a city number and a location number. We replace the x-coordinate with the city number and the y-coordinate with the location number. It is required by ALNS-PDPTW that each vehicle returns to a depot when finished. In the Logistics Planning Problem, we do not know, nor care were the vehicle end up when they have finished their pick-up and delivery requests. Therefore, a depot is introduced into the problem, and represented with city and location with number

zero, which are the two zero's following the start location.

The last two numbers represent the time window. We do not use a time restriction, so we have to make sure the time window is large enough. In Section 3.4.1, we showed how the maximum duration for each instance of the Logistics Planning Problem could be calculated. We use the same number here.

In next block of lines, the requests are defined. Every line has the same format:
<request id> <demand> <pick-up x> <pick-up y> <delivery x>
<delivery y> <pick-up time window start> <pick-up time window end>
<delivery time window start> < delivery time window end>
<pick-up node precedence> <delivery node precedence>
<number of vehicles that can serve this requests>
<list of vehicles that can serve the requests>. The first number is just an identification number, it is not used for any calculations. The second number is the size of the package. Since in the Logistics Planning Problem, the size of the packages are of no concern since the capacity of the vehicles are infinite, we chose for a value of one. If we would choose a different number here, the maximum capacity of the vehicles would have to be adjusted accordingly.

The next four numbers are the pick-up and delivery locations. We used the same substitution of the coordinates with the city and location numbers as we used by the definition of the vehicles. The following four numbers represent the pick-up time window and the delivery time window. Both pick-up and delivery time windows are equal for one request, because we do not want to introduce more constraints on when to pick-up or deliver packages. We use the time windows to make the distinction between the pre-flight, post-flight and in-flight stages. The time windows are calculated according the description of the maximum duration in Section 3.4.1.

The next two numbers are used to establish precedences between the requests. In ALNS-PDPTW, it is already ensured that requests are picked up before they are delivered. We do not want to put extra restrictions on the order in which requests are served, other than to make the distinction between the different phases. Therefore, we use the same precedence number for the pick-up precedence and the delivery precedence, and we number the pre-, in- and post-flight with one, two and three respectively. It is sufficient to make the distinction between the phases using either the time windows or the precedences. To keep the input and output files more understandable, we use both. The final numbers connect the vehicles with the request. The first of them represents the amount of vehicles, the rest of the numbers are the identification numbers of the vehicles that are able to serve this request.

The last two blocks represent the distance-matrix and the time-matrix. We

do not make a difference between time and distance, hence the two matrices are equal. The nodes in ALNS-PDPTW are numbered as follows. First the start-location of the first request, then the end-location of the first request. This is repeated until the last request, then followed with the start-location and the end-location of the first vehicle etc. The matrices are then generated along the following rules. The distance between the same locations, hence the nodes have the same location number and the same city number, is equal to zero. The distance from any location to the depot is zero too. This way we do not introduce extra movements while using a depot. The distance from a location to another with the same city number has a distance of one. All other distances are equal to the maximum length in which the problem could be solved. This is done to discourage a vehicle to use that path, since in the original problem, this link does not exist.

B.4 Enlarging the Problem Instances

After testing with the original problem instances and the derivatives of them, it appeared that the hardest problems had more or less the same proportions between the variables. The number of cities was about 0.7 times the number of packages. The number of locations in a city was on average 8. This number did not needed to be proportional in the number of packages. The goal is to enlarge the problem instances, and therefore the variables, in such a way that the distribution of the packages would stay the same. The scaling is already in the number of cities, therefore, the total number of locations will grow in the right proportion. The number of trucks was about the same amount as the number of packages or slightly less, hence we chose for 0.9 times the number of packages. It followed from the original problem instances that the number of airplanes was about eight airplanes per city. We considered that an excessive amount. An average load of ten packages per airplane was decided to be quite reasonable. Using these proportions, we have generated larger files.

To eliminate the possibility that the problems after filtering would become much smaller, we made sure that all packages would also get a goal location. In the original problem generator, the number of packages that would return in the goal function was randomly decided.

We had to make sure that every city had at least one truck. Since the proportions we applied to generate the files ensures that the amount of cities is smaller than the number of trucks, we assigned the first trucks to the cities in the following way: truck 1 to city 1, truck 2 to city 2, etc. The remaining trucks we have distributed over the cities randomly. The location where a truck would start was also determined randomly. The locations where packages would start and finish were also determined randomly, as was the location where an airplane would start. The latter has been done by randomly choosing a city number and combining it

with the airport-number.

Furthermore, generating large problem instances could be done rather straight forward, while keeping the planning-format in mind. After this initial creation of new files, we have performed the same pre-processing steps as with the original files.

```
(define (problem strips-log-x-3)
(:domain logistics-strips)
(:objects package9 package8 package7 package6 package5 package4
          package3 package2 package1 city14 city13 city12 city11
          city10 city9 city8 city7 city6 city5 city4 city3 city2
          city1 truck14 truck13 truck12 truck11 truck10 truck9
          truck8 truck7 truck6 truck5 truck4 truck3 truck2 truck1
          plane4 plane3 plane2 plane1 city14-3 city14-2 city14-1
          city13-3 city13-2 city13-1 city12-3 city12-2 city12-1
          city11-3 city11-2 city11-1 city10-3 city10-2 city10-1
          city9-3 city9-2 city9-1 city8-3 city8-2 city8-1 city7-3
          city7-2 city7-1 city6-3 city6-2 city6-1 city5-3 city5-2
          city5-1 city4-3 city4-2 city4-1 city3-3 city3-2 city3-1
          city2-3 city2-2 city2-1 city1-3 city1-2 city1-1 )
(:init (obj package9) (obj package8) (obj package7) (obj package6)
       (obj package5) (obj package4) (obj package3) (obj package2)
       (obj package1) (city city14)  (city city13)  (city city12)
       (city city11)  (city city10)  (city city9)   (city city8)
       (city city7)   (city city6)   (city city5)   (city city4)
       (city city3)   (city city2)   (city city1)   (truck truck14)
       (truck truck13) (truck truck12) (truck truck11)
       (truck truck10) (truck truck9) (truck truck8) (truck truck7)
       (truck truck6) (truck truck5)  (truck truck4) (truck truck3)
       (truck truck2) (truck truck1)  (airplane plane4)
       (airplane plane3) (airplane plane2) (airplane plane1)
       (location city14-2) (location city14-1) (location city13-2)
       (location city13-1) (location city12-2) (location city12-1)
       (location city11-2) (location city11-1) (location city10-2)
       (location city10-1) (location city9-2)  (location city9-1)
       (location city8-2)  (location city8-1)  (location city7-2)
       (location city7-1)  (location city6-2)  (location city6-1)
       (location city5-2)  (location city5-1)  (location city4-2)
       (location city4-1)  (location city3-2)  (location city3-1)
       (location city2-2)  (location city2-1)  (location city1-2)
       (location city1-1)  (airport city14-3)  (location city14-3)
       (airport city13-3)  (location city13-3) (airport city12-3)
       (location city12-3) (airport city11-3)  (location city11-3)
       (airport city10-3)  (location city10-3) (airport city9-3)
       (location city9-3)  (airport city8-3)   (location city8-3)
       (airport city7-3)   (location city7-3)  (airport city6-3)
       (location city6-3)  (airport city5-3)   (location city5-3)
       (airport city4-3)   (location city4-3)  (airport city3-3)
       (location city3-3)  (airport city2-3)   (location city2-3)
       (airport city1-3)  (location city1-3)
       (in-city city14-3 city14)   (in-city city14-2 city14)
       (in-city city14-1 city14)   (in-city city13-3 city13)
       (in-city city13-2 city13)   (in-city city13-1 city13)
       (in-city city12-3 city12)   (in-city city12-2 city12)
```

```
        (in-city city12-1 city12)   (in-city city11-3 city11)
        (in-city city11-2 city11)   (in-city city11-1 city11)
        (in-city city10-3 city10)   (in-city city10-2 city10)
        (in-city city10-1 city10)   (in-city city9-3 city9)
        (in-city city9-2 city9)     (in-city city9-1 city9)
        (in-city city8-3 city8)     (in-city city8-2 city8)
        (in-city city8-1 city8)     (in-city city7-3 city7)
        (in-city city7-2 city7)     (in-city city7-1 city7)
        (in-city city6-3 city6)     (in-city city6-2 city6)
        (in-city city6-1 city6)     (in-city city5-3 city5)
        (in-city city5-2 city5)     (in-city city5-1 city5)
        (in-city city4-3 city4)     (in-city city4-2 city4)
        (in-city city4-1 city4)     (in-city city3-3 city3)
        (in-city city3-2 city3)     (in-city city3-1 city3)
        (in-city city2-3 city2)     (in-city city2-2 city2)
        (in-city city2-1 city2)     (in-city city1-3 city1)
        (in-city city1-2 city1)     (in-city city1-1 city1)
        (at plane4 city10-3)        (at plane3 city1-3)
        (at plane2 city11-3)        (at plane1 city11-3)
        (at truck14 city14-1)       (at truck13 city13-1)
        (at truck12 city12-1)       (at truck11 city11-1)
        (at truck10 city10-1)       (at truck9 city9-2)
        (at truck8 city8-2) (at truck7 city7-1) (at truck6 city6-2)
        (at truck5 city5-1) (at truck4 city4-2) (at truck3 city3-2)
        (at truck2 city2-1) (at truck1 city1-1)
        (at package9 city4-2)    (at package8 city9-1)
        (at package7 city1-2)    (at package6 city9-1)
        (at package5 city12-2)   (at package4 city8-1)
        (at package3 city12-3)   (at package2 city12-2)
        (at package1 city8-3))
(:goal (and (at package9 city11-2)   (at package8 city5-3)
            (at package7 city10-3)   (at package6 city9-3)
            (at package5 city10-1)   (at package4 city13-1)
            (at package3 city1-3))))
```

