Transport planning and scheduling

Mathijs de Weerdt

December 21, 1999

The aim of our research project is to create a dynamic, multi-agent system to support distributed scheduling and control for, for example, freight transport. The involved organizations usually refuse to release critical information to another. This calls for a distributed approach. Another problem is that due to disturbances in the world, like delays, often rescheduling is necessary.

Some methods exist to solve simple planning and scheduling problems in an efficient way and much research is done to improve these methods. Also some research is done to adapt these methods for use in a dynamic environment. Our focus is on the design of a structure that allows organizations to cooperate efficiently, but can also keep most of their critical information to themselves. We suppose the existing dynamic scheduling methods can be used in such a structure.

This report describes the problem we are working on in a detailed way. It also describes some known ways to deal with these kinds of problems and the approach we will take to reach our goals.

Contents

1. Introduction

Planning and control of cargo transport by different organizations is a difficult task. Our aim is to support this process by an information technology infrastructure. To build an efficient control system for cargo transportation problems, two topics have to be thoroughly investigated. Since transportation involves scheduling and due to delays, additional orders, etc. also rescheduling, a dynamic scheduling algorithm has to be found. The other topic emerges from the distributed aspect of the problem. Many organizations are involved in the transportation of one piece of cargo and often they do not want to share all their information. Therefore each organization has to be represented by an entity which communicates and negotiates with the others. A system consisting of several of these entities is called a multi-actor or multi-agent system.

We consider both planning and scheduling. Planning is the process of deciding, given a set of goals, which actions to do and in which order. Scheduling is the process of deciding when these actions should be done, by whom, and which resources may be used. Since these processes are narrowly interrelated, we see them as one process that has to be executed by the multi-agent system we are building.

In this document we will outline our research project and give some background information. First we describe the aim of this project and the approach we take to attain our aims. In the next chapter we give a description of the problem we are dealing with. The remaining chapters investigate the hardness of the problem and several existing methods to deal with similar problems.

1.1. Aims of this research project

Our main goal is to design algorithms and a control architecture for planning and scheduling in dynamic environments with the following properties:

- Different actors are involved. These actors are both cooperating and competing with each other. Since they are competing, they do not want to share all their information. Since they are cooperating they must share some information.
- Planning, scheduling and task execution are distributed processes. Local solutions may be in conflict with each other.
- Therefore, coordination of sub-plans is of utmost importance.
- Due to the dynamic environment, re-planning and rescheduling is essential.
- Several actors can provide the resources needed by most subproblems and sub-plans. So there are many ways to execute such a sub-plan.

Creating a system with a previously unfamiliar architecture usually is a tedious task. One often tries to make a design that is very general, wanting to solve all kinds of problems. This is dangerous, since the result may become too general and thus almost useless. One way to prevent such a pitfall, is to use one domain in the real world and building a system to support some of the activities in this domain. After the system is built and proved to function well, the general architecture may be extracted from the design of the system [WJ98].

To motivate our research, let us take a real world case about the transport of freight. To transport a package from for example Amsterdam, in the Netherlands, to London, United Kingdom, several organizations are involved that have to coordinate their actions. We would like to design an information infrastructure to support the coordination of these actions. Eventually this may result in a prototypical software application. We assume each organization has some computer system and the possibility to connect this system to a computer network. So each computer can be contacted by each of the others. The application should be installed on a computer of each organization (see Figure 1.1) and the resources (fixed schedules, flexible transport means), costs and preferences of the organization should be entered into the application. This private information stays inside the local computer system and therefore inside the organization. Only information that is really necessary to coordinate the transport is exchanged with others. When a customer calls in by phone and requests the transport of a certain package from Amsterdam to London, this order is entered into the application by a secretary or may even be entered by the customer himself. The software processes this information: it plans an efficient way to transport the packages and calculates the costs. Of course it is up to the policy of the organization to decide what to bill to the customer.

Right now, we do not deal with the problem of how the application magically constructs a feasible plan. That is the topic of our research of the coming years.[1] Neither do we deal with the way another facility of the application may be implemented: when an order is canceled or a transport delayed, the involved parties should be informed automatically. If re-planning is necessary, the application should warn the user (read secretary) and propose a set of adaptations. Since re-planning sometimes might be very costly, we will require some robustness in the schedules generated by the applications. If there is some slack[2], some disturbances will not immediately cause large changes in the schedules.

Such a coordinating planning system compared to traditional scheduling systems should improve transport organizations on a couple of points. First of all, it handles also re-planning and rescheduling. Second, it deals quickly with negotiating with other organizations about their free slots and prices. Finally, it might find more efficient solutions because of the extended possibility (due to fast negotiations) to cooperate with others.

We will try to extract a more general structure from this system by abstracting from domain dependent details. We expect to get structure or at least guidelines for a system architecture that can be used in other domains, like multi-modal person transport or army campaign planning.

[1] We might think, for example, of the following: First the application tries to schedule the transport using only the resources of the organization itself. If this fails, or if this leeds to a very inefficient solution, the application may contact others and request them to handle part, or maybe even all, of the job.

[2] Maybe the program can learn itself what amount of slack is optimal.

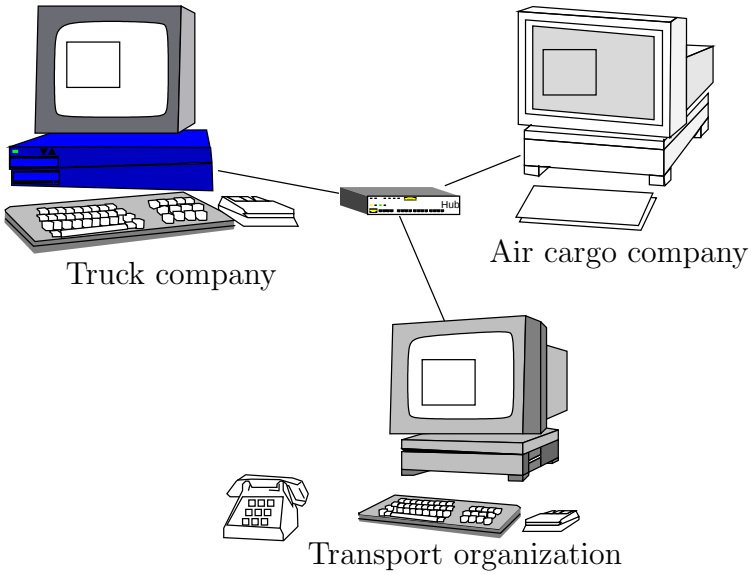

Figure 1.1.: The network of three transport organizations. Instead of the secretaries calling each other on the phone several times a day, their computers are connected and exchange requests and orders.

1.2. Approach

Eventually, we like to have a distributed system to plan, schedule, re-plan, reschedule, and control the dynamic process of freight transport. To achieve this, we take the following approach (that is shown in Figure 1.2). First, we give a precise problem description, so we exactly know what we will deal with and which aspects are irrelevant or not important. We would like to express this problem in a suitable formalism. So the next step is finding a formalism for this description, that can also be used for other, similar problems that might be handled by our techniques. Once we have such a framework we should determine the hardness of the problem. If the problem is fairly easy and we can prove this, we should be able to find existing algorithms that can find the optimal solution within reasonable time. If the response of the system is time-critical, we may want to use a parallel computer or some kind of approximation to get the results in time. But if the problem can be proven to be NP-hard, we know that no algorithms exist that can always give an optimal solution within reasonable time. This will lead to another approach to the problem, in which finding the optimal solution is not the main issue.

To prevent reinventing the wheel, we should investigate existing solutions to this or similar problems. For example one might think of supply chain scheduling (as studied by D. Kjenstad [Kje98]) or the existing load exchange auctions on the internet (such as FreeCargo[Fre99], TranspoWeb[Tra], TimoCom[Tim]). Other topics for investigation are systems that are built in a way we think this system should be built. For example, we may look at multi-agent technology in combination with dynamic planning and scheduling (like in [WM98], [PBC+96]). We should also look at algorithms that solve dynamic scheduling problems in a central way. For example, Stankovic describes such problems (in [SSRB98]). The last important topic we like to mention here is finding criteria to evaluate the resulting system. For example, criteria may be the quality of the resulting transport solutions, the response time for replanning, the

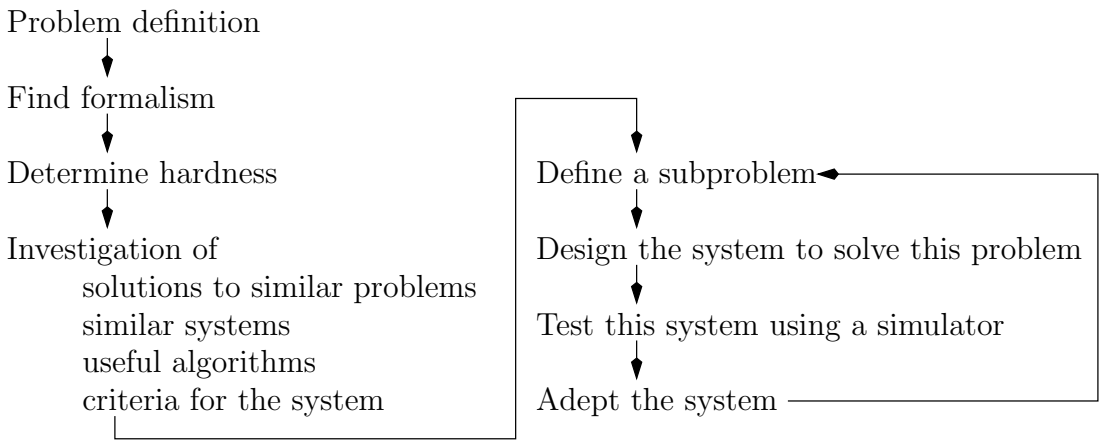

Figure 1.2.: Our approach to designing a freight transport scheduling system.

reusability of the used technology, or the robustness of the resulting schedules.

The next step in our approach is defining a strong simplification of our problem and try to build a system to solve it. We may, for example, assume that all information can be processed centrally and no disturbances will occur. This simple system is to be tested by a simulator. We plan to use the MARS simulator [BdW99]. We expect these tests will lead to some improvements of this early version of the system. Then we enter a cyclic process in which we extend the problem description, then add the required functionality to the system, test the whole system and make the necessary improvements. In the last stage we might also require our system to be open, meaning that everyone can join or leave at any time.

1.3. This report

This report should function as the basis for our research project. We just described the aims of the project, and introduced our problem. In the next chapter we give a very detailed problem description and start a discussion about a suitable formalism or framework. In Chapter 3 we determine the hardness of this problem. And Chapter 4 gives a short overview of existing tools and theories and it also presents first results of solving a very simple version of our problem. In this simple version we assume that no incidents occur and that one agent can perform all the planning and scheduling tasks. In the final chapter we propose a way to continue this research project.

2. Problem description

In this chapter we give an intuitive way to describe the problems we would like to solve. We start by giving some real life problems we expect to have the same structure and therefore may be solved by the same method. In Section 2.2 we show the method to describe such a problem by specifying relevant sets. This method can be thought of as a language in which problems like these can be described. In Section 2.4 we show how to use such a description toe formulate the problem mathematically. The goal of our project is to define a class of problems we would like to find solutions to, to develop a framework that is at least general enough to describe all problems of this class and, last but not least, develop methods to solve the problems described in this framework. We solve these problems by translating them in such a way that they fit in this framework, then using our developed methods to deal with the problem and finally translating the solution from the framework to the real case. The description of problems in the framework is not very easy to use in a solver and is still domain dependent. In Chapter 3 we give a more abstract description that can be extracted from this intuitive description to be able to prove the complexity. Other (more abstract) descriptions can be found in Chapter 4 and are more fit as input for some general solvers.

2.1. Real cases and similar problems

Several problems encountered by organizations and industries are familiar with each other. Often, such a problem involves planning, scheduling and control of scarce resources. Our approach is especially suited for those cases where this process is physically distributed. In this section we describe some of these cases.

2.1.1. Air cargo transport scheduling

An air cargo transport company gets a number of orders to transport packages from a source to a destination [ZLvdH98]. In general these package are sent with the passenger airplanes, so the flight schedules are fixed, that is, to the air cargo department. So the problem description consists of

- a network of airports
- a (often weekly) schedule of legs (when airplanes fly where with which capacity)
- a list of orders: required capacities, releatrainsse and deliver times and places of packages

The goal is to assign packages to legs such that all packages reach their destinations before their deliver times are exceeded. An air cargo company has the capability to add a very limited amount of extra flights or to pass some short-distance orders on to a truck company.

