General methods for synchromodal planning of freight containers and transports

D. Huizing

Graduation thesis
MSc Applied Mathematics, TU Delft

General methods for synchromodal planning of freight containers and transports

by

D. Huizing

to obtain the degree of Master of Science
at the Delft University of Technology,
to be defended publicly on Wednesday August 23, 2017 at 10:00 AM.

Student number:	4176138	
Project duration:	January 9, 2017 – July 26, 2017	
Thesis committee:	Prof. dr. ir. K. I. Aardal,	TU Delft
	Dr. D. C. Gijswijt,	TU Delft, supervisor
	Prof. dr. J. L. van den Berg	University of Twente
	Dr. F. Phillipson,	TNO
	Ir. A. Sangers,	TNO

TU Delft

Abstract

Synchromodal freight transport is introduced as intermodal transport, so container transport that uses several transportation vehicles, with an increased focus on a-modal booking, cooperation and real-time flexibility.

It is confirmed that general synchromodal network planning methods are rare or non-existent at the operational level.

An extensive framework is developed that describes characteristics of different mathematical synchromodal optimisation problems on the tactical-operational levels.

Three different problems are defined using this framework. Solution methods for these three problems are developed in this thesis, with a focus on low computation times so as to facilitate decision-support and real-time flexibility, and a focus on generality so as to make the methods applicable throughout different organisation structures.

In the first problem, it is assumed that the transportation vehicles have fixed time tables and one only has to decide on a container-to-mode assignment, so by what modality-paths all containers reach their destination against minimal total cost. The containers have release times and deadlines. A model that also allows soft due dates is developed. Moreover, the option of using trucks or other 'infinite resources' to help fulfil requests is added. With appropriate graph reductions, this problem can be solved to optimality in little time by solving the minimum cost multi-commodity flow problem on an appropriate space-time network.

In the second problem, the goal is the same but almost any element can be stochastic: for instance, travel times and container release times could be given a discrete probability distribution rather than a fixed value. Rigorous definitions are formulated to capture the generalities in this stochasticity. Multistage stochastic optimisation and Markov Decision Processes are illustrated, but advised against for their computing time: instead, Expected Future Iteration and 70%-Pessimistic Future Iteration are developed and shown to yield near-optimal results in a small amount of time in the simulated environment.

In the final problem, there are no stochastic elements, but the decision-maker is given control over the vehicle time tables in addition to the control over container-to-mode assignments. This problem is argued to be a departure from classical optimisation problems, but shown to still be strongly NP-hard. An integer linear program is developed to solve the problem, but the results show that it scales too poorly to solve problems of 'real life size' in an appropriate amount of time for decision support. The Greedy Gain heuristic and Compatibility Clustering heuristic are developed: they solve much more limited sub-problems with the ILP, but unfortunately, even these sub-problems require too much computational effort at the wished instance size.

A number of topics for future research are formulated, giving concrete advice on how to solve the second problem more robustly and how to solve the third problem more quickly.

Extended management summary

Chapter 1 (Introduction). Intermodal freight transport means transporting freight containers using a series of different transportation modalities: for example, using a barge for the first half of the journey and a truck for the second. Synchromodal freight planning is a form of intermodal planning, with increased focus on at least one of the following three aspects:

1. *A-modal booking*: Customers do not tell the logistics service provider (LSP) what modalities they want their goods transported by, but leave it up to the LSP, just as long as the goods arrive at the agreed time against an agreed price;
2. *Cooperation*: Different transporters share their information and resources to form a synchromodal network, so they or an LSP can find efficient ways to fulfil all transportation requests;
3. *Real-time flexibility*: Planning is done using real-time data: plans are adjusted when disturbances occur, but also, robust plans are made keeping potential future disturbances into account. It must be possible to re-evaluate plans at any moment.

This thesis seeks to answer the following research question:

How can on-line container-to-transport assignment and operational transport scheduling in synchromodal freight transport be optimised against their corresponding definition of 'optimal'?

To this end, this thesis studies:

1. Developing a framework for synchromodal problems and literature;
2. *Problem 1*: how to synchromodally assign containers to transportation modalities if the transports have fixed timetables and there are no unknowns;
3. *Problem 2*: problem 1, but with stochastic elements;
4. *Problem 3*: how to synchromodally determine both the transport timetables and container-to-mode assignments if there are no unknowns.

Solutions should be produced fast enough for operational decision-support.

Chapter 2 (Literature study). In operations research, people often speak of problems on either the strategic, tactical or operational level. This thesis mainly studies operational problems. Synchromodal freight transport is a young concept, and several literature studies point out that operational synchromodal network-wide planning methods are rare or non-existent in literature. Work relevant to this thesis was performed by Pedersen, Kooiman, Rivera, van Riessen, Behdani and Ozdaglar.

Chapter 3 (Framework). *(Co-authored.)* A tactical/operational synchromodal planning problem can largely be characterised by whether the following elements are fixed, controlled, stochastic, dynamic or not relevant:

RO: resource origin	DO: demand origin
RD: resource destination	DD: demand destination
RC: resource capacity	DV: demand volume
RDT: resource departure time	DRD: demand release date
RTT: resource travel time	DDD: demand due date
RP: resource price	DP: demand penalty
TH: terminal handling time	$D2R$: demand-to-resource assignment

Furthermore, information can either be available globally or locally, and optimisation can be either global or local: global optimisation concerns achieving low total network costs, whereas local optimisation concerns achieving low individual costs for different parties. Therefore, the system behaviour can be either *social*, *selfish*, *cooperative* or *limited*.

For example: in the Kooiman pick-up problem [35], barges make fixed round trips, starting at a depot. Containers that must be transported to this depot appear at terminals at random times. At each terminal, the barges must decide how many containers to load here, given that they may have to reserve space for the next terminals, but not knowing exactly how many containers will be released at those terminals by the time they arrive.

In the proposed framework notation, the Kooiman pick-up problem can be denoted as a $\bar{R}, \cancel{TH}|\bar{D}, [D2R], \widehat{DRD}|social(1)$-problem, because: the resources (barges) have mainly fixed features ($\bar{R}$); the terminal handling times are irrelevant ($\cancel{TH}$); the demand items (containers) have mainly fixed features ($\bar{D}$); the demand-to-resource assignment is controlled ($[D2R]$); the demand release dates are stochastic ($\widehat{DRD}$). There is only one decision-making entity, the information is global and the optimisation is global as well ($social(1)$).
More examples of classification of papers are given in Table 3.3. Examples of classification of real life use cases and problems are given in Table 3.4.

If an element varies depending on state or time but in a known way, this element is dubbed dynamic. For example, if resources are pricier during the weekends, this is denoted as $\widetilde{RP}$. If there is more than one decision-making entity in the system, the interaction can be modelled as either $one\ turn\ only$, so everyone gets one turn to make decisions, $equilibrium$, so turns are circulated until no one updates their plans or $isolated$, so decision-makers do not know what decisions the others make. This can be added as a fourth field in the framework notation.

The benefits of this framework lie in structuring existing literature, pointing modellers towards literature useful to their problem and discovering similarities between problems. The challenges lie mainly in the partial subjectivity of classification.

Chapter 4 (Problem 1). In this problem, vehicle schedules are assumed to be already fixed and one only has to decide how to assign containers to modality-paths, to get each container where it needs to be against minimum total cost. Suppose 18 containers are released at location A, time 09:00, and are due at location B, time 14:00; suppose also that one barge and one train depart in the meantime, the former from A to B at 09:30 with capacity 20 and unit price 523, arriving at 13:30; the latter from A to B at 11:00 with capacity 10 and price 498, arriving at 12:30. This can be translated into the space-time network seen in Figure 1. In this specific instance, it is optimal to send 10 containers by train, using all its capacity, and sending the other 8 containers by barge.

In general, solving Problem 1 to optimality is a matter of modelling it as a minimum cost

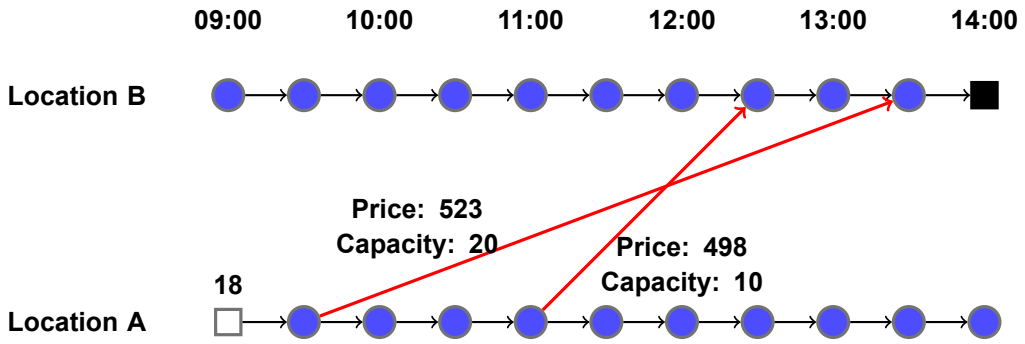


Figure 1: The network described in Figure 4.1, translated to a space-time network without trucks. The black arcs represent 'waiting arcs' with infinite capacity and cost zero. The 18 containers are situated at the white square node in space-time and need to go to the black square node in space-time.

multi-commodity flow problem on a space-time network, which can be solved with an ILP solver. The model can be expanded to also allow using trucks or other 'infinite resources' from any place to any other place at any time, against a high price. The model can also be expanded to allow simultaneous soft due dates and hard deadlines.

When applying Algorithm 1 to remove redundant information from the model, instances of 'real life size', with 40 requests over 32 locations and 121 time steps, can be solved to optimality in an average of only 9 seconds on modest hardware. This is related to how in 100% of the generated instances, it suffices to use an LP-solver rather than an ILP-solver, though instances exist where this is not the case.

Chapter 5 (Problem 2). In this problem, the challenge is again only to assign containers to modality paths; however, many elements can be random. For example, in Figure 2, 1 container has to be transported with soft due date 2 and hard deadline 4: it could be sent by a barge, which has a probability of 1/3 to arrive at time 2, so a lateness penalty of 0 has to be paid, a probability of 1/3 to arrive at time 3, so a lateness penalty of 75 has to be paid, and a probability of 1/3 to arrive at time 4, so a lateness penalty of 500 has to be paid. It could also be sent by a truck, which is certain to arrive on time, but costs 200 to call on instead of 100. Sending it by barge has an *expected* cost of 291.67, while the truck has a certain cost of 200, so it is a 'better' choice to send it by truck, even though barge has a 66.7% probability of being cheaper.