Table B.1: Example problem in STRIPS format.

```
(define (problem strips-log-x-3)
(:domain logistics-strips)
(:objects package9 ... package3 city13 city12 ... city8 city5
          city4 city1 truck13 ... truck8 truck4 truck1
          plane4 ... plane1 city13-3 ... city8-1
          city5-3 ... city5-1 city4-3 ... city4-1
          city1-3 ... city1-1  )
(:init (obj package9) ... (obj package3)
       (city city13) ... (city city8) (city city5) (city city4)
       (city city1) (truck truck13) ... (truck truck8)
       (truck truck4)  (truck truck1)
       (airplane plane4) ... (airplane plane1)
       (location city13-3) ... (location city8-1)
       (location city5-3) ... (location city5-1)
       (location city4-3) ... (location city4-1)
       (location city1-3) ... (location city1-1)
       (airport city13-3) ... (airport city8-3)
       (airport city5-3) (airport city4-3) (airport city1-3)
       (in-city city13-3 city13) ... (in-city city8-1 city8)
       (in-city city5-3 city5)   ... (in-city city5-1 city5)
       (in-city city4-3 city4)   ... (in-city city4-1 city4)
       (in-city city1-3 city1)   ... (in-city city1-1 city1)
       (at plane4 city10-3) ... (at plane1 city11-3)
       (at truck13 city13-1) ... (at truck8 city8-2)
       (at truck4 city4-2) (at truck1 city1-1)
       (at package9 city4-2) ... (at package3 city12-3))
(:goal (and (at package9 city11-2) ... (at package3 city1-3))))
```

Table B.2: The example file in STRIPS format after filtering.

```
(define (problem strips-log-x-3)
(:domain logistics-strips)
(:objects package9 ... package1 city14 ... city1
          truck14 ... truck1 plane4 ... plane1
          city14-3 ... city1-1 )
(:init (obj package9) ... (obj package1)
       (city city14) ... (city city1)
       (truck truck14) ... (truck truck1)
       (airplane plane4) ... (airplane plane1)
       (location city14-3) ... (location city1-1)
       (airport city14-3) ... (airport city1-3)
       (in-city city14-3 city14) ... (in-city city1-1 city1)
       (at plane4 city10-3) ... (at plane1 city11-3)
       (at truck14 city14-1) ... (at truck1 city1-1)
       (at package9 city4-2) ... (at package1 city8-3))
(:goal (and (at package9 city4-3) (at package8 city9-3)
            (at package7 city1-3) (at package6 city9-3)
            (at package5 city12-3) (at package4 city8-3))))
```

Table B.3: The preflight phase of the example file in STRIPS format.

```
(define (problem strips-log-x-3)
(:domain logistics-strips)
(:objects package9 ... package1 city14 ... city1 truck14 ... truck1
          plane4 ... plane1 city14-3 ... city1-1  )
(:init (obj package9) ... (obj package1)
       (city city14) ... (city city1)
       (truck truck14) ... (truck truck1)
       (airplane plane4) ... (airplane plane1)
       (location city14-3) ... (location city1-1)
       (airport city14-3) ... (airport city1-3)
       (in-city city14-3 city14) ... (in-city city1-1 city1)
       (at plane4 city10-3) ... (at plane1 city11-3)
       (at truck14 city14-1) ... (at truck1 city1-1)
       (at package9 city11-3) ... (at package1 city8-3))
(:goal (and (at package9 city11-2) (at package5 city10-1)
            (at package4 city13-1))))
```

Table B.4: The post-flight phase of the example file in STRIPS format.

```
(define (problem strips-log-x-3)
(:domain logistics-strips)
(:objects package9 ... package1 city14 ... city1
          truck14 ... truck1 plane4 ... plane1
          city14-3 ... city1-13 )
(:init (obj package9) ... (obj package1)
       (city city14) ... (city city1)
       (truck truck14) ... (truck truck1)
       (airplane plane4) ... (airplane plane1)
       (location city14-3) ... (location city1-1)
       (airport city14-3) ... (airport city1-3)
       (in-city city14-3 city14) ... (in-city city1-1 city1)
       (at plane4 city10-3) ... (at plane1 city11-3)
       (at truck14 city14-1) ... (at truck1 city1-1)
       (at package9 city4-3) ... (at package1 city8-3))
(:goal (and (at package9 city11-3) (at package8 city5-3)
            (at package7 city10-3) (at package5 city10-3)
            (at package4 city13-3) (at package3 city1-3))))
```

Table B.5: The in-flight phase of the example file in STRIPS format.

```
(define (problem strips-log-x-3)
(:domain logistics-strips)
(:objects package9 ... package4
          city12 ... city8 city4 city1 truck12 truck9 truck8
          truck4 truck1 plane4 plane3 plane2 plane1
          city12-3 ... city12-1 city11-3 city10-3
          city9-3 ... city8-1 city4-3 ... city4-1
          city1-3 ... city1-1 )
(:init (obj package9) ... (obj package4)
       (city city12) ... (city city8) (city city4) (city city1)
       (truck truck12) (truck truck9) (truck truck8)
       (truck truck4)  (truck truck1)
       (airplane plane4) ... (airplane plane1)
       (location city12-3) ... (location city12-1)
       (location city9-3) ...  (location city8-1)
       (location city4-3) ... (location city4-1)
       (location city1-3) ... (location city1-1)
       (location city11-3) (location city10-3)
       (airport city12-3) ... (airport city8-3)
       (airport city4-3)   (airport city1-3)
       (in-city city12-3 city12) ... (in-city city12-1 city12)
       (in-city city11-3 city11) (in-city city10-3 city10)
       (in-city city9-3 city9) ... (in-city city9-1 city9)
       (in-city city8-3 city8) ... (in-city city8-1 city8)
       (in-city city4-3 city4) ... (in-city city4-1 city4)
       (in-city city1-3 city1) ... (in-city city1-1 city1)
       (at plane4 city10-3) ... (at plane1 city11-3)
       (at truck12 city12-1) (at truck9 city9-2)
       (at truck8 city8-2) (at truck4 city4-2)
       (at truck1 city1-1) (at package9 city4-2)
       (at package8 city9-1)       (at package7 city1-2)
       (at package6 city9-1)       (at package5 city12-2)
       (at package4 city8-1))
(:goal (and (at package9 city4-3)  (at package8 city9-3)
            (at package7 city1-3)  (at package6 city9-3)
            (at package5 city12-3) (at package4 city8-3))))
```