Disturbances like delayed or removed flights and withdrawn orders occur rather often. In a more detailed algorithm the cranes, warehouses and (de)consolidation centers should also be scheduled.

Some complexity is added by the following facts. Currently, about 40 percent of the schedule is changed during the day. The unload and reload time is very short, about 35 minutes, so delays early in the morning propagate through the whole day.

2.1.2. Multi-modal freight transport

Suppose several transport organizations would like to combine their resources to transport goods. Then we have

- a network of harbors, airports, railway stations and (rail)roads, etc. to connect them
- a fixed schedule of most airplanes, trains and boats
- information about cranes, trucks and warehouses
- a list of orders, requesting packages to be picked-up and delivered
- a set of transportation devices such as trucks, airplanes, trains

The problem is how to create schedules for all transportation devices such that all packages reach their destinations in time. This should be done in an cost-efficient way and robust, such that rescheduling due to incidents and additional orders does not inflict to many changes in the schedules. An additional difficulty is that most organizations do not want to exchange sensitive information.

2.1.3. Public transport planning

A similar problem is that of a personal multi-modal transport planner. How to get to your destination using the fixed schedule of trains and busses and the possibilities of more flexible transport types like your own bike, by feet or call-a-ride transport like cabs? Unfortunately, some trains or busses might be delayed or even don't drive at all. So during the journey some alternatives should be considered. This problem seems easier than the multi-modal freight transport, because in general public transport organizations are more prepared to exchange all relevant information.

Other similar problems are:

- Army resource planning
- Multi-depot vehicle routing
- Multiple vehicle pickup and delivery planning

When optimizing is not of utmost importance, but finding a feasible solution is.

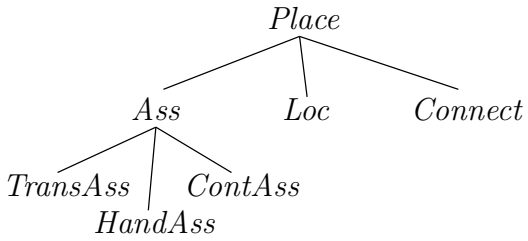

Figure 2.1.: Type hierarchy of *Place*

2.2. General description

The relevant aspects of the problems described in the previous section can be captured in a set of variables and functions on these variables. We propose a method to make such a description in a clear way. First we distinguish all relevant objects in the problem domain and define their types. We see a type as a set of all the objects considered to be of the same sort. Then we describe the properties of the objects using a couple of functions and finally describe the goal of the system in terms of the objects and their properties. We show how to do this in this Section using a specific problem, the multi-modal freight transportation problem. Dean and Greenwald (in [DG92]) used a similar way to describe their transport problems.

2.2.1. Types

We start by defining a couple of types of the objects in the problem environment. The first type we define is $Time$. Since we deal with time tables for some of the modalities and since some packages have explicit deadlines, time is an important aspect of this problem.

- $Time$ is the type of both time points and the length of time intervals.

Locations may be, for example the airports, the transshipment points to get goods from one modality to another, harbors, pickup-, and deliver-points, but also other important facilities where packages are likely to pass through like customs or warehouse locations. These locations are connected by roads, railways, legs, ferry lines, etc. As long as it is possible for an asset to travel from one location to another without stopping at another location, this is considered a connection.

- Loc is the set of all locations.
- The set of all possible connections between the locations is denoted by $Connect \subseteq Loc \times Loc$.
- The set of all packages is denoted by $Pack$: these are the goods which should be transported.

Assets are resources that are able to perform actions on the packages. We distinguish three types of assets. First of all we have the transport units ($TransAss$) which are able to move packages. Transport units are for example airplanes, trucks, ferries, trains, etc. Second we have handling stations ($HandAss$). These assets are usually located at a fixed position and

perform actions like loading, unloading, consolidating, storing, etc. These handling assets are for example cranes, (de)consolidation centers and warehouses. Finally we have a set of containers which contain a set of packages ($ContAss$) like air-freight or euro pallets or sea containers that are standardized to make them easier to handle. To show the hierarchical relation between these types we represent these relations by a tree in Figure 2.1. To each of the asset types belongs a set of actions that can be performed by an asset of that type. These three types of actions are distinguished in the same way. So $TransAction$s which are typically actions like *move* can only be done by asses of the $TransAss$ type, $HandAction$s are actions like load, unload, consolidate and the set of $ContAction$s is possibly empty: containers can perform very few actions. Maybe an action like *collapse* can be put into this set to model the incident of a collapsing container or pallet.

- The set of all assets is called $Ass = TransAss \cup HandAss \cup ContAss$.
- $Action = TransAction \cup HandAction \cup ContAction$ is the set of all possible actions. In Section 2.2.3 we introduce the definition of the current state of the world. An action performed in the world changes this state.

A package can be at a location or inside an asset. An asset can be at a location, a connection or even another asset. We say an asset can be at a place. We define the set of all places as a union of those three types. We also define a super type called Obj to denote all objects.

- We have the set of all places: $Place = Ass \cup Loc \cup Connect$.
- The set of all objects: $Obj = Place \cup Pack$.

The execution of orders: the transportation of packages is controlled by one or more transport organizations. We name the controlling entity agent. Therefore we have the following type:

- the set of control agents: $Agent = \{A_1, \ldots, A_n\}$

Each agent can give load and unload instructions, administration orders, maintenance orders, (de)consolidation instructions, movement orders, and so on, to some of the assets. An instruction is the request by an agent to let an asset perform an action at a specific time.

- The set of all possible asset instructions is given by $Instr \subseteq Time \times Ass \times Action$. The set of all possible asset instructions which can be given by agent A_i is denoted by $Instr_{A_i} \subseteq Instr$.

Clients of the transport organizations can place transport orders. Usually they give a set of orders requiring a set of packages to be delivered at a specific place at a specific time. All orders include information about the package, the pick-up place and time and the required deliver place and deadline.

- $Order = Pack \times Time \times Loc \times Time \times Loc$ is the set of possible orders.

As we like to model the costs made by the system, we introduce a monetary unit: the Euro.

- The type of our currency is denoted by $Euro$.

To describe the dimensions of packages and the distances of connections, we introduce the following type:

- $Length$ for a one dimension and $Volume = Length \times Length \times Length$ to describe a box
- $Speed$ is the type to describe to speed of vehicles in kilometers per hour.
- $Weight$ describes the weight of packages in kilograms.

We distinguish two types of packages that are completely different in the way they can be transported. The type we assumed so far is called 'parts' and consists of all packages that has fixed dimensions and can not be divided into smaller parts. The other type is called 'bulk'. These are materials like oil, wood, coal, stones, sand, and so on, that require special handling and sometimes even special transport devices.

- $PackType = \{\text{bulk}, \text{parts}\}$

The assets can be categorized into a couple of transportation types. These types determine which connections an asset is able to use.

- $TransType = \{\text{road}, \text{air}, \text{bark}, \text{sea}, \text{rail}\}$

We also introduce a type called $Name$ that is used to identify objects. This is usually a name or a number or a combination of both.

- The type $Name$ is a set of possible names. The function $Id : Obj \to Name$ assigns a name to each object for identification.

These types are used to specify more complex types and functions. We start by giving the types of property functions.

2.2.2. Properties

We define property functions on some of the objects in the system. These properties usually induce some additional constraints on the schedule. For each property function we start by giving the type of the properties. The type of location properties is empty. We assume there are no properties of locations that are relevant to the problem. We still introduce this type to allow for future extensions.

- The location property type $LocProp = \emptyset$.
- The property function assigns a property to each location: $locprop : Loc \to LocProp$.

Connections do have interesting properties. They have a transportation type assign to them to tell which type of assets may make use of the connection. Furthermore they have a length, the extra time it may cost to use the connection (in addition to the function of length as specified by the speed of the asset), an amount of fixed costs (such as taxes), and a set of possible departure times representing a time table. If this connection may be used at any time (like a highway), we take for this time-table the empty set.

- Connection properties: $ConnectProp = TransType \times Length \times Time \times Euro \times 2^{Time}$.
- $connectprop : Connect \rightarrow ConnectProp$ is the function that assigns the properties to a connection.

The properties of packages that are relevant to our problem are the volume and the weight of the package. Furthermore it is important whether the package is compositionable, which assets are able to handle the package and what the type of the package is. We assume the packages are introduced in the system as unsplittable parts, so if a customer requests a large package to be transported, it is represented as a set of packages in a container.

- The set of package properties $PackProp = Volume \times Weight \times Bool \times 2^{Ass} \times PackType$.
- The package property function: $packprop : Pack \rightarrow PackProp$.

For assets the capacity is very important. Furthermore, we specify the travel speed in kilometers per hour (if the asset is able to move itself), the transportation type and the actions the asset is able to perform.

- The asset properties type $AssProp = Volume \times Speed \times TransType \times 2^{Action}$
- and the corresponding function $assprop : Ass \rightarrow AssProp$.

For each possible action-asset combination we have an estimated duration and a costs.

- So we have a type $InstrProp = Time \times Euro$
- and a function $instrprop : Ass \times Action \rightarrow InstrProp$.

These properties are important when creating a schedule for the assets and the packages. First we show how to represent a state of the world and then how such a schedule looks like .

2.2.3. States

We would like to reason about the world and we would like to build a simulator. For both we need a model of the world. We describe the world at a specific moment t by a state s_t. We model the dynamics of the world as a transition from one state to another. In the state we capture all the relevant, non-static parts of the world. Therefore, it is exactly the information that should be updated by a simulator. The set of all states is denoted by S. A state is denoted by $s_t = (container, place, costs, leftorders)$ and the mentioned functions are defined as follows:

- The container function $container : Pack \rightarrow Ass \cup Loc \cup \{trash\}$ describes in what asset or at which location a package is stored and since when. The location $trash$ is used to denote that the location of the package is not relevant or known to the system (anymore).
- The place function $place : Ass \rightarrow Place$ assigns a place to each asset and the time the asset arrived at that place.
- Furthermore we have the current time $t \in Time$ and the current costs $costs \in Euro$.

- $leftorders \subseteq 2^{Order}$is a set of orders that still have to be completed.

The state of the world according to an agent might be different from the state of the world, because its view on the world is limited, but it is of the same type (S), augmented with a schedule made by the specific agent. The state of the world according to an agent A_i can be denoted by $s_{t,A_i} = (container_{A_i}, place_{A_i}, costs_{A_i}, leftorders_{A_i}, schedule_{A_i}) \in S_A$.

- A $schedule_{A_i} \subseteq Schedule = 2^{Time \times Instr}$ is a time-table of instructions that will be given to the assets at the given time by agent A_i. Remark that an instruction comprises an execution time point and that the times in a schedule are the time points at which the instruction is given to the asset. $schedule_{A_i}$ is the result of a scheduling function: $scheduler_{A_i} : S_A \rightarrow Schedule$.

This scheduling function should be implemented in some way inside the agents. In Chapter 4 we consider some ideas that might be used to do this.

The dynamics of this system is described by a simulator, which simulates the control commands given by a schedule within a certain time interval. It simulates also events like arrivals of assets at locations and arrival of new orders within this time interval. The start (current) time is part of the input (current) state. The resulting state is the state of the system at the requested time. The simulator in fact implements the effect of actions.

- $simulator : S \times 2^{Instr} \times Time \rightarrow S$ is the simulator function.

In Figure 2.2 the system consisting of a simulated world and one agent that controls the simulated assets is shown. In a system with more than one agent, the scheduler functions of the agents should exchange results as can be seen in Figure 2.3. The orders that enter the system are distributed over the agents such that no two agents get the same order. Every agent observes a part of the state of the world and these parts may overlap. Finally every agent can send instructions to assets in the world. It may be the case that one asset gets instructions from two agents (for example if this asset is a resource that is available to more than one agent), but in most cases each agent sends instructions to its own resources. We think the way the schedulers of the agents exchange partial results is one of the hardest issues of this problem. This is also the main focus of our project.

The problem is how to find a schedule for each agent within the constraints given by the properties, such that all orders are fulfilled.

2.2.4. Restrictions

Although we tried to make this description as extensive as possible, we did make some assumptions concerning the domain. For example we modeled only two types of packages (bulk and parts). Fortunately, it is rather straightforward to extend this to more than two types.