To generalise this idea, much attention is spent in this chapter on how to interpret the objects in the problem and what planning assumptions are made. Multistage stochastic programming and Markov Decision Processes are illustrated as a way to solve this problem, but

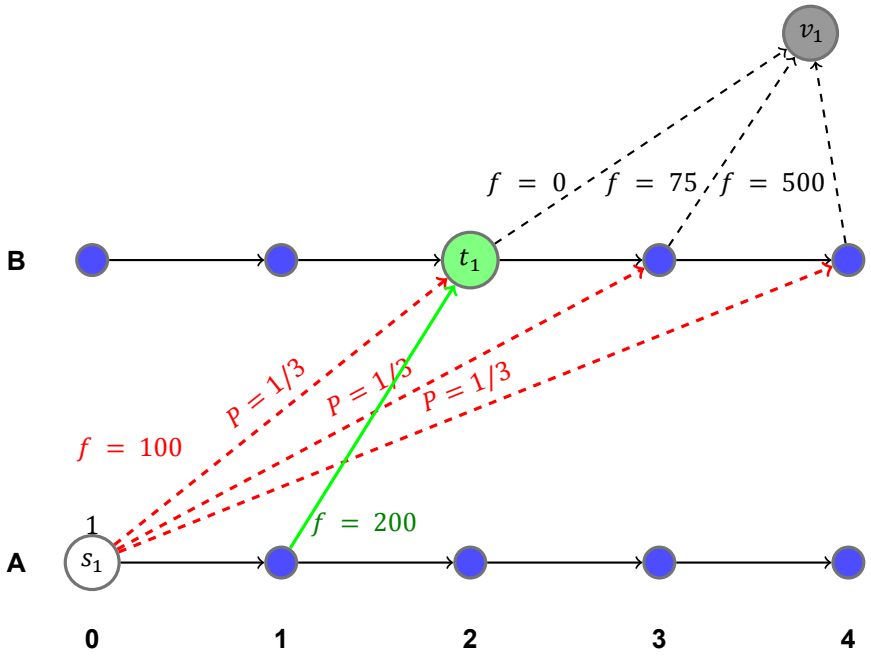


Figure 2: One container has to be assigned to a transportation modality. The red, dashed option has probability $1/3$ of taking two time steps, thus arriving just on time, probability $1/3$ of taking three time steps, causing a lateness penalty of 75 and probability $1/3$ of taking four time steps, causing a lateness penalty of 500. The green option is guaranteed to be on time, but has a base usage cost of 200 instead of 100. Both transports have capacity 10. At time $t = 0$, a planner is faced with the decision: red or green?

they are argued to require far too much computation time.

Instead, two other decision processes are developed. The first is Expected Future Iteration: assume that every unknown will take its expected value, so that a non-random instance can be observed, which is an instance of Problem 1 and can be solved as such. This leads to some decisions that have to be enacted in the current time step: enact those, then go to the next time step and recompute the expected future, knowing all that you know now, and repeat this process ad infinitum.

Expected Future Iteration can give arbitrarily bad results, because it blindly assumes that a certain future will take place and 'does not optimise its plan B'. To partly counteract this, 70%-Pessimistic Future Iteration is developed; though it has the same property, the probability of negative consequences is made smaller at the cost of higher expected costs. It resembles robust optimisation, except no guarantees are made on the probability that the given plan will work out. The method works the same as Expected Future Iteration, except 70%-percentiles are taken instead of expected values; any percentile, of course, could be taken to adjust the amount of 'risk' a decision-maker is willing to make for the possibility of profit.

Both methods produce near optimal average costs for instances of 'real life size' in an average half minute on modest hardware, though these results may differ when used in actual operational practice, rather than the used simulations. The simulation method used by Kooiman may yield better results, because this 'does optimise its plan B', but a faster solution method for Problem 1 would be necessary for a fast implementation of this simulation method.

Chapter 6 (Problem 3). In this final problem, there are no more random elements, but the decision-maker must simultaneously determine vehicle time tables and container-to-mode assignment; in other words, the decision-maker must tell barges and other vehicles where to go when and what containers to take with them. Again, trucks and other infinite resources

can be used to deliver those containers that cannot be delivered by the other vehicles. A typical instance of this problem, together with one of its feasible solutions, are presented in Figure 3.

This problem has a large decision space, and a large number of variables and parame-

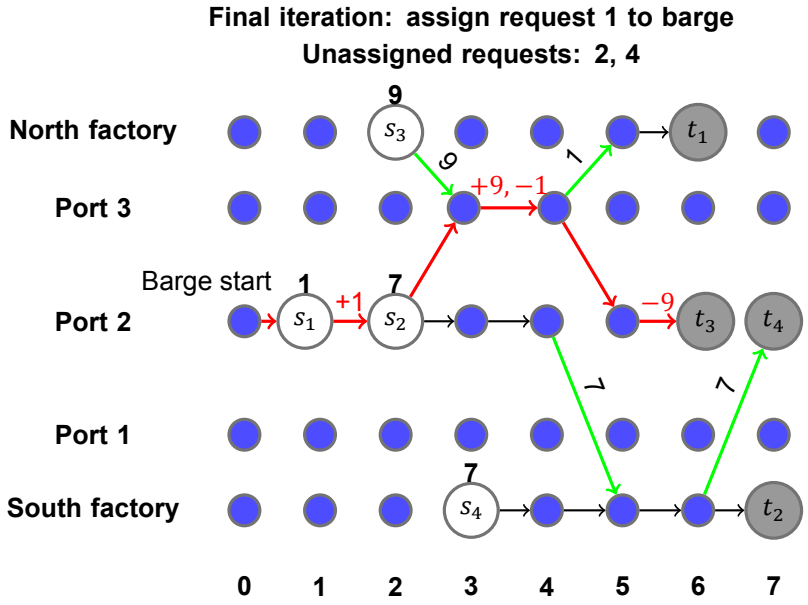


Figure 3: A barge with capacity 20 is sent to handle two of the four requests. Its route is given in red. The first request is to deliver 1 container, released at Port 2, time 1, to North Factory with deadline 6. The other three can be read from the figure similarly. The barge can only visit ports, not factories: request 3 spends the first part of its journey on a truck (green) to Port 3. During time step 1, the one container of request 1 is loaded; during time step 3, the nine containers of request 3 are loaded and the 1 one container of request is unloaded; during time step 5, the nine containers are unloaded. Though black waiting arcs are present everywhere, they are only left in where they might help track the containers' path through space-time. Note that if all costs are 'vertically symmetrical', a better solution would be to take a route over Port 1 instead of Port 3, to handle the 14 containers of request 2 and 4 by barge so that only the 10 containers of request 1 and 3 have to be trucked.

ters must be introduced to discuss the problem properly. The problem can be shown to be NP hard: in other words, that no polynomial time solution method exists, nor that a Fully Polynomial Time Approximation Scheme exists, unless $P = NP$.

An integer linear program is again proposed to solve instances of the problem to optimality. However, only in small instances can this produce results fast enough for operational decision support. Instead, the Greedy Gain heuristic is proposed for larger instances: it assigns requests to vehicles one at a time, based on the highest immediate gain, which can be found by solving an appropriate sub-problem with the ILP. Greedy heuristics typically yield non-optimal results, because they may make unfortunate decisions that seem smart in the moment. To avoid this effect, the Compatibility Clustering heuristic is developed, which first tries to cluster requests on how 'compatible' they are, then solves sub-problems to see how much it would cost to assign some vehicle to some cluster, then takes the minimum cost matching between vehicles and clusters.

Unfortunately, these limited sub-problems add up to something computationally too heavy: none of the three methods, as they stand now, can produce results fast enough for operational decision support in instances of 'real life size'. Furthermore, the Greedy Gain heuristic finds quite non-optimal solutions, and the Compatibility Clustering heuristic achieves worse costs: the actual way to go about clustering may have to be looked at closer in future research.

Future research should aim at speeding up these methods so that results can be found fast enough for operational decision support, or at finding other methods to do so, where this

thesis advises against solving sub-problems with the current integer linear program because of its computational intensity. Several concrete routes of improvement are proposed.

Chapter 7 (Conclusion) A comprehensive framework was developed and its merits and challenges discussed. Problem 1 can be solved to optimality in little time, using an integer linear program. Problem 2 can be solved to near-optimality in little time, using either Expected Future Iteration or 70%-Pessimistic Future Iteration, though this may require verification against real use case data. Problem 3 can be solved, either to optimality or with heuristics, but currently, not in little time for problems of 'real life size'. A number of topics for future research are proposed, mainly in the form of concrete advice for how to speed up these methods and how to achieve better costs with heuristics.

Preface

First of all, my thanks to you: it is a joy and an honour to me that you are reading this, in whatever context that may be. Though I am known to be a man of many words, I will restrict this preface to many words of thanks.

Thanks to my TNO supervisors, Alex and Frank. Thanks to Alex for his encouraging feedback, and Frank for his insightful feedback; thanks to both of them for at least attempting to convince me that my planning was again too ambitious, and for accepting my youthful shortcomings with a healthy dose of humour. I would hope my time as your colleague at TNO could still continue, but if not, thanks for all that you've taught me.

Thanks to my TU supervisor, Dion. Thanks for the patience and positivity when I was convinced I had very nearly proven a polytope integrality property that you clearly saw to not be true. Thanks also for the patience when you could not directly follow my strange models, and the patience when I could not directly follow your advanced graph theoretical arguments. I will look back at how our meetings typically lasted twice as long as appointed as a happy memory.

Thanks also to the other members of my committee, Karen and Hans, for making time to help make my graduation possible. My apologies to Karen for the many forms I asked her to sign.

Thanks to my direct colleagues, Max, Myrte and Iris, for sharing this journey, for being great sparring partners and for putting up with me trying to meddle with your projects all the time.

Thanks to my fellow students. In particular, thanks to Pim: thanks for being a positive role model, and for pointing me towards a field of mathematics where I would never have to look at partial differential equations again.

Most of all, thanks to my friends and family. Thanks to my aunt and uncle, who have helped make this incredible journey possible. Thanks to my parents, brother and sister for their respect and support. Thanks to my best friend for remaining fascinated, even when I would ramble about interior point methods. Thanks to my other best friend, for not being interested in those topics at all, and for thus giving colour to my life. Thanks to my family for always believing I could make it, and thanks to my friends for understanding when I had to skip out for homework, and for being my reason I didn't need a student life. Though I've done many things in this thesis, and have developed a sense of awe for what other mathematicians have done before me, my greatest highlight in this journey will be to tell you what I've done, and if all goes well, to be allowed to graduate in your presence.

D. Huizing
Zoetermeer, August 2017

Contents

1 Introduction

Synchromodal freight transport is a new concept within the logistics sector. It is a continuation of *intermodal freight transport*, that is to say, sending freight in standardised containers that may for example spend the first part of their journey on truck, the second part on barge, the final part on train and the final part on truck again. Synchromodality improves on this by introducing three new ways of working [31]:

1. *A-modal booking*: Customers do not tell the logistics service provider (LSP) what modalities they want their goods transported by, but leave it up to the LSP, just as long as the goods arrive at the agreed time against an agreed price;

2. *Cooperation*: Different transporters share their information and resources to form a synchromodal network, so they or an LSP can find efficient ways to fulfil all transportation requests;

3. *Real-time flexibility*: Planning is done using real-time data: plans are adjusted when disturbances occur, but also, robust plans are made keeping potential future disturbances into account. It must be possible to re-evaluate plans at any moment.

The goal of aspect 1 and 2 is to increase overall efficiency and sustainability, by facilitating *consolidation*, in other words, letting one small request wait at a terminal so it can be combined with some other request. See also Figure 1.1 and the work by Vinke [59]. Aspect 2 also facilitates smarter *equipment repositioning*, for example, by moving leftover empty containers directly to a nearby terminal where they are needed instead of through a depot [2]. In a mathematical sense, having someone coordinate a large network allows for a larger decision space, in which smarter solutions can be found, but for which smarter solution methods may also be necessary.

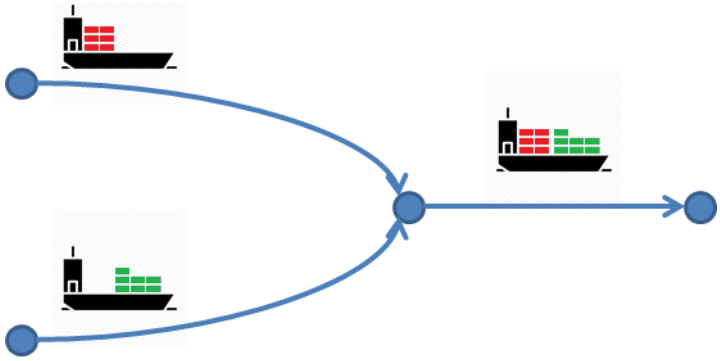


Figure 1.1: Consolidation: one of the benefits of global information and a central operator (aspects 1 and 2 of synchromodality).