Table B.6: The preflight phase of the example file in STRIPS format after filtering.

```
(define (problem strips-log-x-3)
(:domain logistics-strips)
(:objects package9 package5 package4 city13 city11 city10 city1
          truck13 truck11 truck10 plane4 ... plane1
          city13-3 ... city13-1 city11-3 ... city10-1 city1-3 )
(:init (obj package9) (obj package5) (obj package4)
       (city city13) (city city11) (city city10) (city city1)
       (truck truck13) (truck truck11) (truck truck10)
       (airplane plane4) ... (airplane plane1)
       (location city13-3) ... (location city13-1)
       (location city11-3) ... (location city10-1)
       (location city1-3) (airport city13-3) (airport city11-3)
       (airport city10-3)  (airport city1-3)
       (in-city city13-3 city13) ... (in-city city13-1 city13)
       (in-city city11-3 city11) ... (in-city city10-1 city10)
       (in-city city1-3 city1)
       (at plane4 city10-3) ... (at plane1 city11-3)
       (at truck13 city13-1)       (at truck11 city11-1)
       (at truck10 city10-1)       (at package9 city11-3)
       (at package5 city10-3)      (at package4 city13-3))
(:goal (and (at package9 city11-2) (at package5 city10-1)
            (at package4 city13-1))))
```

Table B.7: The post-flight phase of the example file in STRIPS format after filtering.

```
(define (problem strips-log-x-3)
(:domain logistics-strips)
(:objects package9 ... package3 city13 ... city8 city5 city4
          city1 plane4 ... plane1 city13-3 ... city8-1
          city5-3 ... city4-1 city1-3 ... city1-1 )
(:init (obj package9) ... (obj package3)
       (city city13) ... (city city8)
       (city city5) (city city4) (city city1)
       (airplane plane4) ... (airplane plane1)
       (location city13-3) ... (location city8-1)
       (location city5-3) ... (location city4-1)
       (location city1-3) ... (location city1-1)
       (airport city13-3) ... (airport city8-3)
       (airport city5-3) (airport city4-3) (airport city1-3)
       (in-city city13-3 city13) ... (in-city city8-1 city8)
       (in-city city5-3 city5) ... (in-city city4-1 city4)
       (in-city city1-3 city1) ...  (in-city city1-1 city1)
       (at plane4 city10-3) ... (at plane1 city11-3)
       (at package9 city4-3) (at package8 city9-3)
       (at package7 city1-3) (at package5 city12-3)
       (at package4 city8-3) (at package3 city12-3))
(:goal (and (at package9 city11-3)  (at package8 city5-3)
            (at package7 city10-3)  (at package5 city10-3)
            (at package4 city13-3)  (at package3 city1-3))))
```

Table B.8: The in-flight phase of the example file in STRIPS format after filtering.

```
//preflight
                (at plane2 city1-2)
                (at plane1 city2-2)
                (at truck4 city4-1)
                (at truck1 city1-1)
                (at package3 city1-1)
                (at package1 city4-1))
    (:goal (and (at package3 city1-2)
                (at package1 city4-2))))


//postflight
                (at plane2 city1-2)
                (at plane1 city2-2)
                (at truck1 city1-1)
                (at package2 city1-2))
    (:goal (and (at package2 city1-1))))


//inflight
                (at plane2 city1-2)
                (at plane1 city2-2)
                (at package1 city4-2))
    (:goal (and (at package1 city3-2))))
```

Table B.9: The relevant data of a planning problem instance for making problem instances for ALNS-PDPTW.

```
4 4
1 0 10000

0 3 4 1 0 0 0 16
1 3 1 1 0 0 0 16
2 3 1 2 0 0 0 16
3 3 2 2 0 0 0 16

0 1 1 1 1 2 0 4 0 4 1 1 1 1
1 1 4 1 4 2 0 4 0 4 1 1 1 0
2 1 4 2 3 2 4 12 4 12 2 2 2 2 3
3 1 1 2 1 1 12 16 12 16 3 3 1 1

 0   1   16  16  16  16  1   0   16  0   0   0   1   0   16  0
 1   0   16  1   1   1   0   1   16  0   1   0   0   0   1   0
 16  16  0   1   1   16  16  16  0   0   16  0   16  0   16  0
 16  1   1   0   0   1   1   16  1   0   16  0   1   0   1   0
 16  1   1   0   0   1   1   16  1   0   16  0   1   0   1   0
 16  1   16  1   1   0   1   16  16  0   16  0   1   0   1   0
 1   0   16  1   1   1   0   1   16  0   1   0   0   0   1   0
 0   1   16  16  16  16  1   0   16  0   0   0   1   0   16  0
 16  16  0   1   1   16  16  16  0   0   16  0   16  0   16  0
 0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
 0   1   16  16  16  16  1   0   16  0   0   0   1   0   16  0
 0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
 1   0   16  1   1   1   0   1   16  0   1   0   0   0   1   0
 0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
 16  1   16  1   1   1   1   16  16  0   16  0   1   0   0   0
 0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0

 0   1   16  16  16  16  1   0   16  0   0   0   1   0   16  0
 1   0   16  1   1   1   0   1   16  0   1   0   0   0   1   0
 16  16  0   1   1   16  16  16  0   0   16  0   16  0   16  0
 16  1   1   0   0   1   1   16  1   0   16  0   1   0   1   0
 16  1   1   0   0   1   1   16  1   0   16  0   1   0   1   0
 16  1   16  1   1   0   1   16  16  0   16  0   1   0   1   0
 1   0   16  1   1   1   0   1   16  0   1   0   0   0   1   0
 0   1   16  16  16  16  1   0   16  0   0   0   1   0   16  0
 16  16  0   1   1   16  16  16  0   0   16  0   16  0   16  0
 0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
 0   1   16  16  16  16  1   0   16  0   0   0   1   0   16  0
 0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
 1   0   16  1   1   1   0   1   16  0   1   0   0   0   1   0
 0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
 16  1   16  1   1   1   1   16  16  0   16  0   1   0   0   0
 0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
```

Table B.10: The VRP problem instance derived from a coordinated and filtered planning problem instances.

Appendix C

Graphs

In this chapter, we show the graphs used in Chapter 5 again for detailed study, with the exception of the tables and box-and-whisker plots.

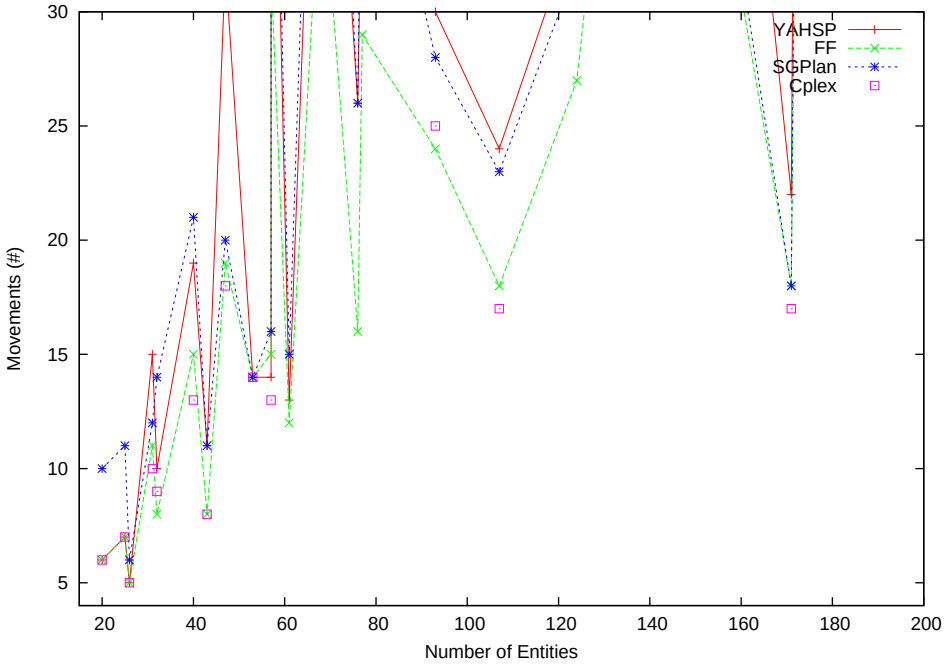


Figure C.1: Figure 5.3(b): The number of movements needed to solve the original problem instances in Detail.