We also simplified the existence of several organizations that would like to work together, but are also each others competitors. We do not know yet what information they would like to give away freely and what they consider sensitive.

Our model of costs is still very primitive. We do not model administration costs for example. But maybe all costs do fit in the general description, since we consider costs as a property of an action.

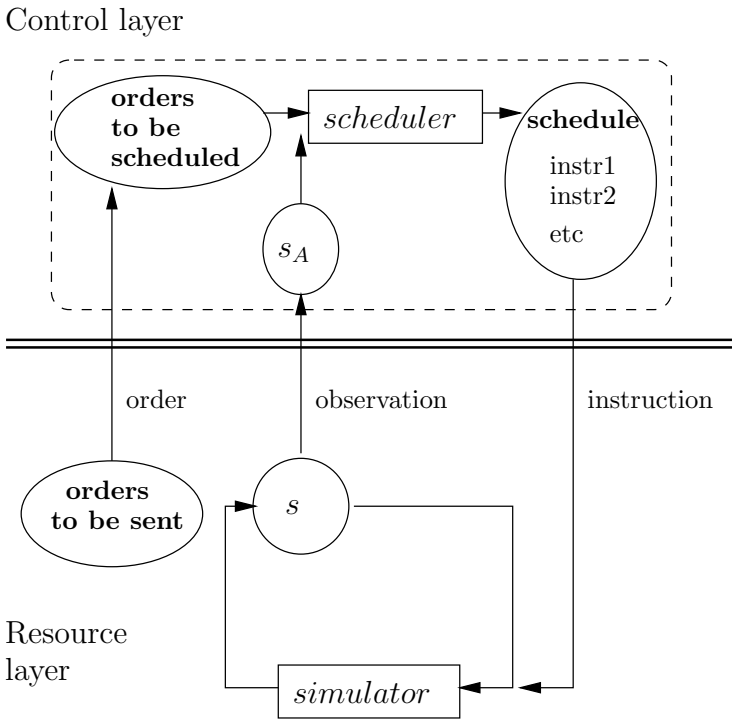

Figure 2.2.: A model of the world (simulator) controlled by one agent.

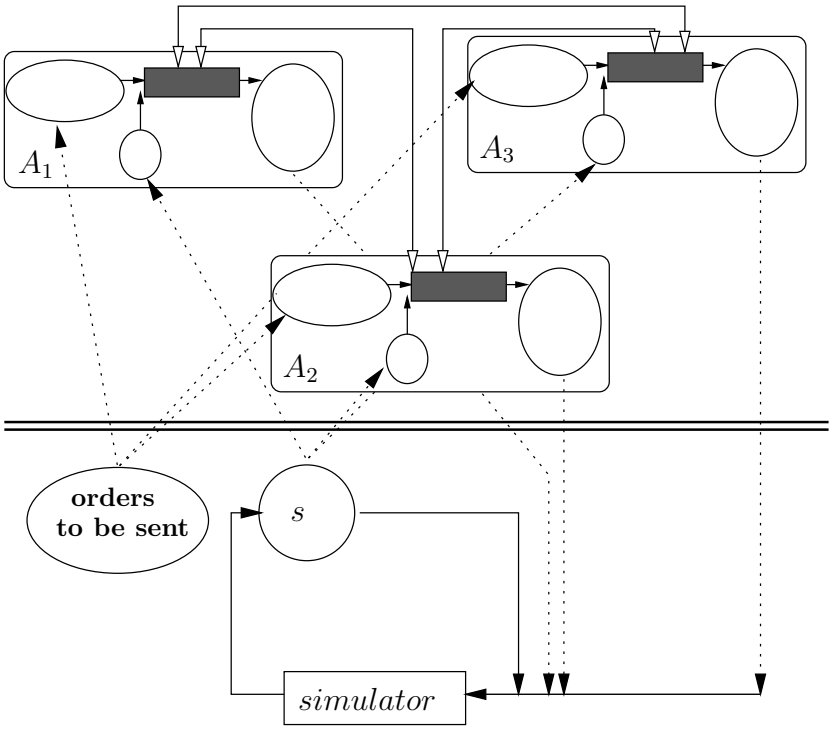

Figure 2.3.: A model of the world (simulator) controlled by more than one agent. The schedulers (gray boxes) exchange intermediate results.

The amount of fuel is not dealt with. We do not plan when a truck should refuel. We do not incorporate when assets should get a maintenance check.

The rest the truck-driver should take is not dealt with: both during long trips and between trips he or she should get the necessary rest. We might let the truck be driven by another truck driver, but this complicates the problem. We also assumed the trucks can stay at a location forever, but in general they return to a garage, sometimes located next to the home of the driver.

We think that these assumptions do not influence the essence of the problem and that once we have solved the problem, it is rather easy to make extensions to the results to incorporate these extras.

2.2.5. Possible disturbances

In real life the environment is dynamic and therefore the problem conditions may change during the day. For example it may be possible that a flight is delayed. In this case the estimated travel time as given by $ConnectProp$ and $AssProp$ is smaller than the real travel time, so the *schedule* made by the responsible agent A_i should be probably adapted to this new situation.

It may also be the case that flights are removed. So in fact the time table in $ConnectProp$ is changed. Or maybe a connection ($\in Connect$) is removed entirely. In this case all schedules (for all i: $schedule_{A_i}$) that used the specified flight should be adapted.

Another disturbance may be that an order ($\in Order$) is withdrawn (a so-called no-show). We assume that for each order one of the agents has responsibility. The concerning agent should remove all instructions that acted on the package of this order form the schedules.

It may also be the case that connections ($Connect$) or assets ($\in Ass$) are added. To get a more optimal solution, all agents may reschedule their actions. This seems only useful when there is still enough time left and a lot of connections or assets are added.

Some locations (Loc) may be unreachable (i.e. due to weather conditions or war). If this is a pick-up or deliver location the concerning order can't be executed. All connections to and from this location should be removed and all schedules which are involved should be adapted.

2.3. Abstract frameworks

In the previous section we gave a detailed description of our problem. This way of describing problems is called a framework. The framework from Section 2.2 is very intuitive. This property makes it rather easy to describe problems in this way. Unfortunately the view this framework gives on the problem still has some elements in it that are very domain dependent. This makes it hard to compare this problem to other problems as we would like to do to be able to say something about the hardness. Another disadvantage of such an intuitive framework is that it is usually not suited as input for general planning tools or problem solvers.

In the next section we give another, more abstract framework and show how a simple version of our problem can be described using that formalism. This formulation enables us to compare our problem to other problems in computer science that have some of the same properties like pickup an delivery and multiple vehicle scheduling with time-windows [GJ79], because these problems often are also formulated in such a mathematical way.

In Chapter 4 we look among others at a formalism that is used for many planning systems (PDDL). It is often directly used as input for solvers. Figure 2.4 shows the relation between

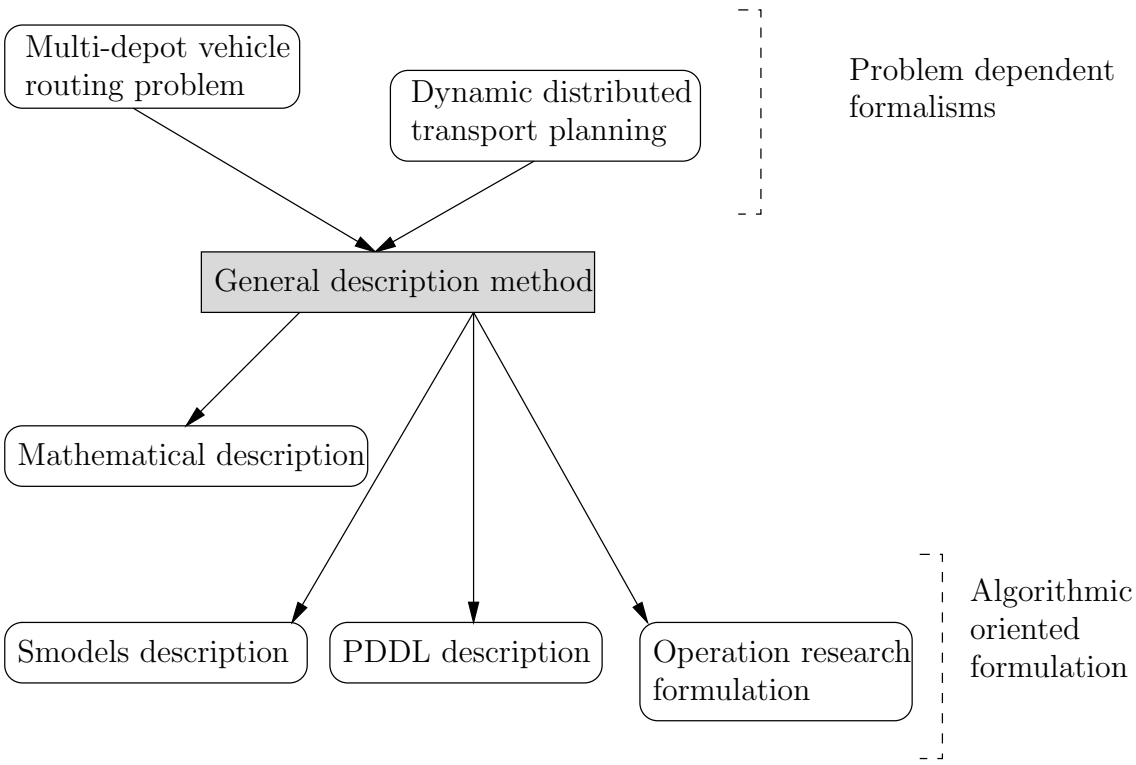

Figure 2.4.: The relation between several descriptions.

these descriptions: the general description method from this chapter can be used to describe all sorts of problems and such a description can be translated to descriptions that are suited to a particular way of solving the problem.

2.4. The General Transportation Problem (GTP)

In this chapter we define a general transportation problem (GTP) and show several special cases of this problem, such as the general pickup- and delivery problem (which has in turn several special cases), the multiple depot vehicle scheduling problem and the vehicle routing problem with time windows. The problem is based on the description in Section 2.2: Given a set of transportation orders and a set of vehicles, how can the packages be transported from their origins to their destinations within their deadlines (soft or hard). We want to be able to deal with costs, capacities, distances and connections depending on the type of the vehicle. We also want to allow orders be executed by more than one vehicle, to take the consolidation and deconsolidation of packages into account, and to cope with the cost of storing packages in warehouses. Furthermore we want to allow fixed schedules for some of the vehicles and a representation of transport services of other organizations, so the formulation also fits in a distributed environment. Finally, we also take a look at the dynamic variant.

In the first section we represent this problem in a mathematical way, like many other problems of this kind are described (for example the general pickup-and delivery problem in [SS95]). Whenever possible we relate the defined sets to the functions, types and objects from Section 2.2. In the second section we sketch the hierarchy of problems that are special cases

of the GTP. Then we discuss the consequences of having a dynamic environment. The last section gives an overview of solution methods found in the literature.

2.4.1. Problem formulation

The General Transportation Problem (GTP) is about constructing routes to satisfy transportation requests. A fleet of vehicles is available to operate the routes. Each vehicle has a given capacity, start location and an end location and its own set of connections associated with costs, times and distances. Each transportation request specifies the size of the load to be transported, the locations where it is to be picked up (origins), the earliest pickup time, where it is to be delivered (destinations), and the latest delivery time. Each load has to be transported by one vehicle from its set of origins to its set of destinations with or without any transshipment at other locations. In general we have one origin and one destination, but we formulate these as sets to allow one origin and multiple destinations and to allow one destination and multiple origins, such as in the multiple depot vehicle scheduling problem.

Let N be the set of transportation requests. For each transportation request $p \in N$, a load of size $\bar{q}^p \in \mathbb{N}$ has to be transported from a set of origins N_p^+ to a set of destinations N_p^-. The load should be picked up after an earliest pickup time t_p^+ and should be delivered at the destination(s) before the latest delivery time t_p^-. Each load is subdivided as follows: $\bar{q}^p = \sum_{j \in N_p^+} q_j^p = -\sum_{j \in N_p} q_j^p$, i.e. positive quantities for pickups and negative quantities for deliveries. Define $N^+ = \cup_{p \in N} N_p^+$ as the set of all origins and $N^- = \cup_{p \in N} N_p^-$ as the set of all destinations. Let $V = N^+ \cup N^-$. Furthermore let M be the set of all vehicles. Each vehicle $k \in M$ has a capacity $Q_k \in \mathbb{N}$, a start location k^+ and an end location k^-. Define $M^+ = \{k^+ \mid k \in M\}$ as the set of start locations and $M^- = \{k^- \mid k \in M\}$ as the set of end locations. Let $W = M^+ \cup M^-$. Let O be the set of all transshipment points and let $L = V \cup W \cup O$ be the set of all locations. For all $i, j \in L$ and $k \in M$ let a_{ijk} denote whether k is able to travel from i to j, d_{ijk} denote the travel distance, t_{ijk} the travel time, and c_{ijk} the travel costs.