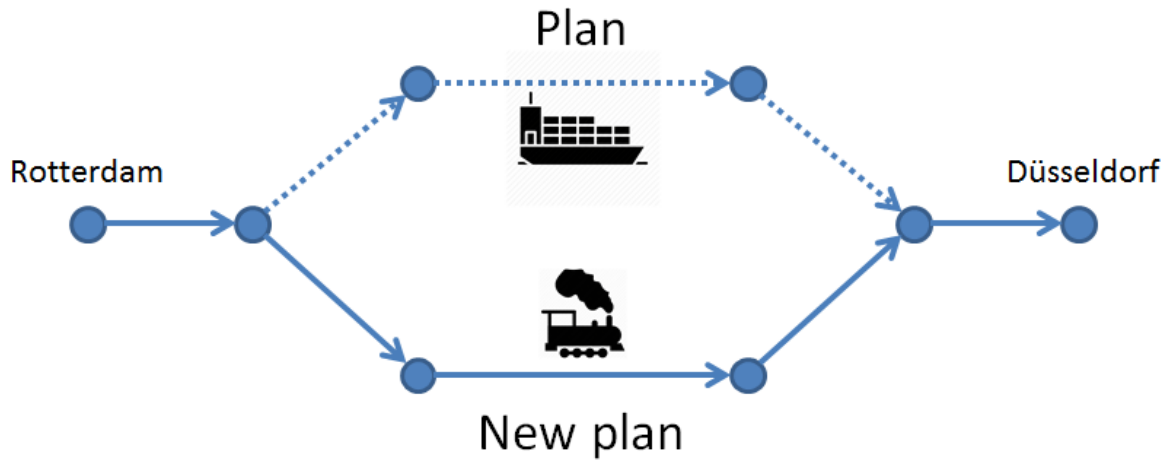


Figure 1.2: On-line changes in transport planning (aspect 3 of synchromodality).

The goal of aspect 3 is to increase flexibility and reliability. For example: if the weather is going to be unfavourable, it may be a good idea to send containers by rail instead of barge. See also Figure 1.2.

Interest in synchromodality has increased, due to improvements in data technology, an increased focus on the more complicated hinterland transport and the ever-growing need for efficiency [3]. However, synchromodality faces several challenges that keep it from being adopted in practice [31]. For one, the information infrastructure and standardisation is absent. Secondly, many people are not even aware of the concept of synchromodality. Thirdly, companies are often reluctant to supply information, because this could harm their competitive position. Fourthly, in the case of having a central operator making decisions for different parties, there is not yet a business model that 'promises' that indeed all parties are better off than when they make their own decisions. The scepticism caused by these last three issues have, as of late, been addressed by serious gaming projects like SynchroMania [10] and SYNCHRO-GAMING [9].

A final challenge is of a mathematical nature: if indeed information and control are shared to some degree, and real-time switching is allowed and expected, *how* does a net operator make an optimal planning? If this question can be answered, it should greatly contribute to the advancement of and towards synchromodal transport.

1.1. Research question

Several theses will be written within a larger NWO program [5] on different aspects of the large, multidisciplinary problem of implementing a synchromodal transport chain. The goal of this specific thesis is to provide freight transport decision-support by optimisation in two fields: *container-to-transport assignment* and *operational transport scheduling*. By the former is meant, for example, deciding whether a container should be transported by barge A, train B, barge C, or barge A but on its voyage of next week. By the latter, the same is meant but with added control on the actual time tables of the resources. These stochastic versions of these problems are viewed as Time-stamp Stochastic Assignment Problems, and as such, this thesis will build upon the work of Kooiman et al. [35]. General optimisation methods will be developed and compared for different net-centric scenarios and the scenarios will be put in a framework. All in all, the following research question is posed:

How can on-line container-to-transport assignment and operational transport scheduling in synchromodal freight transport be optimised against their corresponding definition of 'optimal'?

This is answered along the following sub-questions:

1. How can different scenarios of synchromodal freight transport be classified within an exhaustive framework?
2. How can a low cost net-centric container-to-transport assignment be found fast enough for on-line use if everything is known beforehand?
3. How can a low cost net-centric container-to-transport assignment be found fast enough for on-line use if new data is still expected to come in?
4. How can a low cost net-centric operational transport schedule be found fast enough for on-line use if everything is known beforehand?

1.2. Report structure

Chapter 1 introduced the thesis problem. Chapter 2 will discuss the existing nomenclature and literature that is applied in the thesis. Chapter 3 until Chapter 6 will discuss sub-questions 1 until 4 respectively. In Chapter 7, the results of this thesis will be concluded and future work for members of the NWO project will be proposed.

2

Literature study

In this chapter, the conventional terminology in synchromodal freight transport discussions is summarised, including where the research problem fits within the bigger picture of operations research. When this is determined, relevant literature is summarised.

2.1. Conventional terminology

Several related terms exist in transportation approaches [53]. *Unimodal transport* usually refers, bluntly put, to simply loading cargo onto a truck and driving it from origin to destination. *Multimodal transport* refers to transporting cargo from origin to destination by more than one transportation mode, both in regards to transportation vessel as containment unit. In *intermodal transport*, more than one transportation mode may be used, but the containment unit must always be a container of standardised size. For example: when transporting oil barrels from a Dutch refinery through the Port of Amsterdam to a client in China, this is considered multimodal for first using a truck in its *pre-haul* to the port, then a cargo ship for its *long haul* to China and then a truck for its *post-haul* to the client. However, it is not considered intermodal, as barrels are being used instead of intermodal containers. *Co-modal transport* resembles multimodal transport, but requires a consortium of shippers and has a focus on exploiting the benefits of each transportation mode in a smart way. Finally, synchromodal transport is a version of intermodal transport that focuses on real-time planning flexibility and coordination between different shippers, both using large amounts of real-time data.

Any project that uses the collection and sharing of real-time data to make smarter freight transport decisions can, as such, be seen as a synchromodal transport project. This results in the existence of similar research fields with different names: some of the problems researched in the Netherlands and Austria under the name of synchromodality, are researched in France under the name of 'physical internet'. Some experts advocate not getting stuck in discussions on the semantics of whether a research project or development project is in the field of multimodality, intermodality, synchromodality or physical internet, as long as progress is being made [31]. However, for the purpose of modelling a problem and finding related problems, it is useful to employ consistent terminology that communicates model assumptions and optimisation goals, while remaining aware that valuable research may have been done under different names. Reis discusses the current ambiguity of terminology and the potential usefulness of consistent terminology [49]. This report attempts to use and define more consistent terminology, as further detailed in Chapter 3.

Freight transport decision problems are often categorised on three levels [15, 53, 58]: problems on the strategic, tactical and operational level. *Strategic* problems concern long-term investments in the transportation network, for example, where to build new terminals. *Tactical* problems may concern service design, for example, determining how many times in a

month a barge should make a round-trip. *Operational* problems concern using a current network in an optimal way for problems occurring in the present. The problems covered in this report are considered operational, because of their on-line nature of decision making.

More specifically, Behdani describes [15] not three, but six levels of decision problems in synchromodal freight transport, paraphrased here from most strategical to most operational:

- *Synchromodal Network Design*: determining optimal infrastructural nodes and connections, for example, determining where to build terminals and rail lines;
- *Synchromodal Service Pricing Strategies*: determining how much the client should pay for a-modal service, as opposed to service where the client determines the chosen modes;
- *Intermodal Gain Sharing and Contract Design*: determining the rules between parties in the transport chain, including how the costs and profits are divided;
- *Synchromodal Service Design*: determining delivery priority policies and expected service levels and, using this, designing products of varying price, delivery time and reliability;
- *Operational Resource Scheduling*: determining transportation schedules and container-to-mode assignments on a day-to-day basis;
- *Exception Handling and Real-Time Switching*: updating these schedules and assignments, if beneficial or necessary, using real-time information.

This report focuses on problems on the last two levels, keeping potential real-time switching in the future in mind. More specifically, the following optimisation problems are investigated, in different contexts of synchromodal transportation:

- Determining operational transportation schedules,
- determining operational container-to-mode assignments,
- real-time switching in the above two;

in each case respecting potential real-time switching in the uncertain future.

2.2. Relevant literature

In her recent literature review, SteadieSeifi remarks [53]: “[Co-modality and synchromodality] have not received any attention from the OR community.” In his even more recent overview, Van Riessen adds [58]: “For efficient synchromodal transport plans it is essential to allow real-time switching, i.e. real-time planning updates. This was recognized by all studies that referred to synchromodal transportation, but not many real-time planning methods that provide a network-wide plan exist yet.” However, a number of methods for intermodal planning exist [22]: if a general way of adding real-time switching to these methods can be found, a large amount of synchromodal methods can be created.

In the field of intermodal transportation, Pedersen formulated a Capacitated Multi-Commodity Network Design model, which can be reinterpreted for simultaneous operational scheduling and container-to-mode assignment. He formulates it as a MILP and proposes a Tabu Search method [44]. Bektas comments on this solution method: “Pedersen, Crainic, and Madsen (2006) and Andersen et. al. (2006, 2007) present formulations and propose solution methods, but significant research work is still needed” [16]. Behdani optimally solved off-line multimodal schedule design in a way that may be reinterpreted for container-to-mode assignment [15]. However, his solution method relies on solving a large ILP, which may be too computationally intensive for on-line applications. Van Riessen provides another off-line container-to-mode solution in the form of his LCAT model and quantified the effect of on-line disturbances in the system [57].

In the field of on-line synchromodal container-to-mode assignment, several methods in different directions have been tried. Zhang et al. propose on-line container-to-mode assignment by assigning a request, when it comes in, to its cheapest path that still has capacity remaining [62]. This method completely disregards the capacity that will still be needed for future requests and one can easily think up examples where this disregard yields suboptimal results. Mes et al. also determine shortest paths in an on-line setting, but disregard capacity and instead focus on soft restrictions, decision-support and reducing the amount of computations that have to be done on-line [40].

In also regarding future assignments, the work of Rivera et al. seems more appropriate: they formulate the container-to-mode assignment problem as a Markov Decision Process and approximate its solution by means of Approximate Dynamic Programming [45]. Kooiman et al. investigate a more general spectrum of Time-stamp Stochastic Assignment Problems and a number of solution methods for them. They conclude that their most potent method is a simulation method [35], which Rivera's Approximate Dynamic Programming method resembles. Lium, Crainic and Wallace take this approach one step further: they 'discretise the possible futures' and optimise against the discretised expected value [39]. They underline that this discretisation has to be done in a way such that the optimisation is performed towards the problem and not towards its discretisation. They formulate some stability properties by which to measure this and study the effects of stochasticity and correlations on planning procedures.

Le Li et al. study cooperative synchromodality: their model assumes that a number of 'central' operators control their own subnetworks and that exchange between these networks is determined by negotiation [36]. It may be worthwhile to investigate whether this can be generalised to several settings where multiple agents make decisions towards their own goals and cooperative goals. They formulate the problem in a Distributed Model Predictive Flow Control setting and employ several solution methods from the field of systems and control theory. Their model does not address stochastic elements.

Of course, when modelling agent-centric optimisation, so various parties in a joint network working towards individual goals, it could make sense to study game theory. Though no application of game theory to synchromodal transport was found, Theys approached cooperative intermodality from a game-theoretical perspective [56]: that is to say, he studied how cooperation can still be stimulated when agents set their own interests first. He briefly explains how costs and benefits can be divided using a cost-allocation game, which can be solved with a Nucleolus if the game is subadditive. However, he also comments that for more realistic scenarios of intermodal transport, including those where trucks may be used, the cost-allocation game will often not be subadditive and more advanced game theory and computation power may be required. Game theory in operations research is more widely studied in the book by Baumol [14].