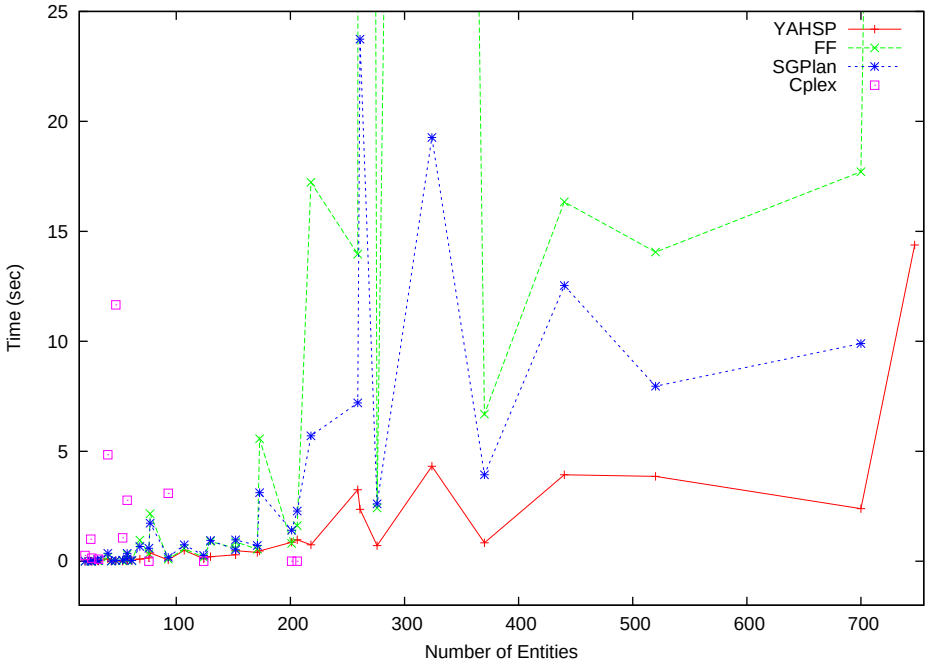


Figure C.2: Figure 5.4: A detailed view of the computation time that was needed to solve the original problem instances.

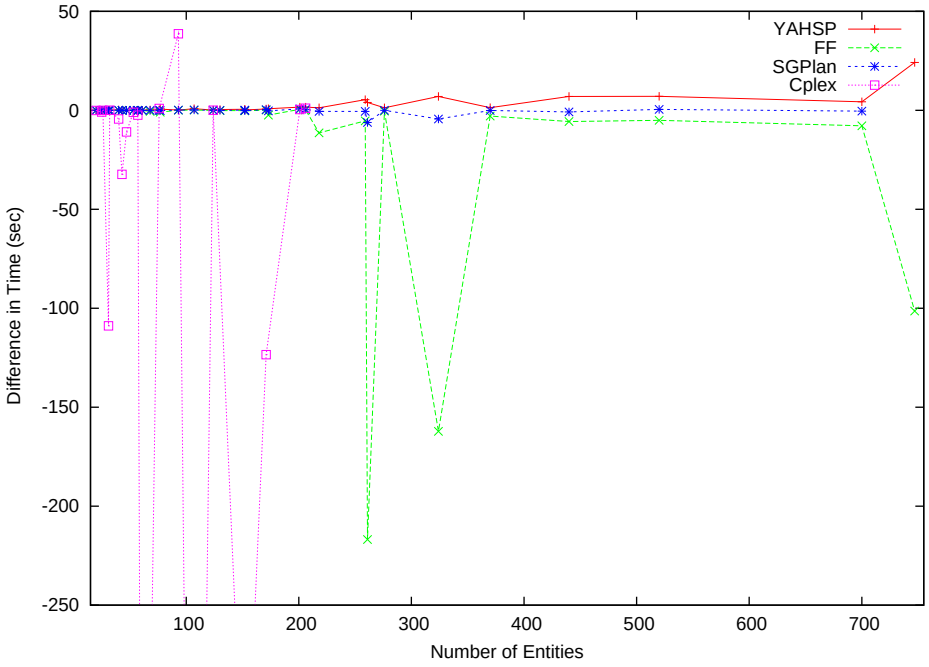
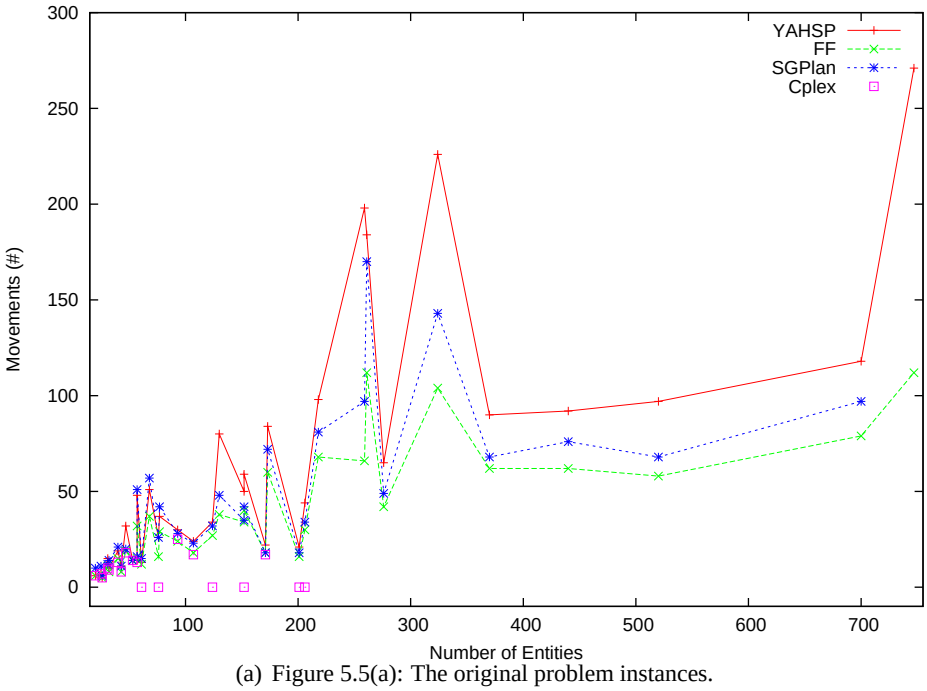
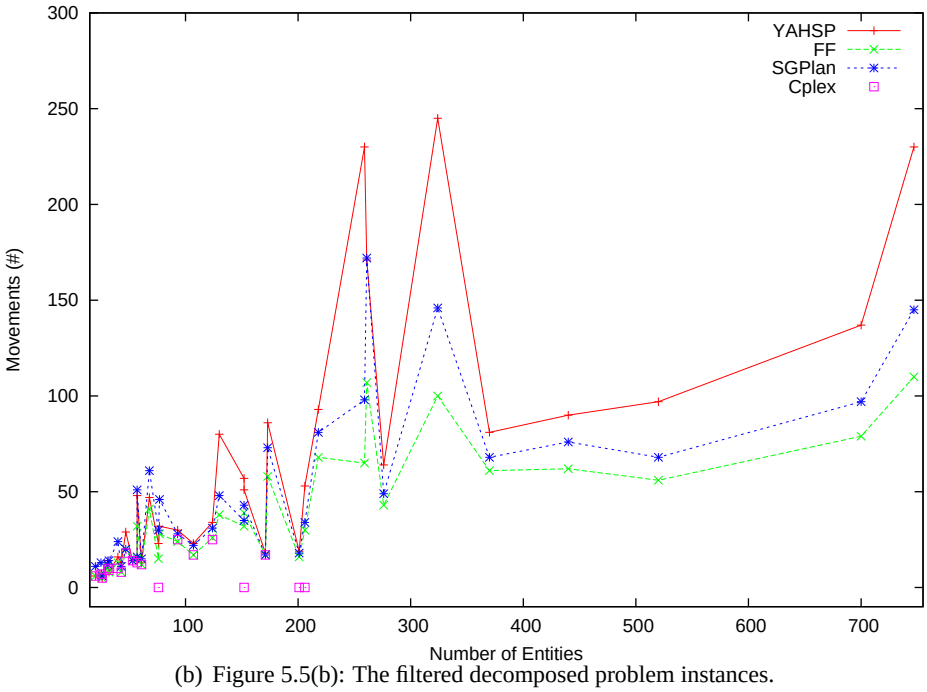


Figure C.3: Figure 5.9(a): The effect of decomposition on computation time of the original problem instances. Negative values mean that the decomposed problem instances are solved in less seconds than the original problem instance.