To represent fixed schedules we introduce a parameter x_k for each vehicle $k \in M$, that is one if and only if k has a fixed schedule. The route of the schedule is stored in a parameter x_{ijk} for $k \in M$, $i, j \in L$, that is is one if and only if k travels from i to j according to its schedule. The departure times of vehicle $k \in M$ from location $i \in L$ are stored in the sets T_{ik}. More than one element in such a set means that the vehicle k returns to the location i according to its schedule. All the parameters of the problem are shown in table 2.1.

We formulate the solution by minimizing a cost function over all possible transportation plans. To define the transportation plan we need several other concepts.

Define for each request $p \in N$ and for each vehicle $k \in M$ a set of **transshipment deliver points** $O_{pk}^- \subseteq O$, and a set of **transshipment pickup points** $O_{pk}^+ \subseteq O$. We require $\left|O_{pk}^-\right| \leq 1$ and $\left|O_{pk}^+\right| \leq 1$: so we do not allow multiple transshipments of the same package by the same vehicle. Furthermore we require a transportation plan to be such that the vehicle routes are connected at these transshipment points, so that the packages really can be transfered from one vehicle to another.

How should we specify which vehicles pickup which part of the load? Right now load from one destination is transported by one vehicle, as in GPDP.

A **transportation route** R_k is a list of locations $[i_1, i_2, \ldots, i_n] : L^*$such that:

1. $i_1 = k^+$(start from the initial location)

Not.	Meaning
N	set of transportation requests
N_p^+	set of origins of request $p \in N$
N_p^-	set of destinations of request $p \in N$
t_p^+	earliest pickup time
t_p^-	latest delivery time
N^+	set of all origins ($\cup_{p \in N} N_p^+$)
N^-	set of all destinations ($\cup_{p \in N} N_p^-$)
V	set all origins and destinations
M	set of all vehicles
k^+	start location of vehicle $k \in M$
k^-	end location of vehicle $k \in M$
W	set of all start and end locations ($\{k^+, k^- \mid k \in M\}$)
O	set of all transshipment points
L	set of all locations ($V \cup W \cup O$)
$\bar{q}^p$	size of request $p \in N$
q_j^p	flow of request $p \in N$ at location $j \in L$ (> 0 is pickup,< 0 is delivery, $= 0$ otherwise)
Q_k	capacity of vehicle $k \in M$
a_{ijk}	$\begin{cases} 1 & \text{if vehicle } k \in M \text{ is able to travel from } i \text{ to } j \\ 0 & \text{otherwise} \end{cases}$ with $i, j \in L$
c_{ijk}	costs of traveling from i to j for vehicle $k \in M$ with $i, j \in L$
c_i	costs of storing at $i \in O$
d_{ijk}	distance between i and j for vehicle $k \in M$ with $i, j \in L$
t_{ijk}	time needed to travel from i to j for vehicle $k \in M$ with $i, j \in L$
x_k	$\begin{cases} 1 & \text{if vehicle } k \in M \text{ has a fixed schedule} \\ 0 & \text{otherwise} \end{cases}$
x_{ijk}	$\begin{cases} 1 & \text{if vehicle } k \in M \text{ has arc } (i,j) \text{ in its schedule} \\ 0 & \text{otherwise} \end{cases}$
T_{ik}	the departure times of vehicle $k \in M$ from location $i \in L$ according to the fixed schedule

Table 2.1.: Notations of sets and parameters used in the GTP

2. For all $p \in N$: one of the following five holds (all (b-e) or none (a) of the load is transported by this vehicle) *Do we really want this?*

   a) $\left(N_p^+ \cup N_p^- \cup O_{pk}^+ \cup O_{pk}^-\right) \cap R_k = \emptyset$

   b) $\left(N_p^+ \cup N_p^-\right) \cap R_k = \left(N_p^+ \cup N_p^-\right)$

   c) $\left(N_p^+ \cup O_{pk}^-\right) \cap R_k = \left(N_p^+ \cup O_{pk}^-\right)$

   d) $\left(O_{pk}^+ \cup N_p^-\right) \cap R_k = \left(O_{pk}^+ \cup N_p^-\right)$

   e) $\left(O_{pk}^+ \cup O_{pk}^-\right) \cap R_k = \left(O_{pk}^+ \cup O_{pk}^-\right)$

3. If $\left(N_p^+ \cup N_p^- \cup O_{pk}^+ \cup O_{pk}^-\right) \cap R_k \neq \emptyset$, then all locations in $N_p^+ \cup O_{pk}^+$ are visited before all locations in $N_p^- \cup O_{pk}^-$. *Do we really want this?*

4. The vehicle load never exceeds Q_k. The initial load and the end load should be zero.

5. $i_n = k^-$(end in the end location)

6. $a_{i_l i_{l+1} k} = 1$ for all $1 \leq l < n$ (travel only be links that are accessible to the vehicle k)

7. each request is picked up after its earliest pick-up time and delivered before its latest delivery time

A **transportation plan** is a set of routes $R = \{R_k \mid k \in M\}$ such that:

1. R_k is a transportation route for vehicle k for each $k \in M$.

2. $V \subseteq \cup_{k \in M} R_k$ (all origins and destinations are visited)

3. for each $p \in N$ and each $k_1 \in M$: if $O_{pk_1}^+ \neq \emptyset$ then $\exists K \subset M$, $k_1 \notin K$, $\cup_{k \in K} O_{pk}^- \supseteq O_{pk_1}^+$ (all pickups from transshipment points are delivered there by other vehicles)

4. for each $p \in N$ and each $k_1 \in M$: if $O_{pk_1}^- \neq \emptyset$ then $\exists K \subset M$ $k_1 \notin K$, $\cup_{k \in K} O_{pk}^+ \supseteq O_{pk_1}^-$ (all deliveries to transshipment points are picked up by other vehicles)

5. each request is delivered at a transshipment point before it is picked up

If we now define $f(R)$ as the price of plan R, being some function of R and the variables t_{ijk}, c_{ijk}, and d_{ijk}, then the general transportation problem is to find a plan R such that $f(R)$ is minimized.

To formulate GTP more precisely, as a mathematical program, we introduce four types of variables: x_{ijk}^p is used to describe that request $p \in N$ is transported by vehicle $k \in M$ from i to j $(i, j \in L)$. To denote that a load $p \in N$ is transfered at a certain location $i \in L$, we use a variable o_i^p. The variable y_{ik} is used to specify the load size of vehicle $k \in M$ arriving at location $i \in L$. Finally, t_{ik} is the departure time of vehicle $k \in M$ from location $i \in L$. All these variables are shown in table 2.2. A complete schedule for all vehicles can be created once we have found the values for these variables.

Empty travels can not be represented!

The mathematical program looks as follows:

Not.	Meaning
x_{ijk}^p	$\begin{cases} 1 & \text{if request } p \in N \text{ is assigned to vehicle } k \in M \text{ from } i \in L \text{ to } j \in L \\ 0 & \text{otherwise} \end{cases}$
o_i^p	$\begin{cases} 1 & \text{if request } p \in N \text{ is transferred at location } i \in L \\ 0 & \text{otherwise} \end{cases}$
y_{ik}	the load of vehicle $k \in M$ arriving at location $i \in L$
t_{ik}	the departure time of vehicle $k \in M$ from location $i \in L$

Table 2.2.: Notations of the variables used in the GTP

$$
\begin{array}{ll}
x_{ijk}^p \leq a_{ij}^k & \text{for all } i,j \in L,\ k \in M,\ \text{and } p \in N \text{ (only travel when possible)} \\
\sum_{i \in L} \sum_{k \in M} x_{ijk}^p = \sum_{i \in L} \sum_{k \in M} x_{jik}^p & \text{for all } j \in L \text{ and } p \in N \text{ (all travels are connected)} \\
\sum_{i \in L} \sum_{p \in N} x_{k^+ik}^p = 1 & \text{for all } k \in M \text{ (each vehicle leaves its start point once)} \\
\sum_{i \in L} \sum_{p \in N} x_{ik^-k}^p = 1 & \text{for all } k \in M \text{ (each vehicle reaches its end point once)} \\
t_{k^+k} = 0 & \text{for all } k \in M \text{ (each vehicle leaves at time 0)} \\
x_{ijk}^p = 1 \rightarrow t_{jk} \geq t_{ik} + t_{ijk} & \text{for all } k \in M \text{ and } i,j \in L \text{ and } p \in N \\
 & \text{(departure times are correct wrt the travel times)} \\
y_{k^+k} = 0 & \text{for all } k \in M \text{ (start loads are zero)} \\
\sum_{k \in M} \sum_{i,j \in L} x_{ijk}^p \geq 1 & \text{for all } p \in N \text{ (each request is dealt with)} \\
\vdots &
\end{array}
\tag{2.1}
$$

2.4.2. Related problems

Several famous problems deal with some sort of transport. In this section we described each of these problems and show one by one that they are a special case of the GTP.

In general we expect the special cases to be somewhat less complicated and therefore maybe having better (read: faster) algorithms. These algorithms may be used as initial solutions for for example local search.

The General Pickup and Delivery Problem (GPDP)

The General Pickup and Delivery Problem (GPDP) [SS95] is a special case of the GTP where no pickup and delivery times are specified, no fixed schedules occur, and transshipment is not allowed. Therefore composition, decomposition and storing in warehouses do not have to be dealt with. Furthermore each vehicle is able to travel from each origin to each destination. This has the following consequences for the problem formulation:

1. Fixed values for some parameters:

   a) $a_{ijk} = 1$ for all $k \in M$, $i,j \in L$

   b) $d_{ijk_1} = d_{ijk_2}$ for all $k_1 \neq k_2 \in M$, $i,j \in L$

   c) $c_{ijk_1} = c_{ijk_2}$ for all $k_1 \neq k_2 \in M$, $i,j \in L$

   d) $t_{ijk_1} = t_{ijk_2}$ for all $k_1 \neq k_2 \in M$, $i,j \in L$

e) $t_p^+ = 0$ and $t_p^- = \infty$ (no pickup and delivery time constraints)

f) $O = \emptyset$ (no transshipment points)

g) $x_k = 0$ for all $k \in M$ (no fixed schedule)

2. More constraints on the variables:

   a) $o_i^p = 0$ for all $p \in N$ and $i \in L$

Three well-known problems are special cases of the GPDP. In the Pickup and Delivery Problem (PDP) each transportation request specifies a single origin and a single destination and all vehicles depart from and return to a central depot. The Dial-a-Ride problem (DARP) is a PDP in which the loads represent people. Therefore we speak of clients or customers and all load size are equal to one. The Vehicle Routing Problem (VRP) is a PDP in which either all the origins or all the destinations are located at the depot.

The *pickup and delivery problem* is a special case of the GPDP for $|W| = 1$ and $\left|N_i^+\right| = \left|N_i^-\right| = 1$ for all $i \in N$. In this case we may define i^+ as the unique element of N_i^+ and i^- as the unique element of N_i^-.

For the *dial-a-ride problem* we also take $|W| = 1$ and $\left|N_i^+\right| = \left|N_i^-\right| = 1$, but also $\bar{q}_i = 1$ for all $i \in N$.

The *vehicle routing problem* is also a special case, namely for $|W| = 1$ and $\left|N_i^+\right| = \left|N_i^-\right| = 1$ for all $i \in N$, and $N^+ = W$ or $N^- = W$.

In the general PDP we also deal with a set of origins or destinations and vehicles with different start and end points and transportation requests evolving in real time (dynamic variant). A good overview of techniques to solve these problems can be found in [SS95].

Dial-a-ride problems (DARP) can be divided into two classes: static problems, where the system has a couple of hours to determine a schedule (i.e. for a new day) or dynamic problems where the system has to come up with a new schedule within several seconds. Neural networks can be used to estimate the travel times [FT99].