All in all, in the context of synchromodal transportation planning, this literature study suggests that movement towards synchromodality has been done in two separate directions. Firstly, in the studied literature, some deterministic methods are extended to optimise against stochastic elements and other interpretations of uncertain future, with various degrees of sophistication. Such methods are important for building solution methods that respect current and future real-time switching. They always balance computational efficiency against how comprehensively to be prepared for any future outcome. Computational efficiency, of course, is paramount for developing the on-line applications this research is devoted to. Secondly, to a limited degree, more complex control structures have been studied than just having one central control tower and no other parties with control and interests. Systems and control theory has been used to model several decision-makers working towards common and individual goals simultaneously in the context of synchromodal transport. Game theory has seen applications in multimodal transport and more broadly in operations research, but seemingly not yet in synchromodal transport. Most notably, no literature has yet been found that com-

bines dealing with uncertain futures and dealing with non-central control structures.

This research will not explicitly focus on the combining of these two directions. Rather, it will assume a simple control tower structure and investigate a wider range of possibly stochastic elements and control elements. That is to say, whereas most papers focus on one specific element being stochastic, such as request arrival times or barge travel times, this research will develop and investigate general stochastic optimisation methods for both container-to-mode assignment and operational scheduling against a wide choice of stochastic elements. Using this, a non-central control structure can be implied: if, for example, the handling processes at terminals are controlled by some other entity and the net operator knows nothing about these processes, the terminal handling times could be modelled as stochastic. This way, this research will address the absence of real-time operational planning methods that allow for real-time switching, for a class of logistic problems rather than for one given case.

3

Framework for synchromodal problems

This chapter is co-authored by Max Roberto Ortega del Vecchyo, graduate student at the department of Applied Mathematics, Delft University of Technology, by Myrte De Juncker, graduate student at the department of Applied Mathematics, Eindhoven University of Technology and by the author of this thesis.

When solving an optimisation problem, it is important to first properly define it. In this co-authored chapter, a framework is introduced by which to classify mathematical models described in the literature on synchromodal transportation problems. In this thesis, three different problems will be discussed; the framework will help in compactly describing the differences between these three problems. As such, this question seeks to answer the following research sub-question:

1. How can different scenarios of synchromodal freight transport be classified within an exhaustive framework?

Other reasons for such a framework will be elaborated on in the next section, but briefly put, this framework should help researchers and developers by pointing towards solution methodologies that are commonly used in their problem instance.

Section 3.1 will describe the context and motivation of the framework. Section 3.2 will introduce the classification framework and Section 3.3 will introduce a short-hand notation for it. In Section 3.4, some examples are provided. In Section 3.5, these examples are used to discuss strengths and weaknesses of the framework.

3.1. Motivation

Synchromodal transport is a topic that requires innovation from different fields of expertise. Pfoser describes the following seven critical success factors for synchromodality [46]:

1. Network, collaboration and trust
2. Awareness and mental shift
3. Legal and political framework
4. Pricing/cost/service
5. ICT/ITS technologies
6. Sophisticated planning
7. Physical infrastructure

From the list above, the different branches of knowledge become more clear. Roughly, it can be argued that the first and second factor are mainly social problems, the third is a political problem, the fourth is a mathematical, social and political problem, the fifth is a technological problem, the sixth is a mathematical problem, and the seventh is a technological and constructional problem. The framework is mainly interested in challenges within the realm of mathematics. These challenges are therefore directly related to the fourth and sixth success factors, but can also impact the first three in an indirect way, by promoting the effectiveness of synchromodality.

Synchromodal problems are often divided into three main categories: strategical, tactical, and operational. The same is true for mathematical synchromodal problems. These problems are related in a pyramidal-like structure in the following sense: tactical problems are usually considered where a specific strategical instance is given, and operational problems are frequently solved where a strategical and tactical structure is fixed. See also see Figure 3.1. Sometimes problems in two consecutive levels are solved simultaneously: for instance, Behdani determines how frequently a resource should be sent out, which is considered a tactical problem, while also allocating the flow of freight [15], which is considered an operational problem.

Mathematical synchromodal transportation problems on a tactical or operational level are usually represented via tools from graph theory and optimisation [53]. However, more often than not, the similarities end there: most of the models used to analyse a synchromodal transportation network are targeted to a specific real problem of interest [53], and knowledge and methods of other branches such as statistics, stochastic processes, or systems and control are often used. The models emphasise what is most important for the given circumstances. Consequently, mathematical synchromodal transportation problems on a tactical or operational level have been reviewed with approaches that may differ in many aspects:

- The exhaustiveness of the elements considered varies: for example, traffic conditions are considered in some models [37], but not all.
- The elements that can be manipulated and controlled may vary: for example, the departure time of some transportation means may be altered if suitable [15] or it may be that all transportation schedules are fixed [40].
- The amount of information relevant to the behaviour of the network may vary, and if a lack of information is considered, the way to model this situation may also vary [45, 62].
- Some models take other stakeholders with control into account and model their behaviour in different ways [36, 56], while many ignore other decision makers [15, 35, 40, 45, 62].

A model is not necessarily improved by making it increasingly exhaustive. As it happens with most model-making, accuracy comes with a trade-off, in this case, computational power. This computational burden is an intrinsic property of operational synchromodal problems [51] and one that is of the utmost importance, given the real-time nature of operational problems: new information is constantly fed and it should be processed in time.

There is no rule of thumb for making the decisions above [53]. Also, each of the decisions mentioned above will shape the model, and likely steer its solution methods to a specific direction. Though literature reviews of synchromodal transportation exist [51, 53], it appears no generalised mathematical model for synchromodal transportation problems has been found yet, nor a way of categorising the existing literature by their modelling approaches.

The framework for mathematical synchromodal transportation problems on a tactical or operational level presented in this chapter aims to capture the essential model-making decisions done in the model built to represent the problem. When no such model is specified, for example in simply defining a problem, it shows the model-making decisions likely to be done in

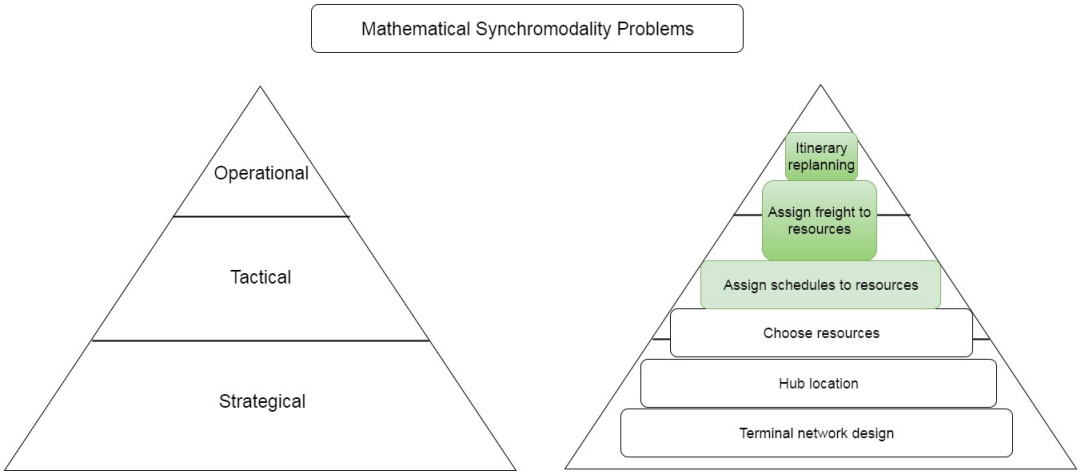


Figure 3.1: Two common categorisations of mathematical synchromodal problems. The three problems coloured in green are the ones considered in this framework: problems more tactical or strategical than assigning schedules to resources are outside the scope of this framework.

that case. Note that this 'likeliness' makes classification partly subjective. This classification is done in an attempt to grasp the characteristics of the model or case in a compact way, enabling easy classification and comparison between models and cases, as well as to see the complexity of a specific case at a glance. Also, it provides a perspective to better relate new problems with previous ones, thus identifying used methodologies for the problem at hand.

3.2. Framework identifiers and elements

The framework will refer to *demand* and *resources*. In synchromodal transportation models, demand will likely be containers that need to shipped from a certain origin to a destination. Resources can for example be trucks, train and barges. However, the framework allows for a broader interpretation of these terms. In repositioning problems, empty containers can be regarded as resources, whereas the demand items are bulks of cargo that need to be put in a container.

The framework has two main parts. The first part consists of the *identifiers*: these are specific questions one can answer about the model that depict the general structure of the model. The other is a list of *elements*: these elements are used to depict in more detail what the nature is of the different entities of the synchromodal transportation problem.

3.2.1. Identifiers

This section will elaborate on the identifiers of the framework. These identifiers are questions about the model. They identify the number of authorities, so how many agents are in control of elements within the model and they will also identify the nature of different elements within the model. The list of elements will be discussed in detail in Section 3.2.2, but they are used to determine which components in the model are under control, which are fixed, which are dynamic and which are stochastic. For instance, the departure time of a barge may be a control element, but it could also be fixed upfront, or modelled as stochastic. Some of the questions address how the information is shared between different agents and if the optimisation objective is aimed at global optimisation or local optimisation. All the answers on these questions together present an overview of the model, which can then be easily interpreted by others or compared to models from the literature.

The identifiers that discuss the behaviour of the model in more detail are discussed below.

1. *Are there other authorities (i.e. agents that make decisions)?*
   Here it is identified if there is one global controller that steers all agents in the network

or that there are multiple agents that make decisions on their own.

- *If there other authorities, how is their behaviour modelled:* one turn only, equilibrium *or* isolated*?*
  If the previous question is answered with yes, so there are multiple agents that make decisions, one needs to specify how these authorities react to each other. Three different ways are distinguished for modelling the behaviour of multiple authorities in a synchromodal network:
  - *One turn only*: this means that each agent gets a turn to make a decision. After the decision is made, the agent will not switch again. For instance, there could be three agents A, B and C. Agent A will first make a decision, then agent B and then agent C. The modelling ends here, since agent A will not differ from its first decision.
  - *Equilibrium*: the difference between "one turn only" and "equilibrium" is that after each agent has decided, agents can alter their decision with this new knowledge. In the same example: agents A, B and C make a decision, but agent A then decides to alter its decision due to the decision of agent B. If nobody wants to alter their decision any more, an equilibrium between the agents is reached.
  - *Isolated*: if the behaviour of the various authorities is isolated, it means that from the perspective of one of the authorities there is only limited information about the decisions of the other agents. For instance: agent C needs to make a decision. It is not known what agents A and B have chosen or will choose, but agent C knows historic data on the decisions of agents A and B. Agent C can then use this information to make an educated guess on the behaviour of agents A and B.

2. *Is information within the network global or local?*
   This identifies if the information within the network is available globally or locally. If the information is locally available, it means that only the agents themselves know for example where they are or what their status is at a certain time. If the information is global, this information is also known to the network operator, to all other agents or both.

3. *Is the optimisation objective global or local?*
   The same can hold for the optimisation objective. If all agents need to be individually optimised, the optimisation objective is local. If the optimisation objective is global, the best option for the entire network is the desired outcome.

4. *Which elements are controlled?*
   In a decision problem, at least one element of the system must be in control. For example: if one wants to determine which containers will be transported by a certain mode in a synchromodal network, the *demand-to-resource assignment* is controlled. If the problem is to determine which trains will depart at which time, the *resource departure time* is controlled. An extensive list of elements is given in Section 3.2.2. Of course, the controllable element can have constraints: perhaps the departure times of trains can be determined, but they cannot depart before a certain time in the morning. This is still a controllable element. Thus, an element is considered controllable if a certain part of it can be controlled.

5. *What is the nature of the other elements:* fixed, dynamic, stochastic *or* irrelevant*?*
   The other elements within the network can also have different behaviours. These four are distinguished:

   - *Fixed*: a fixed element does not change within the scope of the problem.
   - *Dynamic*: a dynamic element might change over time or due to a change in the state of the system, for example when the amount of containers loaded on a barge affects its travel time, but this change is known or computable beforehand.