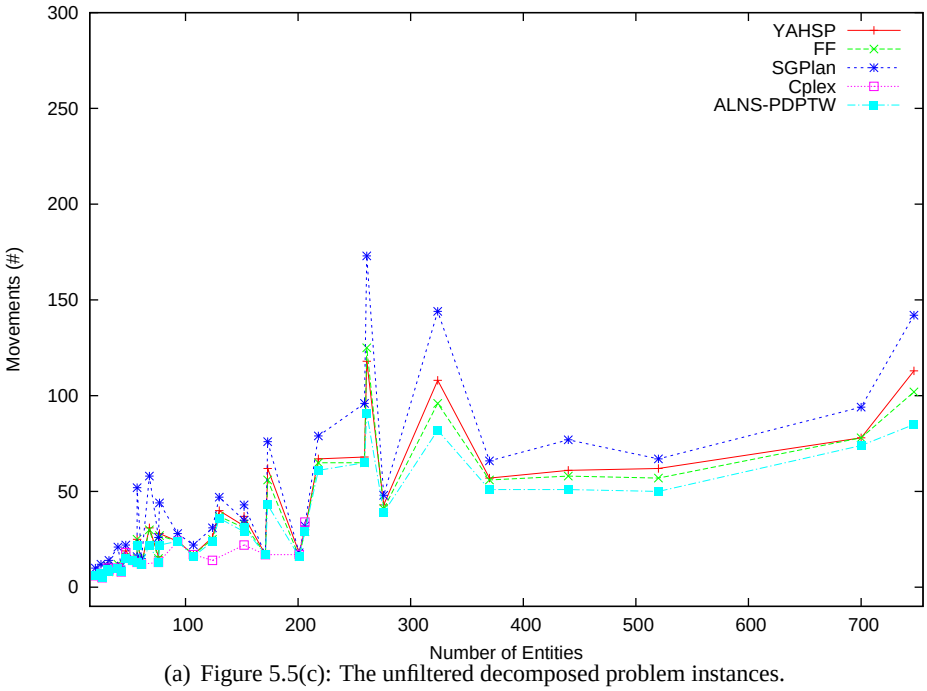


(a) Figure 5.5(a): The original problem instances.

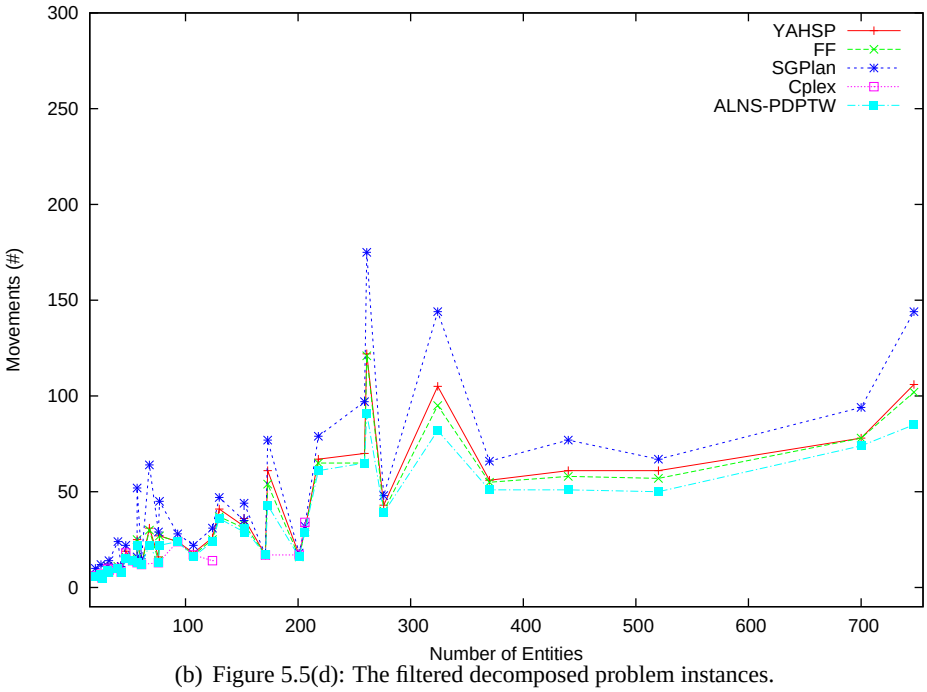


(b) Figure 5.5(b): The filtered decomposed problem instances.

Figure C.4: Figure 5.5 upper graphs: The number of movements to find a solution to the problem instances. The upper graphs show the complete problem instances, the lower graphs show the decomposed problem instances, the left graphs show the unfiltered problem instances, and the right graphs show the filtered problem instances.

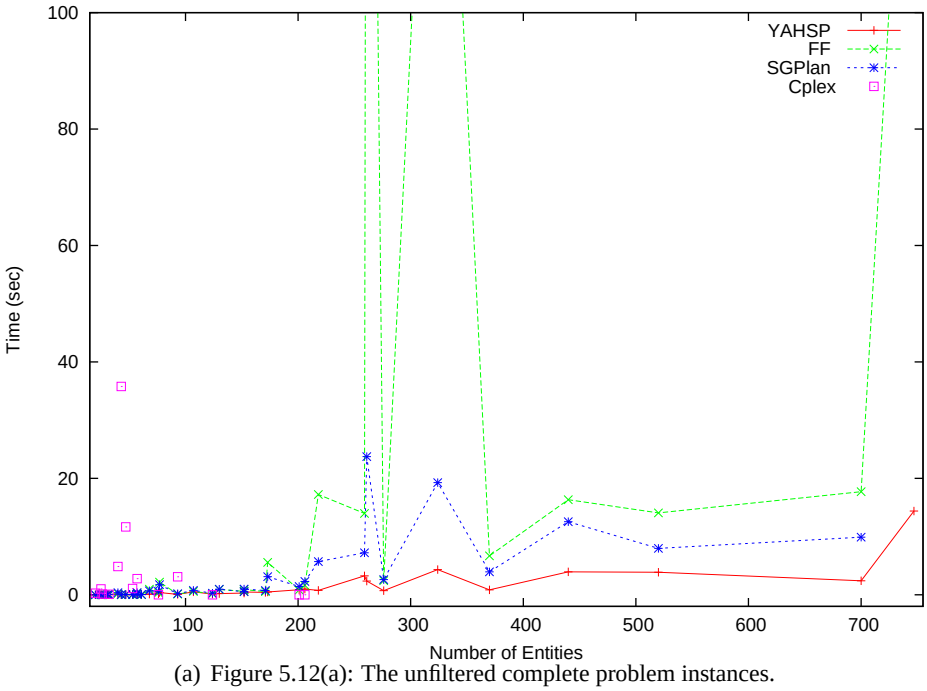


(a) Figure 5.5(c): The unfiltered decomposed problem instances.