Multi-depot vehicle scheduling

Another interesting problem that is related to the GPDP is the multiple depot vehicle scheduling problem (MDVSP) [DDSS95]. This problem is usually formulated as follows: Consider a set of n trips $\{T_1, T_2, \ldots, T_n\}$, where trip T_i has a given duration d_i and should start at time a_i for $i = 1, 2, \ldots, n$. Consider also a set of depots K with the kth depot housing v^k vehicles, $k \in K$. Let $N = \{1, 2, \ldots, n\}$ the set of nodes represent the set of trips and let $n+k$ be the kth depot. Let I be the set of pairwise compatible trips, i.e. $(i, j) \in I$ if and only if $a_i + t_{ij} \leq a_j$.

Consider the Graphs $G^k = (V^k, A^k)$ with $V^k = N \cup \{n+k\}$ and $A^k = I \cup (\{n+k\} \times N) \cup (N \times \{n+k\})$ for each $k \in K$. Also costs are associated to the arcs. For arcs $(i, j) \in I$ the costs are the same for each depot k and denoted by c_{ij}. The costs of the arcs between the depots and the network of trips usually are dependent on k and denoted by $c_{n+k,j}$ and $c_{i,n+k}$ for $j \in N$, $i \in N$ respectively.

The problem consists of finding an assignment of vehicles to trips in such a way that:

- each trip is covered exactly once by a vehicle

- each vehicle used in the solution leaves from its depot, covers a sequence of pairwise compatible trips and returns to its own depot at the end of the route
- the number of vehicles leaving from depot k does not exceed v^k
- minimize the sum of the costs of the routes traveled by the vehicles
- or the number of vehicles used (multi-depot minimum fleet size problem (MDFSP))

The MDVSP has been shown to be NP-hard when $|K| \geq 2$. $|K| = 1$ is solvable as a minimum cost network flow problem. When the costs $c_{n+k,j}$ and $c_{i,n+k}$ are independent of the depot k, $k \in K$, the problem reduces to the single depot vehicle scheduling problem. In particular the MDFSP reduces to the single depot case. This is a way to get a feasible solution to the MDVSP in polynomial time. For this problem heuristics exist that can solve problems up to 1000 trips and 10 depots. We refer to chapter 2 in [DDSS95] (page 49) for further references. Lately Andreas Löbel [Löb99] has found a method to solve instances up to 2 thousand trips within a couple of hours to optimality or an approximation of an instance of about 25 thousand trips. In Section 2.4.3 we discuss one dynamic approach to this problem.

Vehicle Routing Problem with Time Windows (VRPTW)

The Vehicle Routing Problem with Time Windows (VRPTW) consists also of finding a set of minimum cost routes (see [DDSS95]). But now the routes should originate and terminate at a central depot, for a fleet of vehicles which services a set of customers with known demands. The customers must be assigned exactly once to vehicles such that the vehicle capacities are not exceeded. The service at a customer must begin within the time window defined by the earliest time and the latest time when the customer permits start of the service. Time windows can be hard or soft.

The Vehicle Routing Problem (VRP) is a special case of the VRPTW without time windows.

The multiple Traveling Salesman Problem with Time Windows (m-TSPTW) is a special case if the capacities are not binding.

2.4.3. Dynamic GTP

Su [Su99] has studied the dynamic variant of the MDVSP. We take his approach as an example of how to deal with dynamic aspects of the environment.

3. Complexity analysis

To get an indication of which techniques might be applicable to the problem we described in the previous chapter, we compare the problem to some problems that have a known complexity. In this way we are able to establish its complexity. We show in this chapter that a simplification of the problem is NP-hard. Once we know that this problem is very hard, we know that we don't need to take approaches that result in algorithms that find optimal solutions in polynomial time. Furthermore we can draw conclusions about how to set our goals. If this problem is NP-complete, it is very unlikely that we end up with a method that returns always perfect solutions in real-time, but we may try to develop one that returns a good solution in most of the cases.

In this chapter we take a very strong simplification. The original GTP is hard to recognize in the described disjoint connecting paths problem (DCPP), but it still holds that GTP is NP-hard.

3.1. A model for transport scheduling problems

First we introduce some notations we use. Then several objects and functions on these objects are defined and some examples are given. Finally we show what properties of the real-life problems are not covered by these definitions.

Notations

We use $\mathbb{N}$ to denote the natural numbers and $\mathbb{R}$ to denote the real numbers. A list is specified between square brackets $[element_1, element_2, \ldots, element_k] : E^*$. The type of a powerset of, for example, the set of edges E is denoted by 2^E.

Definition 3.1.1 We define the function $first : X \times Y \to X$ to return the first element of a tuple: $first\,((x, y)) = x$ and the function $second : X \times Y \to Y$ to return the second element of a tuple: $second\,((x, y)) = y$.

Graphs

Definition 3.1.2 A **graph** G is a tuple (V, E), where V is a collection of vertices and $E \subseteq \{\{x, y\} \mid x, y \in V\}$ is a set of pair-sets of V. An edge between vertices x and y is a pair-set $\{x, y\}$.

We may assume that the graph is fixed. In that case, the consequences for the domain are that no other sources or destinations can be handled than those in the given set of locations and no other connections can be used than the given set of connections.

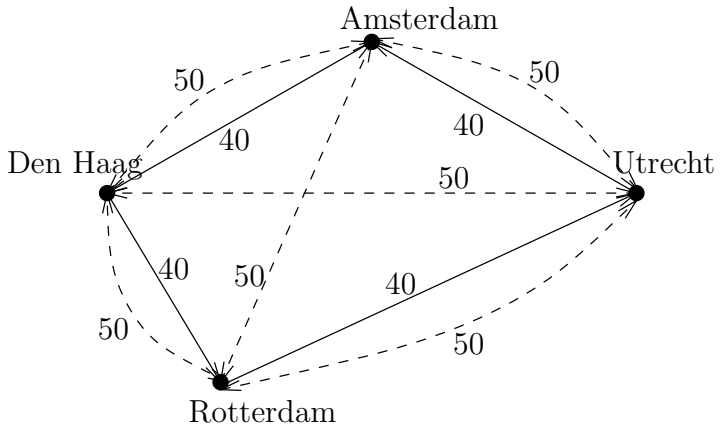

Figure 3.1.: A transport network of four big cities with two transportation types.

In such a graph there is at most one direct connection between any two locations. Since this is not true in general, we define a graph that allows more than one connection between locations, a multigraph.

Definition 3.1.3 A **multigraph** is a graph $G = (V, E)$ where E is a multi-set of edges.

It might be the case that between some locations there is only a direct connection in one direction, for example, because of the flight schedule. Also, the properties of the back connection may differ from the forth one. A directed graph distinguishes the directions of a connection.

Definition 3.1.4 A **directed graph** is a graph $G = (V, A)$ where $A \subseteq V \times V$. We use the functions $from : A \to V$ and $to : A \to V$ to specify the start and end vertex of an arc: $from(e) = first(e)$ and $to(e) = second(e)$.

Definition 3.1.5 A **labeled graph** is a graph in which labels are defined on edges or vertices. Labels are attached constants. The labels we use are $costs_e : A \to \mathbb{R}$ and $cap_e : A \to \mathbb{R}$. If it is clear that these functions are defined on the edges the subscript $_e$ can be omitted and only *costs* and *cap* are used.

A graph can represent a transport network. The vertices represent the locations and an edge between two vertices represents a direct connection between these two locations. An edge that connects a vertex to itself can be used to define the storage or waiting time of a package. We use a directed labeled multigraph to represent the transport network. The multigraph allows us to specify several cost and time length variants or links realizable by different companies between the same two locations.

For example, consider the following situation. We have four big cities: Amsterdam, Den Haag, Rotterdam and Utrecht. Suppose we have cargo trains that travels along these cities in the mentioned order, billing 40 for each track. Suppose furthermore that we also have the possibility to transport the goods by trucks, which is somewhat more expensive (50). This situation is shown in Figure 3.1.

This can be modeled by a directed labeled multigraph $G = (V, E)$ where $V = \{\text{Amsterdam(A), Den Haag(D), Rotterdam(R), Utrecht(U)}\}$ and $E = \{e_1, e_2, \ldots, e_{16}\}$ where

$e_1 = (\text{A}, \text{D})$	$e_5 = (\text{A}, \text{D})$	$e_9 = (\text{R}, \text{U})$	$e_{13} = (\text{A}, \text{R})$
$e_2 = (\text{D}, \text{R})$	$e_6 = (\text{D}, \text{A})$	$e_{10} = (\text{U}, \text{R})$	$e_{14} = (\text{R}, \text{A})$
$e_3 = (\text{R}, \text{U})$	$e_7 = (\text{D}, \text{R})$	$e_{11} = (\text{U}, \text{A})$	$e_{15} = (\text{D}, \text{U})$
$e_4 = (\text{U}, \text{A})$	$e_8 = (\text{R}, \text{D})$	$e_{12} = (\text{A}, \text{U})$	$e_{16} = (\text{U}, \text{D})$

In addition, we specify a label *costs* as follows:

$$costs(e_i) = \begin{cases} 40 & \text{if } i \leq 4 \\ 50 & \text{otherwise} \end{cases}$$

Packages and orders

Now we are ready to introduce packages into the model.

Definition 3.1.6 The finite set of all **packages** is denoted by $\mathbb{P}$. We have a couple of total functions that specify properties of the packages: $src : \mathbb{P} \to V$ specifies the source place of the package, $dst : \mathbb{P} \to V$ specifies the destination, $vol : \mathbb{P} \to \mathbb{R}$ specifies the volume of the package.

Definition 3.1.7 An **order** O is a tuple (P, src, dst, vol), where P is a subset of packages $P \subseteq \mathbb{P}$ and *src*, *dst*, and *vol*, as defined in the previous definition, are functions that specify the required properties of packages from P.

Routes and schedules

Definition 3.1.8 Let $G = (V, A)$ be a directed graph. A **path** or **route** is a sequence of k edges A: $[e_1, e_2, \ldots, e_k : A^*]$ for some $k \in \mathbb{N}$, for which the following hold for each $1 \leq i \leq k-1$: $to(e_i) = from(e_{i+1})$. We call k the **length** of the route. The set of all routes is denoted by $\mathbb{R}$.

Definition 3.1.9 The **capacity** of a route r ($cap_r : \mathbb{R} \to \mathbb{R}$) is the smallest capacity of an edge of the path: $cap_r(r) = min_{1 \leq i \leq k} \{cap_e(e_i)\}$. If it is clear which capacity function is meant, the subscript $_r$ can be omitted.

Definition 3.1.10 The **costs** of a route r ($costs_r : \mathbb{R} \to \mathbb{R}$) is the total cost of all the edges of the route: $costs_r(r) = \sum_{i=1}^{k} costs_e(e_i)$.

Definition 3.1.11 Let $G = (V, A)$ be a directed labeled graph. Let $O = (P, src, dst, vol)$ be an order. We define a function $pkgs : \mathbb{R} \to 2^{\mathbb{P}}$ that gives the set of **packages** that use the route in the graph G, such that for any route $r = [e_1, e_2, \ldots, e_k]$, it holds for each package $p \in pkgs(r)$ that $first\,(src(p)) = from(e_1)$ and $first\,(dst(p)) = to(e_k)$ and for a timed route $r = [(t_1, e_1), (t_2, e_2), \ldots, (t_k. e_k)]$, it holds for each package $p \in pgks(r)$ that $src(p) = (from(e_1), t_{src})$, $t_{src} \leq t_1$, $dst(p) = (to(e_k), t_{dst})$, $t_k + time(e_k) \leq t_{dst}$. Of course this

function should also be defined such that $\sum_{p \in pkgs(r)} vol(p) \leq cap(r)$. We define the left-hand side of this equation to be the **volume** of the route r ($vol_r : \mathbb{R} \rightarrow \mathbb{R}$). This is the total volume of all the packages that are assigned to the route.

Definition 3.1.12 A feasible pkgs function

Plans

Definition 3.1.13 Let $G = (V, A)$ be a directed labeled graph. Given an order O and a time table *timetable*, a **plan** is a tuple $(R, pkgs)$ where R is a set of timed routes and $pkgs$ assigns packages to these routes, such that for each $e \in A$: $\sum_{r \in R} vol(r) \leq cap(e)$.