- *Stochastic*: a stochastic element is not necessarily known beforehand. For instance it is not known when requests will arrive, but they arrive by a Poisson process. It might also occur that the time the request is placed is known, but the amount of containers for a certain request follows a normal distribution.
- *Irrelevant*: the list proposed in Section 3.2.2 is quite extensive. It might occur that for certain problems not all elements are taken into consideration to model the system. Then these elements are irrelevant.

6. *What is the optimisation objective?*
   This identifier is for the optimisation objective. One can look at the exact same system but still want to minimise a different function. One could think of travel times and CO_2 emissions. It is also possible to provide a much more specific optimisation objective. Examples of optimisation objectives are in Section 3.4.

3.2.2. Elements

In this section a list of elements is given that exist in most synchromodal transportation problems. They are divided in two parts: *resource elements* and *demand elements*. The resource elements are all elements related to the resources, which are mostly barges, trains and trucks. However, for compactness, terminals are also viewed as a resource. The demand elements are all elements related to the demand, which are most of the time freight or empty containers. Most elements mentioned in this list are straight-forward, but small clarifications are given if considered necessary.

- Resource elements:
  - *Resource Type* (RT): Different modalities can be modelled as different resource types. Another way to use this element is for owned and subcontracted resources.
  - *Resource Features* (RF): These features can be appointed to the different resource types or can have the same nature for the different types. For instance, it may be that there are barges and trains in the problem, but their schedules are both fixed, thus making the nature of the resource features *fixed* for both resource types.
    - *Resource Origin* (RO)
    - *Resource Destination* (RD)
    - *Resource Capacity* (RC): Indication of how much demand the different resources can handle.
    - *Resource Departure Time* (RDT)
    - *Resource Travel Time* (RTT): Time it takes to travel from the origin to the destination in the case of a moving resource.
    - *Resource Price* (RP): This can be per barge, train, truck, container or other resource item.
  - *Terminal Handling time* (TH): Time it takes to handle the different types of modes at the terminal. This can again be per barge, train, truck, container or other resource item.
- Demand elements:
  - *Demand Type* (DT): One can also think of different types of demand. For instance, larger and smaller containers or bulk.
  - *Demand-to-Resource assignment* ($D2R$): The assignment of the demand to the resources.
  - *Demand Features* (DF)
    - *Demand Origin* (DO)
    - *Demand Destination* (DD)
    - *Demand Volume* (DV): It might be that different customers have a different amount of containers that are being transported.

- ◊ *Demand Release Date* (DRD): The release date is the date at which the demand item is available for transportation.
- ◊ *Demand Due Date* (DDD): Latest date that the demand item should be at its destination.
- ◊ *Demand Penalty* (DP): Costs that are incurred when the due date is not met or when the container is transported before the release date, which is sometimes possible when coordinated with the customers.

3.3. Notation

In this section, some notation is introduced which will make it easier to quickly compare different models. Obviously, it is hard to keep a compact notation and still incorporate all aspects of a synchromodal system. Therefore, some of details are left out of the compact notation. When comparing models in detail, it is easier to look at all answers to the identifiers given in Section 3.2.1. The proposed notation has similarities to Kooiman's framework for Time stamp Stochastic Assignment Problems [35], Kendall's notation for classification of queue types [33], the notation of theoretic scheduling problems proposed by Graham, Lawler, Lenstra and Rinnooy Kan [28] and other frameworks.

A synchromodal transportation model is described by the notation:

$$R|D|S \quad \text{or} \quad R|D|S|B,$$

depending on whether or not there are other authorities in the system. The letters denote the following things:

- R: resource elements,
- D: demand elements,
- S: system characteristics,
- B: behaviour of other authorities (if applicable).

Resource and demand elements

The first two entries in the notation can be filled with all elements mentioned in the list in Section 3.2.2. As mentioned before, an element can be one of five different things: controlled, fixed, dynamic, stochastic or irrelevant. If the element of Demand-to-Resource assignment (D2R) is being observed, the following notation reflects the five possibilities for this element:

- controlled element: $[D2R]$,
- fixed element: $\overline{D2R}$,
- dynamic element: $\widetilde{D2R}$,
- stochastic element: $\widehat{D2R}$,
- element irrelevant: $\not{D2R}$.

Writing down all elements will still result in a large string of text. Therefore, it is suggested to use R and D for the most common aspect and further noting only the elements that are different. Suppose for example that for the resource elements everything is fixed, except the departure time, which can be controlled. This will be written as: $\overline{R}, [RDT]$.

If different resource types or demand types have a difference in some of the elements, one can also write this down by separating the classifications by brackets. Suppose for example: for barges everything is fixed, except for the dynamic capacity, but for the train the capacity is stochastic and the other elements are fixed. This can be noted in the following way: $\{\overline{R}, \widetilde{RC}\}, \{\overline{R}, \widehat{RC}\}$. Note that this is the only way in which the types are incorporated into the notation. To know which types are used in the different models, one has to look at the expanded notation. This choice is also made for the sake of compactness.

System characteristics

For the system characteristics, a notation was developed to immediately answer questions 1, 2 and 3 of the identifiers. Thus: are there other authorities, is the information global or local and is optimisation global or local?

The notation is based on Figure 3.2. In a similar way to this figure, the four options for the field *System characteristics* in the notation are:

- *selfish*: information global and optimisation local,
- *social*: information global and optimisation global,
- *cooperative*: information local and optimisation global,
- *limited*: information local and optimisation local.

Figure 3.2: Different models of a synchromodal network.

In order to see if there are other authorities within the system, either an (1) or (1+) is written behind the option chosen. If it is known how many authorities there are it also possible to denote that number between brackets. One could for example write down: social(1) or cooperative(1+).

Behaviour of other authorities

If there are other authorities within the system, question 1*a* in Section 3.2.1 states their behaviour should be known. The options are the same as discussed before:

- *one turn only*,
- *equilibrium*,
- *isolated*.

This field can be left blank if there are no other authorities in the system.

Remarks

The notation we developed does not include the optimisation objective. This is done on purpose. Within a specific model there is of course an option to look at different optimisation objectives. Since these might be quite elaborate, no way was found to shorten these objectives to a few words in a way that does them justice. This would only result in notation that needs more clarification. If a reader is interested in the optimisation objective, he or she can look at the entire list of identifiers instead of the compact notation.

This framework is developed in collaboration with multiple parties that study synchromodal systems. Therefore, many of the resource and demand elements that are most common in synchromodal problems are identified. However, for certain specific problems one might need to extend the framework. This can easily be done in the same way as the framework was set up. For example, one can add some elements within the list of elements or a different nature of one of the elements. However, one must keep in mind that the scope of the framework mainly covers mathematical problems on the operational and tactical levels.

3.4. Examples

As discussed earlier, one of the ideas of the framework is that, when starting work on a new problem, one can first classify the assumptions this model would need, then investigate papers that have similar classification. Therefore, a number of classification examples are presented in this section for both existing models as well as new problems. First, the identifier questions are answered for the Kooiman pick-up case [35] in Table 3.2. Then, it is shown how this can be written in the compressed notation. Afterwards, Table 3.3 shows compressed notation of some other problems described in papers, so that the interested reader can study more examples of the framework classification. Then, using Table 3.4, some real life cases are examined and possible classifications are given. These real life problems do not yet have an explicitly described model, so this classification is based on how someone *could* approach and model these practical problems, but other modellers may make other modelling decisions. Finally, the given examples will be used as input for discussion.

In the Kooiman pick-up case, a barge makes a round trip along terminals in a fixed schedule to pick up containers to bring back to the main terminal; however, the arrival times of the containers at the terminals are stochastic. At each terminal, a decision has to be made of how many containers to load onto the barge, and an estimate has to be made of how much capacity will be needed for later terminals, all while minimising the amount of late containers. The actual time of residing at the terminal is disregarded. In Table 3.2, the framework questions are answered. The reader is referred to Table 3.1 for a reminder of the framework element abbreviations.

RO: resource origin	DO: demand origin
RD: resource destination	DD: demand destination
RC: resource capacity	DV: demand volume
RDT: resource departure time	DRD: demand release date
RTT: resource travel time	DDD: demand due date
RP: resource price	DP: demand penalty
TH: terminal handling time	$D2R$: demand-to-resource assignment

Table 3.1: Abbreviations of the framework elements used in the compressed notation.

Other authorities	No
Information global/local	Global
Optimisation global/local	Global
Resource elements	RT: barges Controlled resource elements: none RF: fixed, except ~~TH~~
Demand elements	DT: freight containers Controlled demand elements: $D2R$ DF: fixed, except $\widehat{DRD}$
Optimisation objective	Maximise percentage of containers that travel by barge instead of truck

Table 3.2: The framework questions applied to the Kooiman pick-up case.

Note that only barges have been taken into consideration as resources, not trucks. It would have been possible to describe trucks as resources as well, but they are classified here as part of the lateness penalty, because there is no decision-making in how the trucks are used. Also, it may seem strange to speak of global or local information and optimisation when there are no other decision-making authorities. The information is considered global, because the only decision-making authority knows 'everything' that happens in the network; the optimisation is considered global, because the decision-maker wants to optimise the performance over all demand in the network put together, not over some individual piece or pieces of freight.

Using the framework notation, most of Table 3.2 can be summarised as follows:
$\bar{R}, \sout{TH} | \bar{D}, [D2R], \widehat{DRD} | social(1)$. Only the optimisation objective and type specifications are lost in this process.

In Table 3.3, the framework is applied to more problems from academic papers. In this table, the optimisation objectives are included to illustrate the wide range of optimisation possibilities. It is not actually necessary to describe the optimisation objective when using the compressed problem notation. In some cases, especially practical problem descriptions, optimisation objectives may not yet be explicitly known. Therefore, Table 3.4 leaves them out. In that table, some practical problem descriptions are reviewed and the framework is applied to these descriptions.

Behdani [15] $\bar{R}, [RDT] \mid \bar{D}, [D2R] \mid\mid social(1)$ Objective: minimise transportation costs and waiting penalties
Kooiman [35] $\bar{R}, \sout{TH} \mid \bar{D}, [D2R], \widehat{DRD} \mid social(1)$ Objective: maximise percentage of containers by barge instead of truck
Le Li [36] $\bar{R}, \sout{RDT} \mid \bar{D}, [D2R], \widetilde{DV}, \sout{DRD}, \sout{DDD} \mid cooperative(1+) \mid equilibrium$ Objective: with self-optimising subnetworks, minimise total cost in union
Lin [38] $\bar{R}, \widetilde{RC}, \sout{RP} \mid \bar{D}, [D2R] \mid\mid social(1)$ Objective: minimise total quality loss of perishable goods
Mes [40] $\bar{R}, \widetilde{RP}, \sout{RC} \mid \bar{D} \mid social(1)$ Objective: best modality paths against different balances of objectives
van Riessen [50] $\{\widetilde{R}, \bar{RO}, \bar{RD}, [RDT]\}\{\widetilde{R}, \bar{RO}, \bar{RD}\}, \bar{TH} \mid \bar{D}, [D2R], \widetilde{DP} \mid social(1)$ Objective: minimise transport and transfer cost, penalty for late delivery and cost of use of owned transportation
Rivera [45] $\widetilde{R} \mid \widehat{D}, [D2R] \mid\mid social(1)$ Objective: minimise expected transportation costs
Theys [56] $\bar{R}, [RP], \sout{RDT} \mid \bar{D}, [D2R], [DP], \sout{DRD}, \sout{DDD} \mid selfish(1+) \mid equilibrium$ Objective: fairest allocation of individual costs
Zhang [62] $\bar{R} \mid \widetilde{D}, [D2R] \mid\mid social(1)$ Objective: maximise balance of governmental goals

Table 3.3: For selected papers, a classification of where their problem falls in the synchromodal framework.