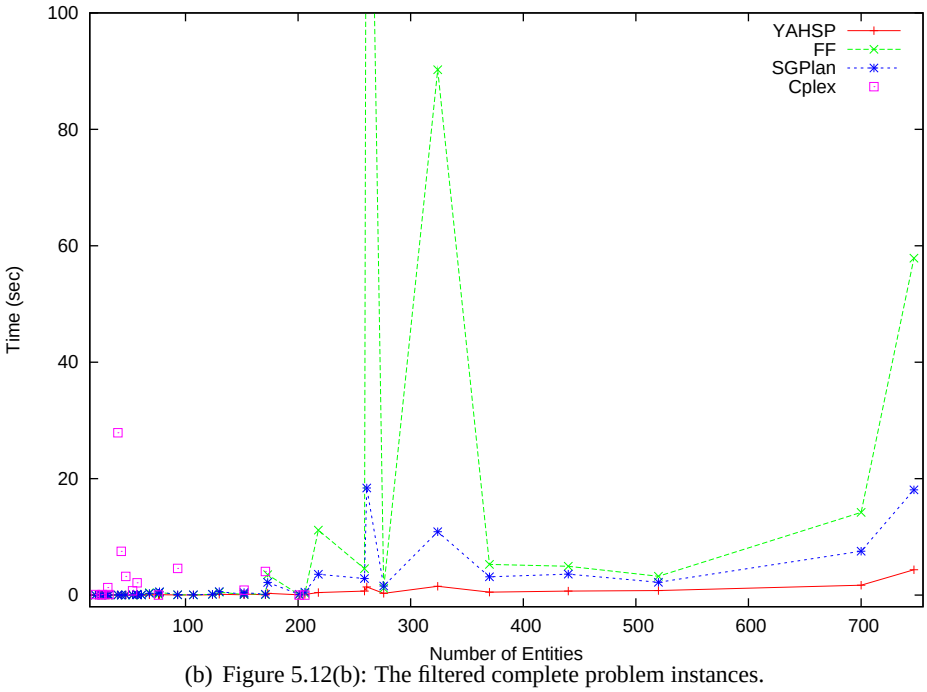


(b) Figure 5.5(d): The filtered decomposed problem instances.

Figure C.5: Figure 5.5 lower graphs: The number of movements to find a solution to the problem instances. The upper graphs show the complete problem instances, the lower graphs show the decomposed problem instances, the left graphs show the unfiltered problem instances, and the right graphs show the filtered problem instances.

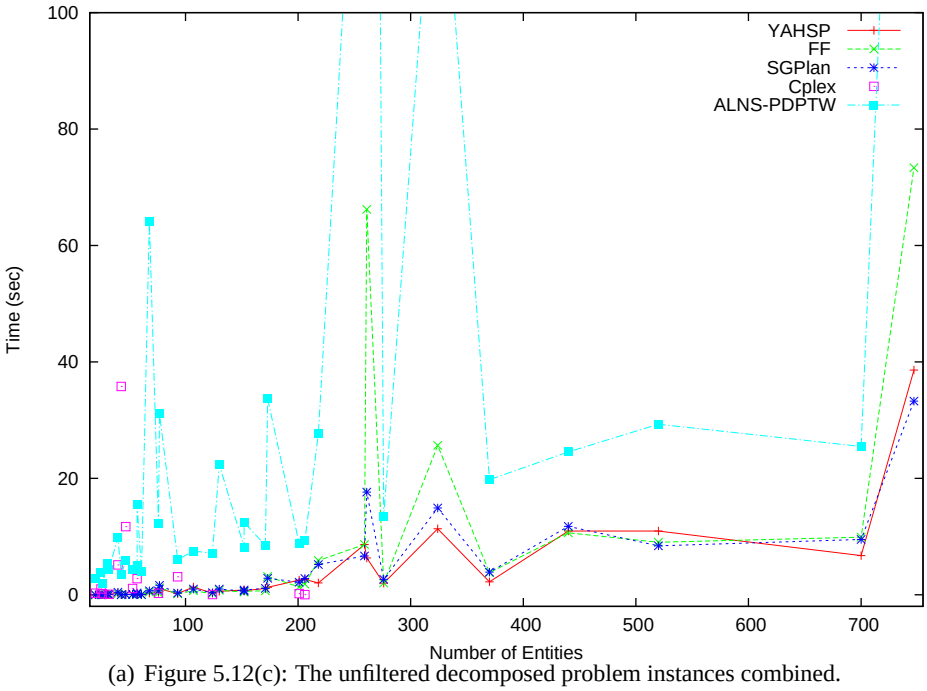


(a) Figure 5.12(a): The unfiltered complete problem instances.

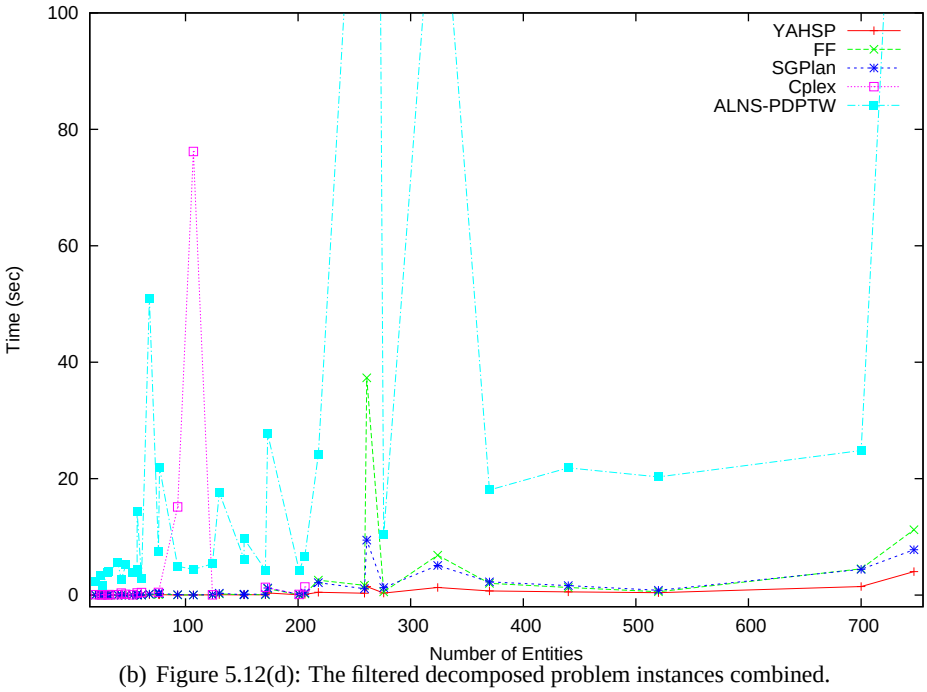


(b) Figure 5.12(b): The filtered complete problem instances.

Figure C.6: Figure 5.12 upper graphs: The computation time in seconds that was needed to solve each problem instance. In case of a decomposed problem instance, we added the computation time of the in-flight phase to those of the post- and pre-flight phases.

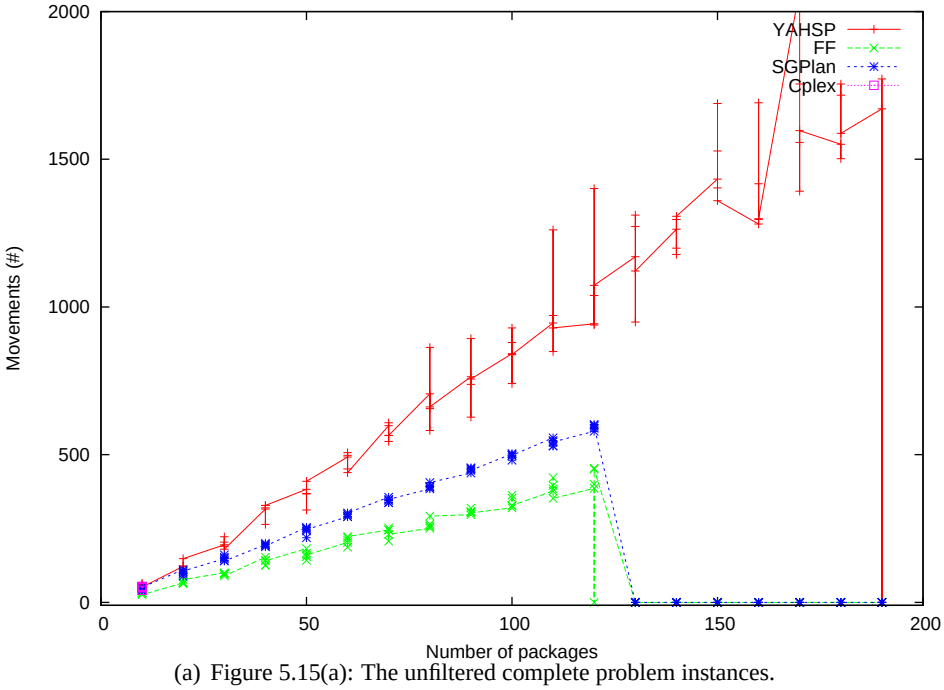


(a) Figure 5.12(c): The unfiltered decomposed problem instances combined.