3.2. The optimization problem

Given a directed graph $G = (V, A)$, a cost function for each arc $\$: A \rightarrow \mathbb{R}$, a capacity function for each arc $cap : A \rightarrow \mathbb{R}$ (with $\forall_{a \in A} cap(a) = 1$), a set of packages P, a (dynamic) set of orders $O \subseteq P \times (V \times V)$, find a planning $c : P \rightarrow 2^A$ such that for all $(p, (v_1, v_2)) \in O$ (with $v_1 \neq v_2$) the following holds:

- $\exists v_i : (v_1, v_i) \in c(p)$ The release point for package p is included in the planning for p.
- $\exists v_j : (v_j, v_2) \in c(p)$ The deliver point for package p is included in the planning for p.
- $\forall (v_i, v_j) \in c(p) : (v_i = v_1 \vee (\exists v_k : (v_k, v_j) \in c(p))) \wedge (v_j = v_2 \vee (\exists v_k : (v_i, v_k) \in c(p)))$ Each edge in a planning has a subsequent edge or reaches the deliver point and has a prior edge or started at the release point.
- $\forall_{a \in A} \left(\sum_{p \in \{p | p \in P \wedge a \in c(p)\}} vol(p) \leq cap(a) \right)$ The total volume of all the packages (we assume $vol(p) = 1$) which travel along an edge is less than or equal to the capacity of that edge. In the case that $cap(a) = 1$ this is equivalent to: $\exists_{a \in A} a \in c(p) \rightarrow \nexists_{p' \in P, p' \neq p} a \in c(p')$: an edge is part of at most one package route.

This planning should be chosen such that the total costs are minimized:

$$\text{minimize} \sum_{a \in A} \sum_{p \in \{p | p \in P \wedge a \in c(p)\}} \$(a)$$

Things we did NOT model explicitly
Some of these can be modeled implicitly.

- Several package properties:
  - deterioration
  - length, width, height
  - weight
  - decompositionable

  - parts or bulk
  - unsuitable for some kinds of transport

- connection properties:
  - reliability of time length and costs
  - costs as function of time length
  - maintenance

- distinction between connections and assets
- transshipment time and cost
- initial distribution of packages

The assumptions made to create this simplification of the problem described here seem severe, but they might turn out alright. For example the time and the schedules were omitted, but time, schedules and capacity can be combined in a new type of capacity which represents the maximum number of packages along this arc in a whole cycle (of the schedule). Unfortunately, capacities are also omitted in this simplification. They can be added straightforwardly by replacing each arc with capacity n by n arcs of capacity 1, but this will explode the network for large capacities.

3.3. Disjoint connecting paths problem (DCPP)

Consider the disjoint connecting paths problem ([Kar75]): Given a graph $G = (V, A)$ and a collection of disjoint vertex pairs $(s_1, t_1), (s_2, t_2), \ldots, (s_k, t_k)$, does G contain k mutually vertex-disjoint paths, one connecting s_i and t_i for each i, $1 \leq i \leq k$? This problem has been proven to be NP-complete ([Kar75]) and can be reduced to UFPC.

Theorem 3.3.1 The unsplittable flow problem with costs (UFPC) (as described in Section 3.2) is NP-complete.

Proof

UFPC clearly is an NP problem. (It is solvable by a nondeterministic algorithm in polynomial time.)

NP-hardness (at least as hard as an NP-complete problem) then can be shown by a reduction by local replacement of the disjoint connecting paths problem. Let the graph $G = (V, A)$ and a collection of disjoint vertex pairs $ST = \{(s_1, t_1), (s_2, t_2), \ldots, (s_k, t_k)\}$ with $k \leq |V|$ constitute an arbitrary instance of the DCP problem.

The UFPC problem is specified by: $G' = (V', A')$ where $V' = \{v_j \mid v \in V, j = 1 \vee j = 2\}$ and $E' = \{(v_1, v_2) \mid v \in V\} \cup \{(v_2, w_1) \mid (v, w) \in A\}$, furthermore the cost function $\$' : A' \to \mathbb{R}$ is defined as: $\$'(a) = \begin{cases} 1 \text{ if } a \in A \\ 0 \text{ otherwise} \end{cases}$ and the capacity function $cap' : A' \to \mathbb{R}$ is defined as $\forall_{a \in A'} cap'(a) = 1$. We create a set of packages $P = \{p_1, \ldots, p_k\}$, one for each order and the set of orders $O \subseteq P \times (V \times V)$ is

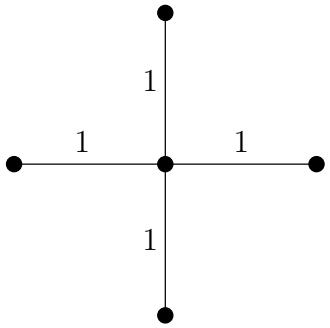

Figure 3.2.: A difficult graph

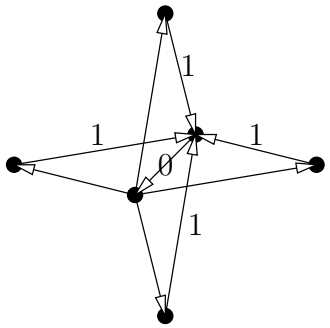

Figure 3.3.: The graph from Figure 3.2 translated to simplification

defined as the collection of a package with a disjoint vertex pairs $(p_i, (s_{i,2}, t_{i,1}))$ (for all $(s_i, t_i) \in ST$). It is easy to see that this transformation can be done in polynomial time. An example of such a transformation can be found in Figure 3.2 and 3.3.

We claim that the disjoint connecting paths problem has a solution if and only if the constructed unsplittable flow problem has one.

Suppose the solution to our problem (UFPC) is a set of k paths. Path p_i starts in s_2 and ends in t_1. Then this solution can also be transformed to a solution to DCPP in the following way: for each path, remove all v_2's from the path except s_2 and rename all v_1's to v. This is a solution to DCPP since

- all arcs in the path are connected (in the graph G), since $(v, w) \in path_i$ only if $(v_1, v_2) \in path$, $(v_2, w_1) \in path$, $(w_1, w_2) \in path$ $\{(v_1, v_2) \mid v \in V\}$
- no vertex v occurs in more than one path. This can be seen as follows: (v_1, v_2) occurs at most once in a path because of capacity 1. The arc (v_1, v_2) corresponds to the vertex v and therefore v also occurs in at most one path.

Since UFPC is in NP and at least as hard as an NP-complete problem, it is a NP complete problem itself.

End of Proof

A reverse problem translation is also possible, but not necessary to prove NP-hardness. It is included here as an illustration.

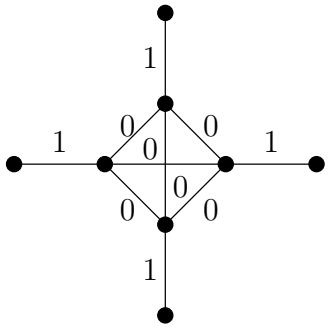

Figure 3.4.: The graph from 3.2 translated to DCPP

Translation from UFPC to DCPP Create for each vertex v a fully connected subgraph of as many vertices as arcs connected to v. Each of the arcs within this subgraph should have cost 0. Each vertex of this subgraph should be connected to precisely one of the arcs connected to v in the original graph. This translation is done so each solution to DCPP will give a solution to the simplification and each solution to the simplification is also a solution to DCPP. An example of a problem graph can be found in Figure 3.2and its translation in Figure 3.4.

Multi-commodity flow relaxation

There exists a relaxation of the UFPC, which is a linear program (LP) and can be solved efficiently [BS97].

Complexity

In Section we saw that the simplified problem is equivalent to the disjoint connecting paths problem. This problem is NP-complete (see [GJ79], page 217). Complexity is open for any fixed $k \geq 2$, but can be solved in polynomial time if $k = 2$ (k is the number of orders).

3.3.1. Solving large instances

Large instances of this problem may be solved by creating a layered structure in the network. For example, the intercontinental flights are represented as links in a network in which the vertices represent the continents. Each of these vertices is in fact a whole network. (Additional problem: how much "delay" in a vertex if the next continental flight on a route leaves from another airport in the same continent.) A leveled architecture might also be useful in a context with more than one organization.

In the next section some attention will be paid to the size of the instances which can be solved.

4. Overview of current technology

The constraints on the problem solving method lead us to several interesting and possibly very useful theories and ideas. In this chapter we present an overview of what we think are possible building blocks for a theory about interorganizational planning. The planning process consists globally of two cycles (see Figure 4.1): A planning and negotiation cycle to accommodate to each others plans and a monitoring and replanning cycle that copes with unexpected events. In this chapter we focus on the first cycle and do not pay much attention to the dynamic aspect of the problem. In the first section we consider several tools that are able to construct a plan using knowledge about the effects of actions. In Section 4.2 several routing algorithms that solve problems that are similar to a central version of our problem are described and their usefulness to us is discussed. Then we discuss some previous attempts to plan and control transport distributedly. This finally leads to an overview of negotiation and auction techniques.

4.1. Planning tools

The goal of this section is not to give a complete overview of all kinds of planning tools, but to give an idea of what sort techniques are available and how they work. A more thorough investigation, including test results, etc. can be found in [dW99]. In this section we describe some different tools and give a short explanation of how they work internally and how they can be used, an example of a problem description in the formalism of the tool, and information about the availability of the source code, and current state and the future of the tool.

4.1.1. Smodels

Two ways exist to implement an interpreter for logic programs. The most used is using a well-founded semantics [vGRS91], but recently Niemela et al. [Nie98][NS97] have implemented

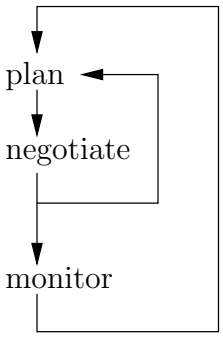

Figure 4.1.: The planning process of one organization.

```
at_London :-.
move_AmsterdamLonden :- at_Amsterdam.
at_London :- move_AmsterdamLondon.
move_ParisLonden :- at_Paris.
at_London :- move_ParisLondon.
at_Amsterdam.
```

Figure 4.2.: An example of a set of rules. Actions are represented by (at least) two rules. Goal is probably also the head of one of the postcondition rules.

both the well-founded semantics and the stable-model semantics in a tool called Smodels. This implementation is able to solve a range of computational problems related to a given normal logic program. First of all, it is able to compute the well-founded model of a program. Second, it can decide whether a program has a stable model. Third, it can generate all or a given number of stable models of a program. Finally, it is able to answer two basic queries: is a given literal satisfied in some or all of the stable models.

The input to the parser part of the tool is a program P consisting of a set of rules. These rules are of the following form. They start with an atom, denoting its head, followed by the inference symbol ':-', in turn followed by the body that is a comma-separated list of atoms and negations of atoms. Smodels will search for a stable model S. What this model is, can be explained as follows. First we define the reduct P^S: this can be seen as the set of potentially applicable rules, given the stable model S and can be obtained from a program P by deleting each rule that has a negative literal $not(c)$ in its body with $c \in S$ and all negative literals in the body of the remaining rules. Furthermore we define the set of consequences of a program P as follows: $Cl(P)$ is the deductive closure of the rules. Using these definitions we can define a stable model: A set of ground atoms S is a stable model if and only if $S = Cl(P^S)$.

We can describe a planning problem by giving a set of rules, such that the resulting stable model represents a plan. We did this for a very specific problem. How we did this will be shown in another report [dW99]. We suffice here with a simple example (see Figure 4.2). To make it easier to specify these rules, there exist some more advanced rules using variables or special constructs that are translated by the parser to the standard rules.

The implementation of the stable model semantics for ground programs in the Smodels system is based on bottom-up backtracking search. Here the search space for stable models is pruned efficiently by exploiting the minimality and groundedness properties of stable models. This is based on an approximation technique for stable models which is closely related to the well-founded semantics. It is in fact using a framework for developing bottom-up methods for computing extensions of default logic.

The latest version is 2.24 of less than a month ago (29 October 1999) and the authors are currently still improving and extending the functionality of Smodels.