Lean and Green Synchromodal [4] $\bar{R} \mid \bar{D}, [D2R] \mid\mid selfish(1)$
Rotterdam – Moerdijk – Tilburg [7] $\bar{R}, \widehat{RTT}, \widehat{TH} \mid \bar{D}, [D2R] \mid\mid social(1)$
Synchromodaily [11] $\bar{R}, [RDT] \mid \widehat{D}, [D2R] \mid\mid social(1)$
Synchromodal Control Tower [8] $\bar{R}, [RC], \widehat{RP}, \widehat{RTT}, \widehat{TH} \mid \bar{D}, [D2R], [DV] \mid\mid social(1)$
Synchromodal Cool Port control [1] $\bar{R}, [RDT], \widehat{RTT} \mid \bar{D}, [D2R], \widehat{DDD}, \widetilde{DP} \mid\mid social(1)$

Table 3.4: For selected use cases, a classification of where a possible model for this problem would fall in the synchromodal framework.

Suppose now the reader wants to model an agent-centric synchromodal network. Here all agents want to be at their destination as fast as possible, but everyone does share the information about where they are and where they are going with everybody else in the network. Table 3.5 shows the answer to the questions of the framework. In the short notation this problem is:

$$\overline{R} \mid \widehat{D}, [D2R], \sout{DP} \mid \text{selfish}(1+) \mid \text{equilibrium}$$

Other authorities	Yes
Information global/local	Global
Optimisation global/local	Local
Resource elements	RT: barges, trains and trucks Controlled resource elements: none RF: fixed
Demand elements	DT: containers Controlled demand elements: $D2R$ DF: stochastic, except ~~DP~~
Optimisation objective	Minimise travel times

Table 3.5: The framework questions answered for the agent-centric synchromodal network.

3.5. Discussion

The given examples show some strengths and limitations of the classification framework, which are discussed in this section.

One of the goals of this framework was to offer guidance when tackling a new problem: as an example, if the problem from the Synchromodaily [11] case is modelled in a non-stochastic way, one can now see that it may be worthwhile to study the solution method presented by Behdani [15], because they then have the same compressed framework classification. If such a record is kept of papers and models, this could greatly improve the efficiency of developments in synchromodal transport. This would fulfil the second goal of the framework: to collect literature on synchromodal transportation within a meaningful order.

The final goal of this framework was to expose and compare relationships between seemingly different problems: for example, one can now see that the problems described by Le Li [36] and Theys [56] have similarities, in that they investigate negotiation between parties and do not focus on timeliness of deliveries. Similarly, one can see that the model assumption Mes [40] makes in disregarding resource capacity, is an unusual decision.

In the Synchromodaily case [11], the given interpretation of the problem implies that the demand features are stochastic. However, the problem could also be approached in a deterministic way, depending on choices that the modeller and contractor make based on the scope of the problem, the requirements on the solution and the available information. This shows the most important limitation of the classification framework: what classification to assign to a problem or model remains dependent on modelling choices, as well as interpretation of problem descriptions. Even without framework, however, modelling choices will always introduce subjective elements into how a real-world problem is solved. This framework can be used to consistently communicate these underlying model assumptions.

A second limitation of the framework is that, because of the large amount of elements described in it, two similar problems are relatively unlikely to fall in the exact same space in the framework because of their minor differences. Therefore, one should not only look for problems with the exact same classification, but also problems with a classification that is only slightly different. In a more general sense, solution methods may apply to far more than one of these very specific framework classes. If two problems have the exact same controlled elements, it is imaginable that their models and solution methodologies may largely apply to the other. As a point of future research, it could be interesting to investigate which classification similarities are likely to imply solution similarities, which may also be a stepping stone towards a general solution methodology.

As a final limitation, the compressed notation does not reveal that the paper by Lin [38] and the 'Synchromodal Cool Port control' [1] case both focus on perishable goods. This shared focus is not only cosmetic: mathematically, it may imply objective functions and constraints not focused on in other cases. To mitigate this limitation, anyone using the framework is

advised to offer both a compressed and an extended description of their problem or model.

3.6. Conclusion

In this chapter, a classification framework is offered to classify different mathematical synchromodal problems. If the Kooiman pick-up case [35] is considered, where barges with a fixed route have to decide how much containers to load at each location when the container arrival times are stochastic, one could classify this problem as $\bar{R}, \cancel{TH} | \bar{D}, [D2R], \widehat{DRD} | social(1)$, which signifies that this is a problem where:

- all resource features are fixed ($\bar{R}$), except that the terminal handling times (TH) are disregarded ($\cancel{TH}$);
- all demand features are fixed, except that the demand-to-resource assignment ($D2R$) is controllable ($[D2R]$) and the demand release dates (DRD) are stochastic ($\widehat{DRD}$);
- the problem is $social(1)$, so there is only one decision-making authority and the information and optimisation are both global.

A comprehensive list of elements is given in Section 3.2.2; a short one is given in Table 3.1.

The framework is helpful in the following ways:

1. When investigating a problem, classifying it will help point towards literature that has investigated similar problems;
2. The framework applies order to the dispersed research done in synchromodal transport;
3. The framework reveals similarities and common practices in modelling synchromodal problems, which may assist in working towards general solution methodologies.

Additionally, this thesis investigates three problems and their differences can be quickly seen by comparing their framework notations.

The most important limitation of the framework is that classification remains subject to the subjectivity inherent to modelling. Other limitations are that similar problems are relatively unlikely to have the exact same classification due to the extensiveness, and that information is lost on the optimisation objective and other priorities. For each of these limitations, it is suggested to combat them by more closely reading the concerned papers.

4

Deterministic container-to-mode assignment

In this chapter, the following research sub-question will be answered:

2. How can a low cost net-centric container-to-transport assignment be found fast enough for on-line use if everything is known beforehand?

With that, the first and most basic problem of this thesis is studied. With *deterministic container-to-mode assignment* (Problem 1), the following $\bar{R}|\bar{D},[D2R]|social(1)$-problem is meant:

> to assign freight containers to transports, so that the containers reach their destinations before a deadline against minimum total cost, given that the transports have fixed given schedules and all features of the problem are deterministic.

An example is given in Figure 4.1.

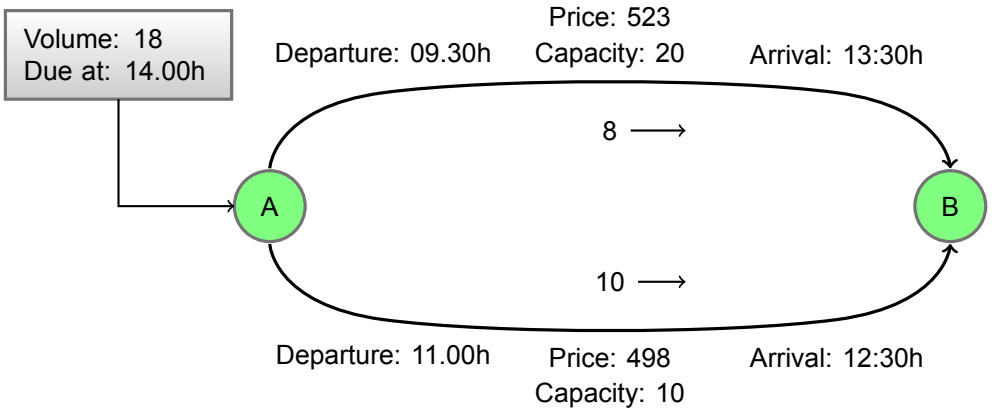


Figure 4.1: An example of deterministic container-to-mode assignment, or Problem 1. 18 containers at location A are due at location B at 14.00h. There are two available transits: the top one has departure time 09.30h, arrival time 13.30h, price 523 and capacity 20; the bottom one has departure time 11.00h, arrival time 12.30h, price 498 and capacity 10. The cheapest feasible container-to-mode assignment is to send 10 containers over the bottom transit and 8 over the top transit.

Deterministic container-to-mode assignment has several direct and indirect applications. The studied one is of planning container flows on an operational level, given some transports with fixed schedule and no stochastic elements. However, it is also reinterpretable to intermodal service network design on the tactical level, for example to determine weekly freight flows: this is merely a difference in time-scale and whether time 'loops' or not. Even

when considering stochastic elements, being able to solve deterministic container-to-mode assignment is useful for comparing what could have been possible if the future was known or predicted well enough. As a final use, Chapter 5 will contain stochastic container-to-mode subproblems that reduce to deterministic ones.

4.1. Modelling the problem as a minimum cost multi-commodity flow problem on a space-time network

The network represented in Figure 4.1 differs from classical graphs in that the edges have time restrictions attached to them. Though time-dependent graphs have been studied as such [24], this thesis uses the common technique of rewriting the graph to a space-time network [12, 29]. Solving Problem 1 can then be done by solving the non-negative integral minimum cost multi-commodity flow problem on such a space-time network. Both concepts are elaborated on in this section, as well as extensions with which to allow some lateness of deliveries and calling on trucks in times of need.

4.1.1. Space-time networks

Graph problems with time restrictions may benefit from being written as a *space-time network*. Such a representation can make the time restrictions explicit in the graph structure. A space-time network is a digraph where a node does not represent a physical place, but rather a physical place at a certain timestep. If $D = (V, A)$ is a digraph where the arcs have departure and arrival times, a space-time network $\mathcal{S}$ can be based on it in the following way:

- Pick a number of time steps T, based on the desired time window length and time discretisation fineness;
- For $v = 1, \dots, |V|$, $t = 0, 1, \dots, T$, define space-time node $s_{v,t}$;
- Let $\mathcal{S}$ have as node set the 'grid' $\{s_{v,t} | v \in V, t \in \{0, 1, \dots, T\}\}$;
- Initialise the arc set of $\mathcal{S}$ as all 'waiting arcs', in other words, all arcs in the set $\{(s_{v,t}, s_{v,t+1}) | v \in V, t \in \{0, 1, \dots, T-1\}\}$ with weight 0 and capacity ∞;
- For every arc a in A, translate it to an arc in $\mathcal{S}$ by finding the space-time nodes corresponding to the start and end points of a, then connecting them with the same weight and capacity as a. Add the resulting arc to the arc set of $\mathcal{S}$;
- Add 'truck arcs' to the arc set of $\mathcal{S}$, depending on the modelling choice for trucks that can be called on at any time. Some of the possible choices will be presented in Section 4.3.

In Figure 4.2, the network of Figure 4.1 is translated to a space-time network. This space-time network assumes that there are no trucks, thus no truck arcs. Indeed, the time restrictions described in Figure 4.1 are now embedded explicitly in the graph structure of the graph in Figure 4.2.

4.1.2. Minimum cost multi-commodity flow

In the previous section, it was shown how the studied problem can be represented in a space-time network, thus how the time dependencies can be made explicit in the graph structure. A question that was not answered, however, was how to optimally move the container demands from their origins in space-time to their due destinations in space-time. This can be done by solving the *non-negative integral minimum cost multi-commodity flow problem* (MCMCF) on the space-time network.