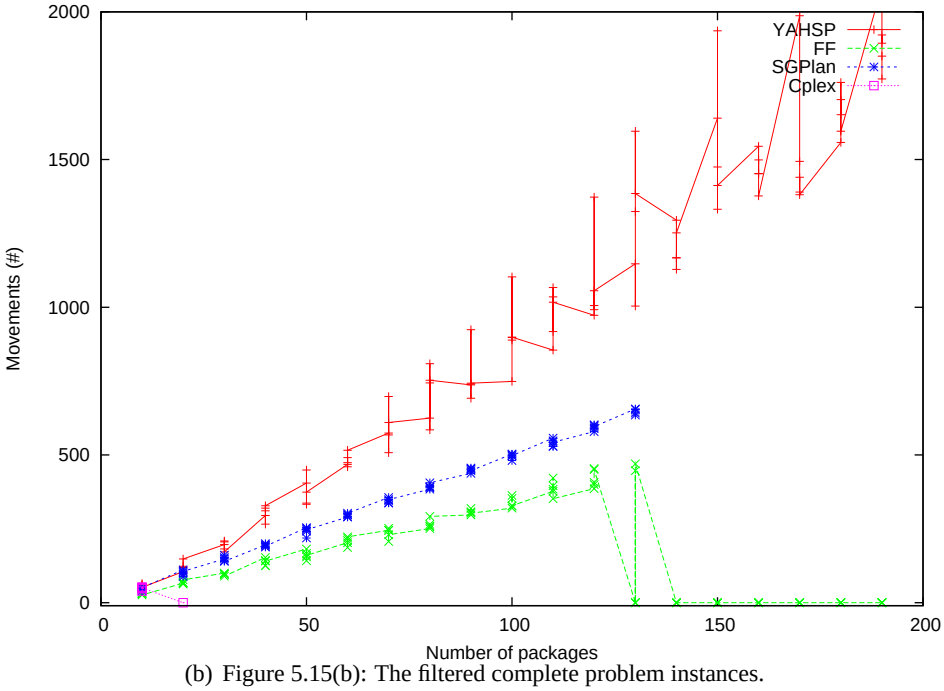


(b) Figure 5.12(d): The filtered decomposed problem instances combined.

Figure C.7: Figure 5.12 lower graphs: The computation time in seconds that was needed to solve each problem instance. In case of a decomposed problem instance, we added the computation time of the in-flight phase to those of the post- and pre-flight phases.

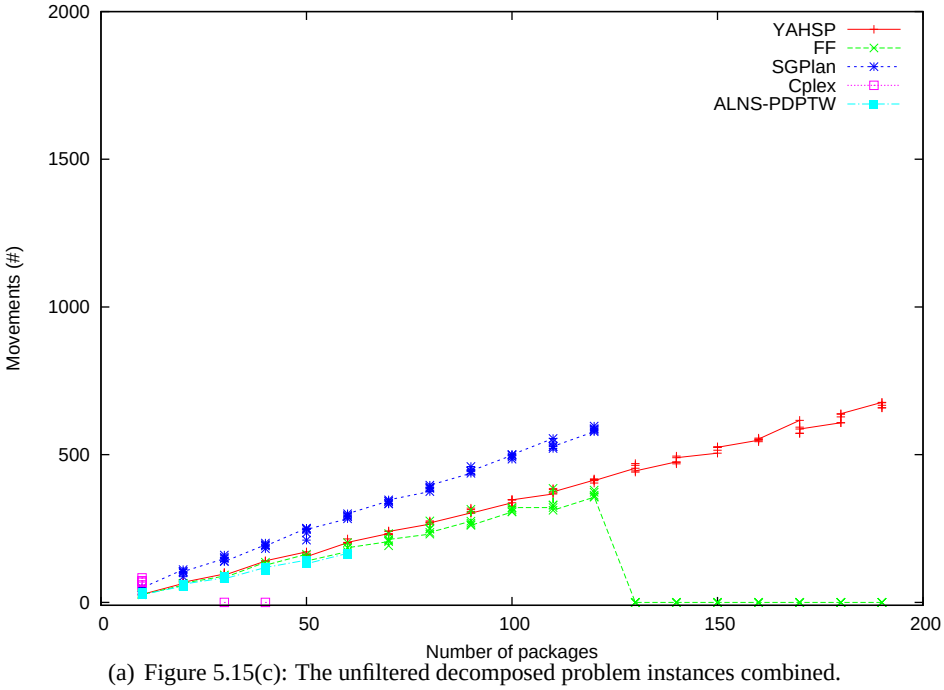


(a) Figure 5.15(a): The unfiltered complete problem instances.

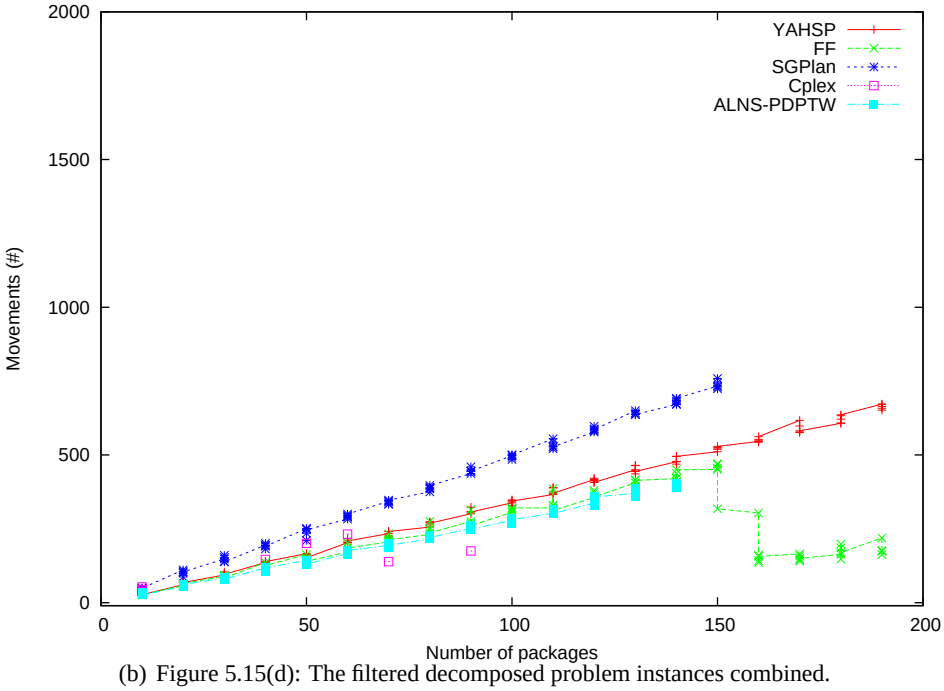


(b) Figure 5.15(b): The filtered complete problem instances.

Figure C.8: Figure 5.15 upper graphs: The calculated plan quality in the number of move actions from vehicles needed to solve each of the larger problem instance.