4.1.2. Graphplan

Planning is a process of finding a correct action sequence. One can in general execute one action at a time. An action changes the state of the system. Creating a plan can therefore be seen as finding a correct sequence of state changes such that the final state validates the goal requirements. States can be represented by nodes and state changes can be represented by arcs. The process of finding a plan is than equivalent to finding a path in the graph repre-

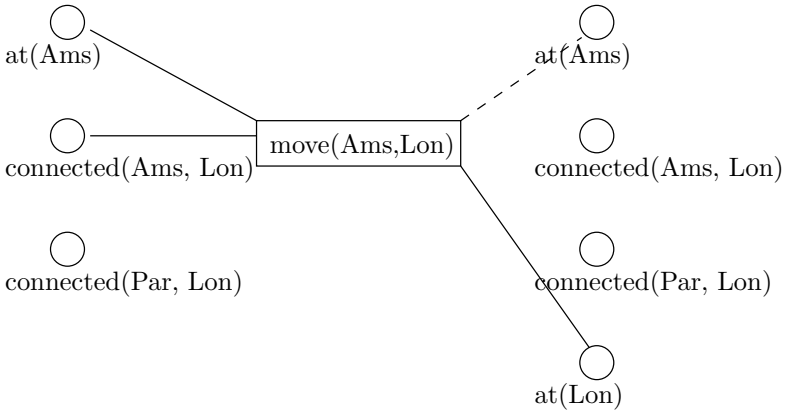

Figure 4.3.: A very simple planning graph with three layers.

```
:init (and (at Amsterdam) (connected Amsterdam London) (connected Paris London))
:goal (at London)
(defschema (move)
:parameters (?x ?y)
:precondition (and (at ?x ) (connected ?x ?y))
:effect (and (not (at ?x)) (at ?y)) )
```

Figure 4.4.: An example of a STRIPS description of the initial situation, the desired goal state and a move action.

sentation from the initial state to the goal state. The number of nodes in this representation will be very large, even for medium and small sized problems, but fortunately there are more efficient ways to specify such a planning graph.

Graphplan (see for example [Wel99]) is a tool that uses a more advanced way to model a planning graph. It creates a directed, layered graph. Two types of layers are interleaved: a proposition layer and an action layer. A proposition layer consists of nodes that each represent an atom or the negation of the atom. The first layer is a proposition layer representing the initial state. The action layer contains a node for each possible action. The nodes are connected by three types of arcs. Precondition-arcs connect the action to the atoms of its precondition of a previous layer. Add-arcs connect an action to the nodes of the next layer, representing an atom that is a direct consequence of the postcondition of the action. Delete-arcs connect actions to the nodes representing atoms that are disabled by the action. In Figure 4.3 such a planning graph can be seen.

Graphplan determines so called mutual exclusion relations for each layer to indicate which atoms can not be true at the same time (layer). To find a correct plan, it builds the graph while searching for a proposition layer that, together with the mutual exclusion relations, implies the goal state. Since the mutual exclusion relation is not complete, the plan must be checked and if not correct, the process continued.

The input to Graphplan is specified using a formalism called STRIPS (Stanford Research Institute Solver) to describe actions and states. States are described by conjunctions of predicates on constants and actions are defined in terms of a complete precondition using predicates and a postcondition consisting of a add-list of predicates and a delete-list of predicates. Figure 4.4 gives an example of how these terms are specified in STRIPS.

Many tools, such as Blackbox (see next section) and IPP [KNHD97a][KNHD97b] are based on Graphplan. Even most of the tools based on a translation to the satisfiability problem use the graph representation we described.

4.1.3. Blackbox

There is no strict order among the available tools. Some are better at some problems, others at other problems. The Blackbox tool by Kautz and Selman [KS98] tries to combine the advantages of both by using several methods together. This tool first constructs a planning graph. This graph is used by the Graphplan algorithm, but also used to create a CNF well founded formula. This formula is used by several Satisfiability solvers (Walksat and Satz). Blackbox gives the included solvers in turn a possibility to solve the problem. The time the solvers get in one turn increases each step.

The input to this solver is also specified in the STRIPS formalism, which is not a surprise, because the Graphplan algorithm is included in this solver. It seems that for a large set of problems, this tool gives very good results.

Unfortunately, this tool is not able to deal with capacity constraints. The following tool, however, is.

4.1.4. LPSat

Compilation to boolean satisfiability has become a powerful paradigm for solving AI problems. However, among others, Wolfman and Weld [WW99c] encountered the problem that, although many AI problems require some sort of metric reasoning, none of the existing tools was able to deal with that. So they tried to combine a SAT solver with an incremental simplex algorithm.

The input is specified in a variant of PDDL[McD98] (Planning Domain Definition Language), which is an extension of STRIPS. Figure 4.5 is an example of a problem specified in this formalism. It shows that it is possible to deal with capacity constraints.

Due to the simplex calculations, it takes quite some time to solve moderate instances (15 packages, 4 cities), but the class of problems that can be solved in this way is much larger than for any of the others. Currently, there is only one version of this tool and the author is sometimes working on it. Unfortunately, the source code is not available.

4.2. Routing algorithms

In operation research several problems are studied that have a close relationship with the ones we are dealing with. These studies usually try to develop algorithms that solve the problem centrally and most of them do not cope with the dynamic aspect, but lately more and more dynamic variants are developed and published. These algorithms are very interesting to us.

Several algorithms exist to solve the problems from the classification from Section 2.4. We do not give an overview of these algorithms, but we do give some references.

In the general PDP we deal with a set of origins or destinations and vehicles with different start and end points and transportation requests evolving in real time (dynamic variant). A good overview of techniques to solve these problems can be found in [SS95].

Dial-a-ride problems (DARP) can be divided into two classes: static problems, where the system has a couple of hours to determine a schedule (i.e. for a new day) or dynamic problems

(define (problem capcosts)
(:domain capacity)
(:domain-variables (?volume[package1] 3) - float (?capacity[airplane1] 6) - fluent)
(:objects package1 - PACKAGE airplane1 - AIRPLANE Amsterdam London Paris - LOCATION)
(:init (connect Amsterdam London) (connect Paris London) (at package1 Amsterdam) (at airplane1 Amsterdam))
(:goal (at package1 London)
)
(:action LOAD-AIRPLANE
:parameters (?obj - PACKAGE ?airplane - AIRPLANE ?loc - LOCATION)
:precondition (and (at ?obj ?loc) (at ?airplane ?loc) (test ?volume[?obj] ¡= ?capacity[?airplane))
:effect (and (not (at ?obj ?loc)) (in ?obj ?airplane) (set ?capacity[?airplane] ?capacity[?airplane] -?volume[?obj]))
)
(:action FLY-AIRPLANE
:parameters (?airplane - AIRPLANE ?loc-from ?loc-to - LOCATION)
:precondition (and (at ?airplane ?loc-from) (connect ?loc-from ?loc-to))
:effect (and (not (at ?airplane ?loc-from)) (at ?airplane ?loc-to))
)
etc.

Figure 4.5.: An example of a LPSAT-PDDL problem description.

where the system has to come up with a new schedule within several seconds. Neural networks can be used to estimate travel times: [FT99][SSNB95]

4.3. Multi-agent technology

In this section we discuss a couple of existing systems that support transport (or alike) processes using multi-agent technology. For each project we describe the problem that is being solved, the method the researchers use to solve this problem, and the current state of the method. Again we do not claim to give an exhaustive overview of all multi-agent transport projects, but we hope to give some idea of approaches that are taken towards solving these kinds of problems.

4.3.1. Supply chain formation

The problem Walsh and Wellman [WW99b] try to deal with is supply chain formation. This is an important problem in the area of transporting goods. Several aspects of this problem lead to specific ideas about the design of a multi-agent system to support this task. A supply chain is a network of strongly interrelated exchange relationships among multiple levels of production. Such a supply chain is the result of the involved parties getting a (transportation) task that none of them is able to perform on its own. The following observations can be made.

First of all a hierarchical subtask decomposition is needed. It is necessary to delegate subtasks to other agents, because one agent hasn't got the resources to perform a task on its own. Unfortunately this may also lead to resource contention. That is multiple agents may

want to use the same resource and therefore can not be active at the same time. A second observation is that no entity has all information necessary to centrally form supply chains, therefore we should look at decentralized algorithms and thus distributed decision making. Furthermore, agents do not generally have the incentive to truthfully reveal information, so strategic interactions are important and should be investigated thoroughly. Finally, uncertainty plays a role at several points: failures may occur in agents and in communications, and agents may intentionally fraud. Legal decommitment may be accepted to get to better solutions, but this also introduces more uncertainty.

To deal with these problems, the following approach is taken. Resource contention is represented using a task dependency network with all possible consumers, producers, and suppliers. The goal is to search for an allocation of the tasks to some of these possible agents. This is done using a market-based protocol (they showed that a greedy algorithm simply does not work). The agents negotiate through simultaneous auctions. Experiments suggest this protocol reliably converges to solutions. A combination with contract decommitment to remove dead ends shows high values on average [WW99a]. These ideas are rather fresh and the experiments have not been conducted thoroughly as yet.

4.3.2. Cooperative transportation scheduling

At the German institute for artificial intelligence, people have been working on several projects involving transport and/or multi-agent systems for years. The research has been based for years on a very interesting system is called MARS (Modeling a Multi-Agent Scenario for Shipping Companies). Even the most recent project, called TeleTruck, uses most of the ideas presented back in 1995. In this section we describe both projects shortly.

In several papers the MARS system is described (i.e. [FKM$^+$95][FMP95]). Transportation companies have to carry out transportation orders which arrive dynamically. For this purpose they have a set of trucks at their disposal. Standard Operation Research approaches can hardly cope with the dynamics of this domain. The orders also contain time and/or cost constraints. The complexity of the orders may exceed the capacities of a single company, therefore cooperation between companies is required in order to achieve the goal in a satisfactory way. Also the common use of shared resources, e.g. a train or a ship, requires coordination between the companies. So, although each company has a local, primarily self-interested view, cooperation between them is necessary to achieve reasonable global plans.

The problems in this system are approached in a very distributed way. Often the companies are chosen to be the agents, but in this approach the trucks have plans and negotiate with each other and with the company agent. In the system three types of cooperation exist. Vertical cooperation is the process of task decomposition and task allocation between a shipping company and its trucks. Secondly, horizontal cooperation is the cooperation among a group of autonomous shipping companies, for example by exchanging orders and information about free loading capacity. Finally enhanced cooperation is a procedure in which it is possible for a truck to return an assignment to its company. This task is then entered into a new auction or exchanged with another company.

A truck and its company can use three different task decomposition models for task decomposition and allocation. The most simple form is a centralized model using the contract net protocol (CNP). Using a heuristic the company agent decides which trucking agent should do what task. A more complicated, but also more flexible model is a decentralized task decomposition model using an extended form of the CNP. In this model the company agent

auctions a complete order and then it is possible for trucking agents to do bids for parts of the order. Finally the completely decentralized decomposition model uses simulated trading to allow trucking agents to get rid of very bad assignments. Only if after an auction round and after exchange with the other companies did not reveal a more efficient solution, the original owner sticks to its task. (This protocol is used for enhanced cooperation.) These protocols are described in [FM96] using formal decision theory.

TeleTruck [BFV98][FVB99] continues on the work done related to the MARS system. The goal is to develop an interactive system for computer-supported dispatch in the haulage sector. It takes care of routine tasks and proposes dispatch solutions, thus enhancing the productivity of the dispatchers. The haulage companies and their means of transportation are modeled as autonomous problem solvers (agents) with local knowledge and abilities. In TeleTruck the assignment of customer orders is handled by an auction-like process of negotiation, where agents representing the transportation resources place bids that take into account the vehicle's local cost function. The resulting assignment can be further improved at a later stage by the use of dynamic anytime algorithms.

4.3.3. Negotiation theories

Several theories exist to coordinate actions between agents. One might think of transaction protocols, auctions, and even whole market mechanisms. More information on these subjects can for example be found in [Cle96] and [WWWMM98].

5. Discussion

We described a couple of transportation problems that have many similarities. Based on this knowledge about problems in the real world, we proposed a method to describe such problems and we formulated a more general transportation problem (GTP) mathematically. Then we showed for some problems how they are special instances of the GTP[1]. To get an idea about how difficult this problem is, we took a very simple subproblem and showed this to be NP-hard.

The other contribution of this report is an investigation of possible useful tools and theories. In the previous chapter three topics are studied: planning tools, operation research approaches to transportation problems, and multi-agent approaches to this kind of problems.

In this chapter we give some conclusions (so far they can be made at this point) and give directions for our further research.