The MCMCF is the problem of moving flow of different types as cheaply as possible through a network where arcs have weights and may have limited capacity. More specifically, denote for some digraph $D = (V, A)$ and list of commodities K the following variables and parameters:

- The variables $x_{i,j}^k$ for the amount of flow of commodity k that is being sent over arc (i, j);

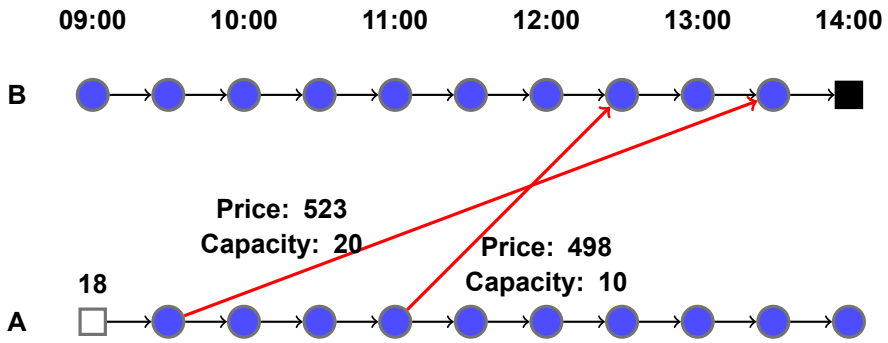


Figure 4.2: The network described in Figure 4.1, translated to a space-time network without trucks. The black arcs represent 'waiting arcs' with infinite capacity and cost zero. The 18 containers are situated at the white square node in space-time and need to go to the black square node in space-time.

- The parameters $c_{i,j}$ for the capacity of arc (i,j), so the maximal total amount of flow that can be sent over this arc;
- The parameters $f_{i,j}$ for the cost of sending one unit of flow of any commodity over arc (i,j);
- The parameters s_k for the source node from which the flow of commodity k emanates, and t_k for the sink node where all the flow of commodity k should go;
- The parameters d_k for the amount of flow of commodity k that emanates from s_k and needs to go t_k.

Then the MCMCF can be written as the following minimisation problem, in integer linear programming (ILP) form:

$$\begin{aligned}
\min \quad & \textstyle\sum_{(i,j)\in A}\sum_{k\in K} f_{i,j}x^k_{i,j} & & \\
s.t. \quad & \textstyle\sum_{k\in K} x^k_{i,j} \le c_{i,j} & \forall (i,j)\in A & (4.1)\\
& \textstyle\sum_{(s_k,j)\in A} x^k_{s_k,j} = d_k & \forall k\in K & (4.2)\\
& \textstyle\sum_{(i,t_k)\in A} x^k_{i,t_k} = d_k & \forall k\in K & (4.3)\\
& \textstyle\sum_{(i,v)\in A} x^k_{i,v} = \sum_{(v,j)\in A} x^k_{v,j} & (\forall k\in K)(\forall v\in V\backslash\{s_k,t_k\}) & (4.4)\\
& x^k_{i,j}\in \mathbb{N} & (\forall (i,j)\in A)(\forall k\in K) & (4.5)
\end{aligned}$$

Inequality (4.1) states that the total flow on any arc cannot exceed the arc capacity. Equality (4.2) states that for every commodity k, a total of d_k flow of commodity k must leave the source node s_k. Similarly, equality (4.3) states that exactly d_k flow of commodity k must enter the sink node t_k. Equality (4.4) states that in all other nodes, there must be flow conservation: whatever flow of some commodity enters the node, must also leave it. Finally, (4.5) states that the amount of flow of any commodity that may traverse any arc must be a natural number. The latter is essential for modelling the problem at hand, as for the practicality of the model, it is not allowed to put a non-integral or negative amount of containers onto a transit.

Now, Problem 1 can be modelled and solved as follows. Given some transport network with some fixed mode schedules, one can translate this to a space-time network. Next, every *batch* of containers that is released at some location and some time and that is due at some location and some time, can be interpreted as a commodity with corresponding d_k (volume), s_k (source node in the space-time network) and t_k (sink node in the space-time network). Finally, the optimal assignment can be seen as an instance of the MCMCF, and the corresponding ILP can be solved using an ILP solver.

The MCMCF and other linear cost multicommodity network flow problems are well studied, and the interested reader is referred to Kennington's survey [34].

4.1.3. Allowing lateness with virtual sinks

In the logistics business, deadlines are often considered to be soft [17, 55]. Henceforth, this report will refer to a *due date* as the time demand is supposed to be at its destination, which may be violated against some penalty. With *deadline*, a hard final date will be meant. A demand item is allowed to have both a due date and a deadline.

One may easily expand the given model to incorporate these due dates and deadlines with time-specific unit penalties. An example of this is given in Figure 4.3. The idea is to, for every commodity k, replace the original sink space-time node, t_k, by a virtual sink node v_k floating outside of space-time. Suppose the due location is L, the due time is t, the deadline is u and the time-dependent unit penalty is specified by some $(u-t)$-dimensional vector $\vec{p}$. Then the first node space-time node 'after' t_k, $s_{L,t+1}$, can be connected by an arc to v_k with the first entry of $\vec{p}$ as its weight and with infinite capacity. The same can then be done at times $t+2$, $t+3$, …, u. Of course, an arc with infinite capacity and cost 0 should be drawn from t_k to v_k to account for goods delivered on time. One may choose to implement connections beyond time u as well, using the last entry of $\vec{p}$ as weight; this is useful to guarantee feasibility in Chapter 5. The last entry of $\vec{p}$ can then be made arbitrarily high to imply deadlines. If they are all made arbitrarily high, this implies that the due date equals the deadline, on other words, that no delay is allowed. It is worthwhile to note that the presented extension makes the model applicable to $\bar{R}|\bar{D},[D2R],\widetilde{DP}|social(1)$ problems where the delay penalty depends arbitrarily on time and linearly on amount.

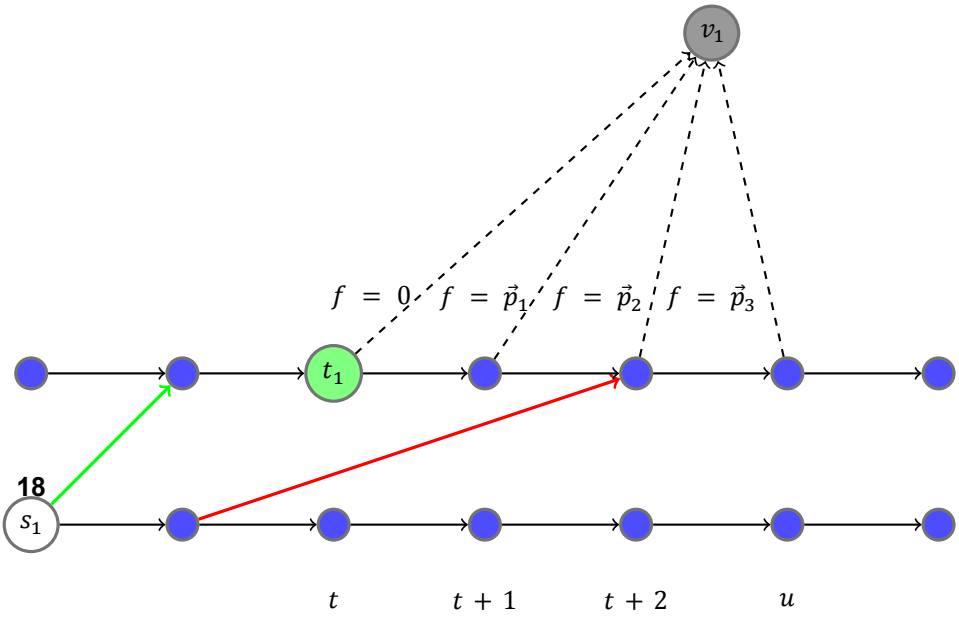


Figure 4.3: 18 containers should be delivered to t_1, but they are allowed to be up to three time steps late, if they are made to be due at virtual node v_1. If x containers are i time steps late, the penalty is $x \cdot \vec{p}_i$. If the green arc is very expensive and $\vec{p}_2$ is very low, it may be worthwhile to send containers over the red arc rather than the green arc. One may choose to connect the nodes beyond u to v_1 as well, against cost $\vec{p}_3$ or ∞, in order to guarantee the existence of feasible paths at any time step.

4.2. Solving to optimality

If there is only one batch of containers, the MCMCF reduces to a non-negative integral minimum cost flow problem. This problem is known to be solvable in polynomial time and space, because the continuous version is known to have an *integral solution polytope* [30]: in other words, every vertex of the solution polytope has integral coordinates. Every linear program has its optimal values in vertices of its solution polytope. LP solvers can find the optimal solution, thus the optimal vertex, in polynomial time and space, for example by using interior point methods. Therefore, the optimal solution of the LP-relaxation of the non-negative

integral minimum cost flow problem can be found efficiently. This gives a lower bound on the non-negative integral minimum cost flow problem. But because this optimum is in a vertex and all vertices have integral coordinates, this optimum has only integral flows, thus is a feasible solution to the non-negative integral minimum cost flow problem that equals a lower bound on the problem, thus is an optimum. In other words: if the LP-relaxation of a problem has an integral solution polytope, the integral optimum can be found simply by using an LP-solver.

Finding an non-negative integral two-commodity flow on a directed graph, however, is proven by Even to be NP-complete [25]. Finding one with minimum cost and with at least two commodities must be at least as difficult. In some cases, it is possible to easily rewrite a MCMCF as a single minimum cost flow problem, by creating a super source s' from which all sources receive their flow and a super sink t' where all sinks send their flow to. An example of this is given in Figure 4.4. However, the optimal single minimum cost flow cannot always be reinterpreted as a feasible multi-commodity flow, as shown in Figure 4.5.

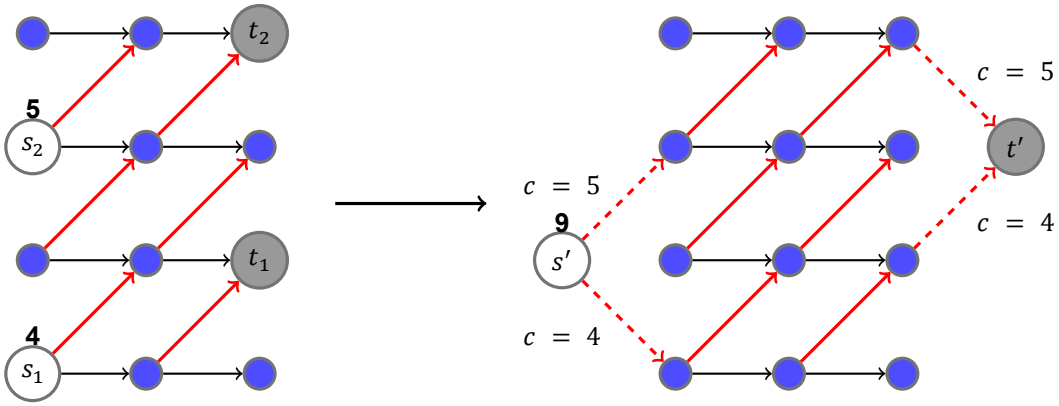


Figure 4.4: On the left, a two-commodity minimum cost flow problem on a space-time network, where all arcs have infinite capacity. On the right, an equivalent single-commodity minimum cost flow problem, where the dashed arcs have indicated capacity and all the others have infinite capacity. Every flow that is feasible in the network on the right, can be reinterpreted as a feasible flow in the network on the left, because no flow from s_1 can ever reach t_2 and vice versa.