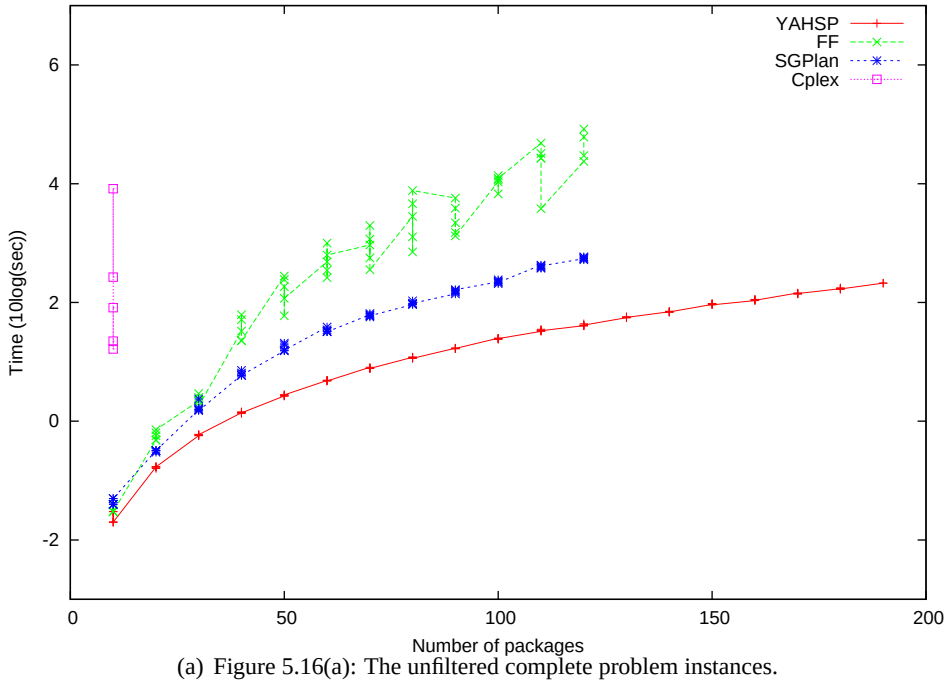


(a) Figure 5.15(c): The unfiltered decomposed problem instances combined.

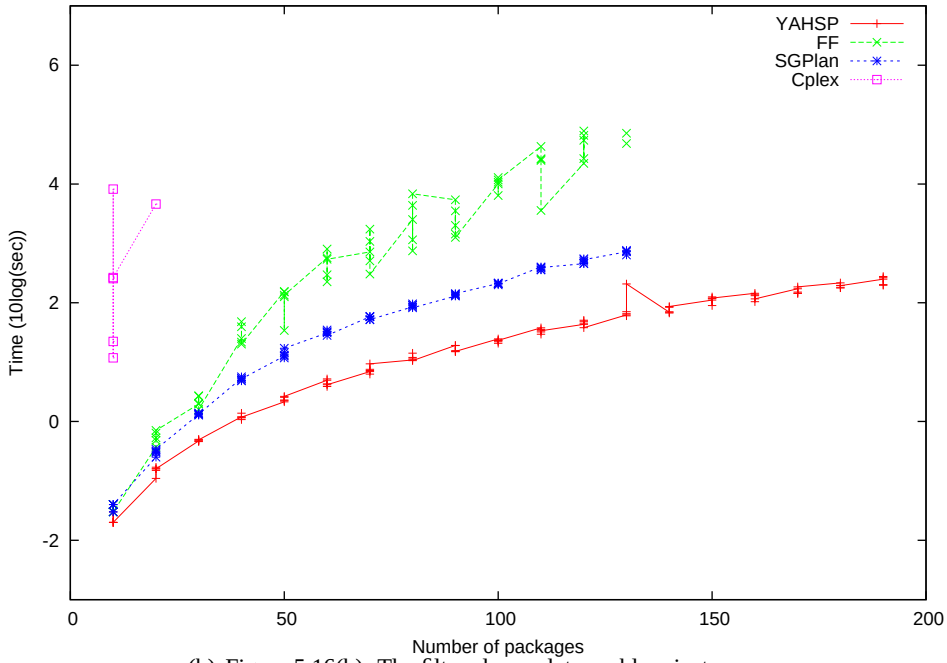


(b) Figure 5.15(d): The filtered decomposed problem instances combined.

Figure C.9: Figure 5.15 lower graphs: The calculated plan quality in the number of move actions from vehicles needed to solve each of the larger problem instance.

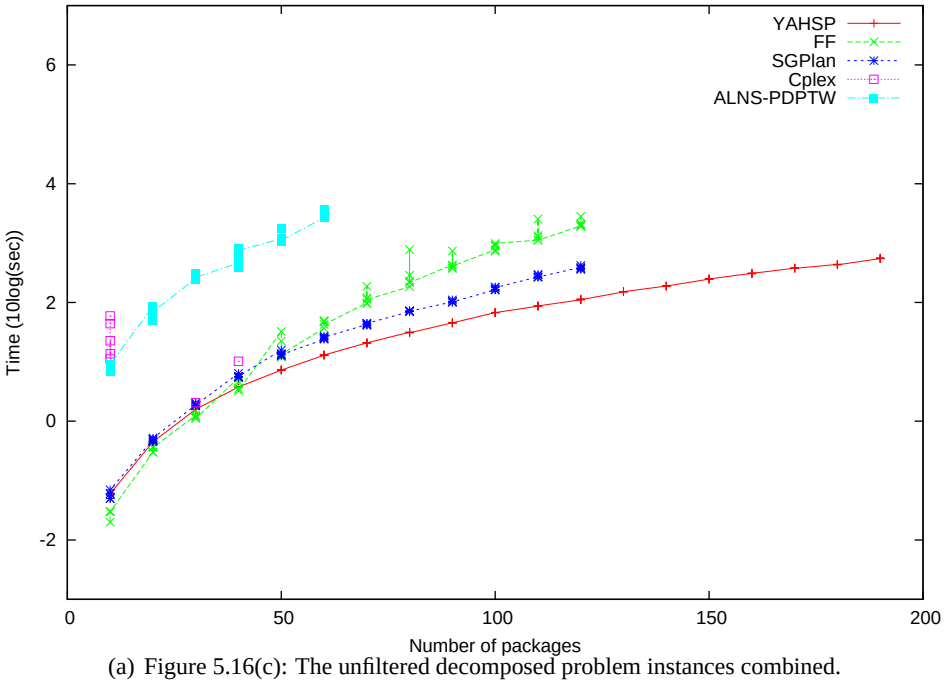


(a) Figure 5.16(a): The unfiltered complete problem instances.

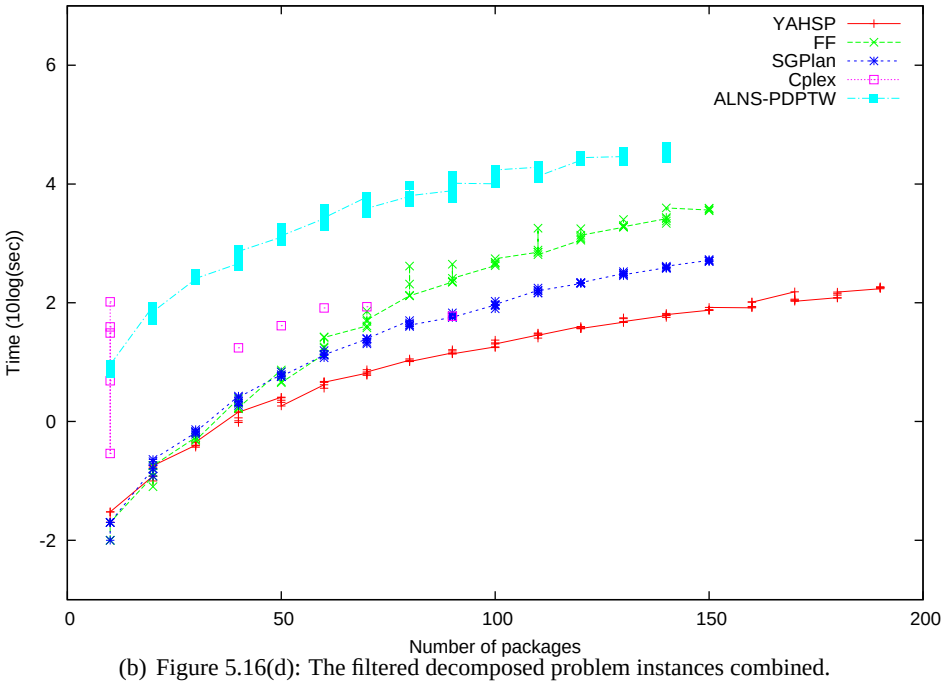


(b) Figure 5.16(b): The filtered complete problem instances.

Figure C.10: Figure 5.16 upper graphs: The computation time in seconds against a logarithmic scale.



(a) Figure 5.16(c): The unfiltered decomposed problem instances combined.



(b) Figure 5.16(d): The filtered decomposed problem instances combined.

Figure C.11: Figure 5.16 lower graphs: The computation time in seconds against a logarithmic scale.