5.1. Conclusions

We have seen that the problem is very hard, though several useful attempts have been made to find ways to deal with similar problems. Some tools exist that may help, but they have to be improved and adjusted. We think a combination of planning tools and (approximation or any-time) algorithms for specific problems may be the best approach for us. Wolfman and Weld also combined planning and mathematical programming with their LPSAT [WW99c].

The method proposed in Section 2.2 of describing a problem can be used as input for a translation to both the input of a planner and the mathematical formulation of (for example) an LP-solver.

The classification of transportation problem by defining the GTP can be used to relate our research to others and to determine which algorithms (for which problems) may be useful to us.

5.2. Future research

In the next year we will continu to improve the definition of the GTP and specify as many subproblems as possible. We may also add a survey of available solution techniques to the subproblems.

Also we will improve the description method and precisely define how to use such a description to create a mathematical or a STRIPS formulation.

Another relevant subject is about which planner we should use and in which combination of other techniques.

[1] To formulate a problem that contains all described problems as a special case, still some research has to be done.

But the most interesting question is how can several entities (or agents) solve such a problem together? To answer this question we first have to specify which information should stay private to the agent. We should describe exactly what variant of the problem we have for each agent and then we must find a way to accommodate their private plans to each other. This can be done by bilateral negotiation or maybe a kind of auction protocol or a combination of both.

These ideas should be tested using a simulator. The simulator should continuously simulate the movement of the vehicles and the positions of packages. The packages pop up at their release time and place. The simulator should execute transport actions given by our algorithm. These actions look like "airplane a_j: pick up package p_i at (dep.) time t_k and location v_l". The orders should be entered to our algorithm by the user through some kind of interface. We will use MARS (Multi-Agent Real-time Simulator) [BdW99] to do these simulations, but we have to implement most domain-dependent functionality ourselves using Java and Matlab. We think we should test our ideas in the following order:

1. one central planner
2. multiple planners, fixed structure (chain-manager like), predefined deals
3. multiple planners flexible structure (multiple chain-managers)
4. dynamic set of planners, flexible structure (open environment)

We hope we can generalize our ideas about the multi-agent structure, once we have fully developed them, to other domains than transportation.

We like to end with a couple of research questions:

- Are there any algorithms that can create and update a schedule for real-life networks optimally and fast enough? (Answer: since the problem is NP-hard, this is very unlikely)
- Are there any such algorithms that can approximate a solution? Can they be proven to lie within certain bounds? (Answer: until now, we only found some static approximation algorithms for variants of this problem [BS97].)
- Is there a way to run such an algorithm distributedly? I.e. with each node having access to only a part of the information and with some communication between the nodes? (Note the difference between distributed and parallel.)
- Do we need / have use for a common structure for such a distributed computation? If so, what are useful properties of such a structure?

A. Terminology

distributed the processing is done by multiple entities, often physically divided and each with its own resources; in general, the division of work is based more on available resources (information) than on equal shares; distributed processing is often done because of the locations of the resources

dynamic additional constraints can be added during the solution and execution phase[1]

incremental the solution is improved by small steps; intermediate solutions are correct, but less efficient or more costly

off-line analysis (by designers)

on-line same as dynamic

parallel the processing is done by multiple processors, often located together and sometimes using a shared memory; parallel processing is often done because of speed-ups

planning formulating a program for a definite course of action

real-time describes an application which requires a program to respond within some small upper limit of response time

scheduling setting an order or time for planned events

static the whole problem is known in advance and calculated only once

[1] maybe we should distinguish between these two also

Bibliography

[BdW99] A. Bos and M.M. de Weerdt. Description of the MARS simulator. Technical report, Delft University of Technology, 1999.

[BFV98] H.-J. Bürckert, K. Fischer, and G. Vierke. Transportation scheduling with holonic mas – the teletruck approach. In *Proc. of the Third International Conference on Practical Applications of Intelligent Agents and Multiagents (PAAM'98)*, 1998.

[BS97] A. Baveja and A. Srinivasan. Approximation algorithms for disjoint paths and related routing and packing problems. In *Proc. IEEE Symposium on Foundations of Computer Science*, pages 416–425, 1997.

[Cle96] S.H. Clearwater. *Market-Base Control – a paradigm for distributed resource allocation*. World Scientific Publishing Co., 1996.

[DDSS95] J. Desrosiers, Y. Dumas, M.M. Solomon, and F. Soumis. Time constrained routing and scheduling. In M.O. Ball, T.L. Magnanti, C.L. Monma, and G.L. Nemhauser, editors, *Network Routing*, volume 8 of *Handbooks in Operations Research and Management Science*, pages 35–139. Elsevier Science Publishers B.V., 1995.

[DG92] T. Dean and L. Greenwald. A formal description of the transportation problem. Technical Report CS-92-14, Department of Computer Science, Brown University, March 1992. Available at ftp://ftp.cs.brown.edu/pub/techreports/92/cs92-14.ps.Z.

[dW99] M.M. de Weerdt. Planning tools. Technical report, Delft University of Technology, to appear, 1999.

[FKM+95] K. Fischer, N. Kuhn, H.J. Müller, J.P. Müller, and M. Pischel. Sophisticated and distributed: The transportation domain. In C. Castelfranchi and J.-P. Müller, editors, *From Reaction to Cognition*, volume 957 of *Lecture Notes on Artificial Intelligence*, pages 122–138. Springer Verlag, 1995.

[FM96] K. Fischer and J.P. Müller. A decision-theoretic model for cooperative transportation scheduling. In W. van de Velde and J.W. Perram, editors, *Agents Breaking Away — Proc. of the 7th European Workshop on Modelling Autonomous Agents in a Multi-Agent World (MAAMAW'96)*, volume 1038 of *Lecture Notes on Artificial Intelligence*, pages 177–189. Springer Verlag, 1996.

[FMP95] K. Fischer, J.P. Müller, and M. Pischel. A model for cooperative transportation scheduling. In *Proc. of the 1st International Conference on Multiagent Systems (ICMAS'95)*, pages 109–116, San Francisco, June 1995.

[Fre99] FreeCargo. Road haulage freight exchange for european transport companies. `http://www.freecargo.co.uk/`, 1999.

[FT99] Liping Fu and S. Teply. On-line and off-line routing and scheduling of dial-a-ride paratransit vehicles. In *Computer-Aided Civil and Infrastructure Engineering*, volume 14, pages 309–319. Blackwell Publishers, 1999.

[FVB99] P. Funk, G. Vierke, and H.-J. Bürckert. A multi-agent systems perspective on intermodal transport chains. In E. Erkens, editor, *Proc. of the Conference on Logistik Management (LM'99)*, Bremen, 1999. Springer Verlag.

[GJ79] M.R. Garey and D.S. Johnson. *Computers and intractability – a guide to the theory of NP-completeness*. W.H. Freeman and company, New York, 1979.

[Kar75] R.M. Karp. On the complexity of combinatorial problems. *Networks*, 5:45–68, 1975.

[Kje98] D. Kjenstad. *Coordinated supply chain scheduling*. PhD thesis, Norwegian University of Science and Technology, 1998.

[KNHD97a] J. Koehler, B. Nebel, J. Hoffman, and Y. Dimopoulos. Extending planning graphs to an ADL subset. In *Proc. of the 4th European Conference on Planning*, volume 1348 of *Lecture Notes on Artificial Intelligence*, pages 273–285, Toulouse, 1997. Springer Verlag. Available at `http://www.informatik.uni-freiburg.de/~koehler/ipp/publi.html`.

[KNHD97b] J. Koehler, B. Nebel, J. Hoffman, and Y. Dimopoulos. Extending planning graphs to an ADL subset. Technical Report 88/97, University of Freiburg, 1997. Available at `http://www.informatik.uni-freiburg.de/~koehler/ipp/publi.html`.

[KS98] H. Kautz and B Selman. BLACKBOX: A new approach to the application of theorem proving to problem solving. In *Working notes of the workshop on planning as combinatorial search, held in conjunction with AIPS'98*, Pittsburgh, PA, 1998.

[Löb99] A. Löbel. Solving large-scale multiple-depot vehicle scheduling problems. In N.H.M. Wilson, editor, *Computer-Aided Transit Scheduling*, Lecture Notes in Economics and Mathematical Systems, pages 193–220. Springer Verlag, 1999. Available at ftp://ftp.zib.de/pub/zib-publications/reports/SC-97-17.ps.Z.

[McD98] D. McDermott. PDDL – the planning domain definition language. Technical Report TR-98-003, Yale Center for Computational Vision and Control, October 1998.

[Nie98] I. Niemelä. Logic programs with stable model semantics as a constraint programming paradigm. In *Proc. of the Workshop on Computational Aspects of Nonmonotonic Reasoning*, Trento, Italy, June 1998. Available at `http://www.tcs.hut.fi/Publications/reports/A52/niemela.ps.gz`.

[NS97] I. Niemel and P. Simons. Smodels - an implementation of the stable model and well-founded semantics for normal logic programs. In *Proc. of the 4th International Conference on Logic Programming and Nonmonotonic Reasoning*, volume 1265 of *Lecture Notes in Artificial Intelligence*, pages 420–429, Dagstuhl, Germany, July 1997.

[PBC+96] B. Pell, D.E. Bernard, S.A. Chien, E. Gat, N. Muscettola, P.P. Nayak, M.D. Wagner, and B.C. Williams. A remote agent prototype for spacecraft autonomy. In *Proc. of the SPIE Conference on Optical Science, Engineering, and Instrumentation*, 1996. Available at `http://ack.arc.nasa.gov/ic/projects/mba/`.

[SS95] M.W.P. Savelsbergh and M. Sol. The general pickup and delivery problem. *Transportation Science*, 29(1):17–29, 1995.

[SSNB95] J.A. Stankovic, M. Spuri, M. Di Natale, and G.C. Buttazzo. Implications of classical scheduling results for real-time systems. *IEEE Computer*, 28(6):16–25, June 1995.

[SSRB98] J.A. Stankovic, M. Spuri, K. Ramamritham, and G.C. Buttazzo. *Deadline Scheduling for Real-Time Systems – EDF and Related Algorithms*. Kluwer Academic Publishers, 1998.

[Su99] C.-T. Su. Dynamic vehicle control and scheduling of a multi-depot physical distribution system. In *Integrated Manufacturing Systems*, volume 10, pages 56–65. MCB university press, 1999.

[Tim] TimoCom. Timocom truck and cargo, european-wide load space and freight database. `http://www.timocom.de/e_home1.html`.

[Tra] TranspoWeb. Transpoweb freight exchange. `http://transpoweb.hypermart.net/index-uk.html`.

[vGRS91] A. van Gelder, K.A. Ross, and J.S. Schlipf. The well-founded semantics for general logic programs. *Journal of the ACM*, 38(3):620–650, July 1991.

[Wel99] D.S. Weld. Recent advances in AI planning. *AI Magazine*, 20(2):93–123, Summer 1999.

[WJ98] M.J. Wooldridge and N.R. Jennings. Pitfalls of agent-oriented development. In *Proc. of the 2nd International Conference on Autonomous Agents (Agents'98)*, pages 385–391, Minneapolis, USA, 1998.

[WM98] D.E. Wilkins and K.L. Myers. A multiagent planning architecture. In *Proc. of the 4th International Conference on Artificial Intelligence Planning Systems (AIPS'98)*, 1998.

[WW99a] W.E. Walsh and M.P. Wellman. Efficiency and equilibrium in task allocation economies with hierarchical dependencies. In *Proc. of the 16th International Joint Conference on Artificial Intelligence (IJCAI'99)*, 1999.

[WW99b] W.E. Walsh and M.P. Wellman. Modeling supply chain formation in multi-agent systems. In *International Joint Conference on Artificial Intelligence-workshop on Agent Mediated Electronic Commerce*, 1999.

[WW99c] S.A. Wolfman and D.S. Weld. The LPSAT engine and its applications to resource planning. In *International Joint Conference on Artificial Intelligence(IJCAI'99)*, 1999.

[WWWMM98] M.P. Wellman, W.E. Walsh, P.R. Wurman, and J.K. MacKie-Mason. Auction protocols for decentralized scheduling. In *Proc. of the Eighteenth International Conference on Distributed Computing Systems*, Amsterdam, May 1998.

[ZLvdH98] K. Zhu, M.W. Ludema, and R.E.C.M. van der Heijden. A multi-agent based implementation of the pipeline concept for air cargo logistics. Technical report, Dept. of Transportation and Logistics' Organization, Faculty of Systems Engineering, Policy Analysis and Management, Delft University of Technology, 1998.