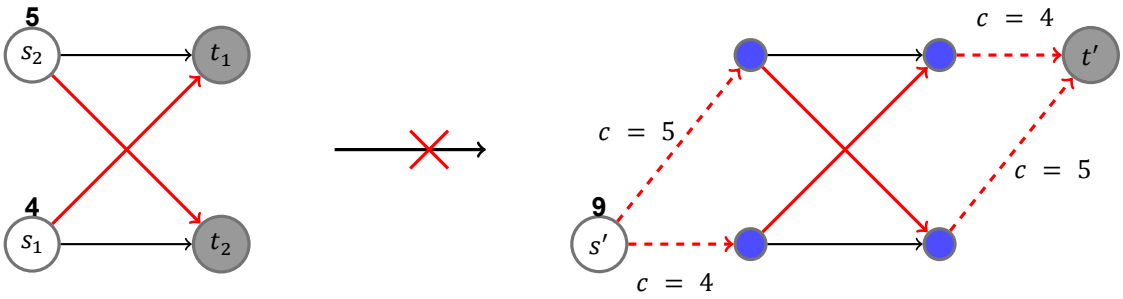


Figure 4.5: On the left, a two-commodity minimum cost flow problem on a space-time network, where all arcs have infinite capacity and the red arcs are more expensive than the black arcs. On the right, a single-commodity minimum cost flow problem on the same network but with a super source and super sink, where the dashed arcs have indicated capacity and all the others have infinite capacity. The optimal flow in the problem on the right will send 4 containers over the black arcs and 1 over a red arc, as the black arcs are cheaper than the red arcs. However, this result cannot be reinterpreted as a feasible flow for the problem on the left.

Surprisingly, however, the LP-relaxation of the studied MCMCF's almost always already has an integral optimum. The implication is that it almost always suffices to solve the MCMCF by means of an LP solver, which are typically fast, instead of an ILP solver, which are typically slow. In generating 500 random instances of MCMCF on space-time networks, all 500 had this immediate integrality. Ozdaglar noted that most of their non-integral optima were due to some form of perfect symmetry in a cycle [43], which could lead one to suspect that

digraphs *without directed cycle* do have an integral solution polytope. This, unfortunately, is not always true: Figure 4.6 shows a digraph without directed cycle which has no feasible non-negative integral two-commodity flow, but which does have a non-integral one. Figure 4.7 shows a space-time network based on this previous digraph. In this space-time network, two expensive truck arcs have been added to enable integral feasible flows. But even then, solving the LP-relaxation on the instance in 4.7 gives a non-integral optimum, as it will try to squeeze itself through the cheap red network isomorphic to the one in Figure 4.6.

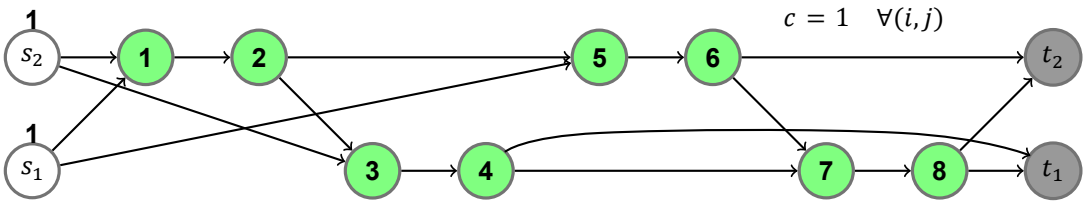


Figure 4.6: A digraph without directed cycles. Assuming that each arc has capacity 1, it has no feasible non-negative integral two-commodity flow, but it does have a non-integral one: namely, send half a unit of commodity 1 over $(s_1, 1, 2, 3, 4, t_1)$ and the other half over $(s_1, 5, 6, 7, 8, t_1)$ while sending half a unit of commodity 2 over $(s_2, 1, 2, 5, 6, t_2)$ and the other half over $(s_2, 3, 4, 7, 8, t_2)$.

The LP-relaxations of these examples have non-integral optima, because the cheap paths are unfortunately 'intertwined'. When randomly generating an instance by drawing cheap arcs between random points in space-time, one may expect that a situation like this seldom occurs. However, it may occur quite often in practice, when a handful of barges travel over a roughly similar sequence of locations and overtake each other. Therefore, without knowing or specifying the dynamics of the system, it is difficult to say how often this case of a non-integral LP-relaxation optimum will occur.

What can be predicted, however, is that they are unlikely to influence computational time much. In the given examples, as soon as one of the non-integral flow variables is branched upon, the resulting branches do have integral LP-relaxation optima. This property is due to the fact that the conflicts caused by these entwinings are 'resolved' by giving one of the paths priority. As branch-and-bound applications first check whether the LP-relaxation has an integral optimum, the advise is to simply use ILP solvers: if their search trees do not have depth 1, they should have depth no more than the amount of commodities caught in each entwining, given that resolving entwinings does not create new entwinings.

A final note made on this problem, is that the problem and its ILP may lend themselves well to Lagrange relaxation on the capacity constraints: in other words, to punish flows that exceed capacity, but not forbid them. Lagrange relaxations are known to give bounds at least as strong as LP-relaxations [32], and the optimum of the Lagrange relaxation can probably be computed easily: without the capacity constraints, the problem reduces to a shortest path problem for each request. These Lagrange relaxations are irrelevant when the solution polytope is indeed integral, but may give improvements for non-integral solution polytopes or help in developing shortest path-based heuristics.

4.3. Infinite resource models and corresponding graph reductions

An assumption that is sometimes made in literature [35] is that freight can always be transported from any location to any other location by trucks with infinite capacity and a given speed, but that trucks are more expensive than other modalities. Instead of trucks, one could also model the option of subcontracted transport that ensures timely arrival against a price, without much mathematical difference. This method of modelling will prove useful in Chapter 6 to limit the sheer amount of vehicles to explicitly control. From this point, a distinction is made between *infinite resources*, like a separate truck department or subcontracted transportation, and *vehicles*, like controlled barges and trains. To steer intuition, the

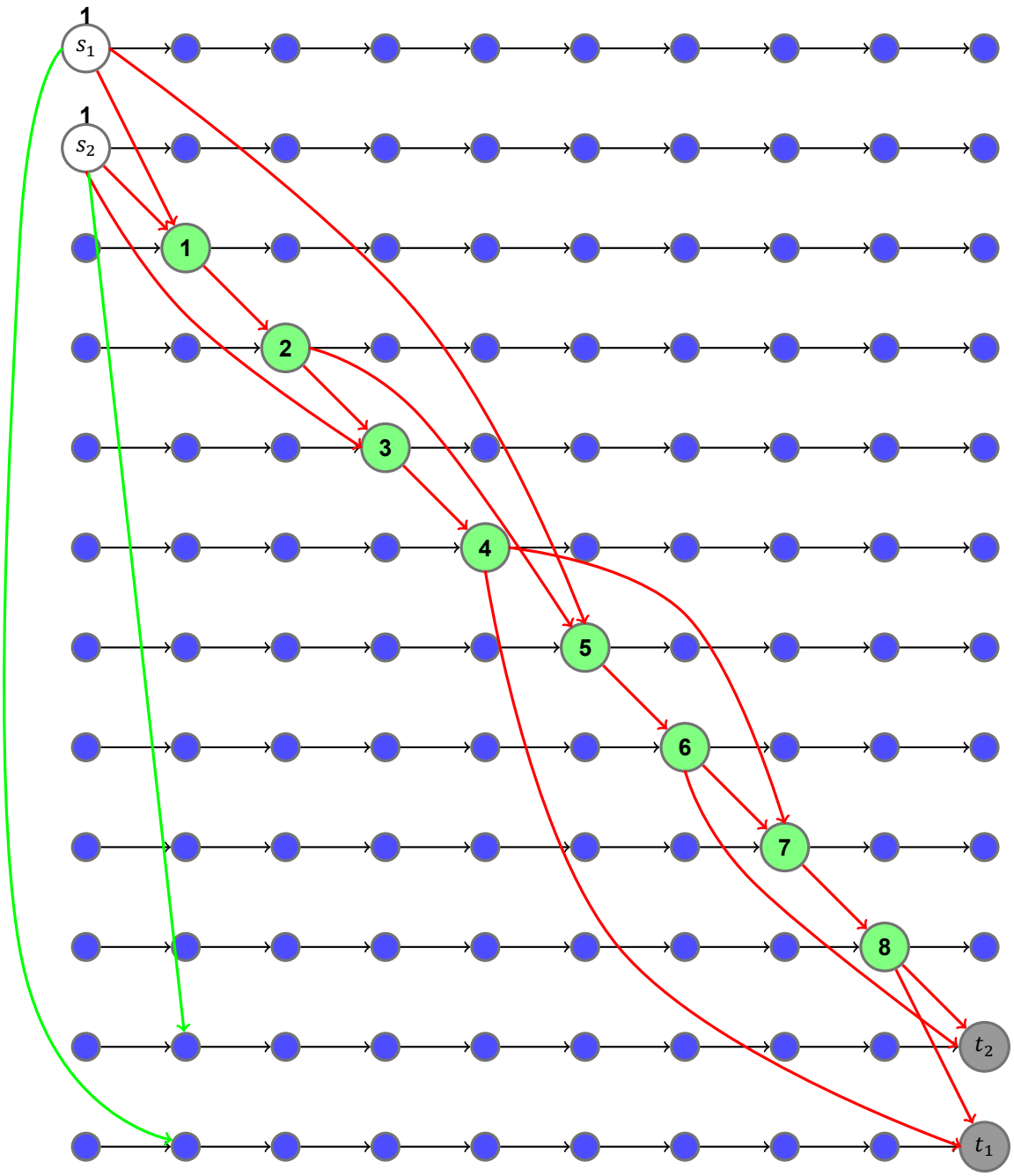


Figure 4.7: An instance where the LP-relaxation of the MCMCF has a non-integral optimum, given that the red arcs are cheap and have capacity 1, while the green arcs are expensive and have capacity 100. There is an integer feasible point, namely by sending all flow over the green arcs. However, it is possible and cheaper to send the flow only over red arcs; this subgraph is isomorphic to the one in Figure 4.6.

terms 'trucks' and 'infinite resources' will sometimes be used interchangeably throughout this report.

If infinite resources are modelled, so it is allowed to send containers from any place to any other place at any time against a high price by for example trucks, this would imply that a lot of truck arcs should be added to the previously discussed space-time networks to incorporate this flexibility. However, adding a truck arc to the space-time network from any location to any other location at any time comes at the cost of adding $n(n-1)\cdot(T-1)$ arcs, thus $n(n-1)\cdot(T-1)\cdot|K|$ variables, where n is the number of locations in the system, T is

the final time step and K is the set of observed commodities. Even though it was argued that ILP solvers should find an optimum in relatively little time, the underlying LP solvers do become significantly slower when an excessive amount of variables are added. The overall computational performance can be significantly improved by excluding unnecessary nodes and arcs in the space-time networks and conflating waiting arcs where possible. How such a reduction process can be done, depends on the modelling choices for the trucks. Therefore, this section describes two possible models for trucks with a corresponding graph reduction.

4.3.1. Double matrix infinite resources

Suppose that, indeed, the following assumptions are made:

- Trucks or other infinite resources can always be employed from anywhere to anywhere with infinite capacity;
- The infinite resources have a fixed travel time $M^1_{i,j}$ that depends only on origin-destination pair (i,j);
- Transporting containers with an infinite resource from location i to location $j \neq i$ has a unit cost $M^2_{i,j}$ per container that depends only on origin-destination pair;
- Triangle inequality: trucking from A to B directly is always cheaper than through C.

Allowing such trucks has two important merits: it is now possible to send containers by truck if all other modality-paths are infeasible in time or capacity, and under smart choices of the matrices M^1, M^2, the model now more closely simulates the practice of dividing a journey up into pre-haul, long haul and post-haul.

Figure 4.8 shows a space-time network where it is always possible to take a truck from any location to any other location in one time step. The truck arcs have high cost and infinite capacity. This space-time network has a lot of redundancy:

- Time step 0 is superfluous;
- Because of the triangle inequality, location A is useless;
- First trucking from C to D and then waiting one time step is equivalent to first waiting one time step and then trucking from C to D. Only one truck arc is needed to represent this option of trucking. In order to facilitate synchromodality, so to facilitate keeping options open until more has become known, one could choose to truck only at the last minute to go to t_k;
- Similarly, if flow is to travel over some non-truck transit arc, one could truck only at the last minute to catch that transit;
- If the waiting arc from $s_{C,3}$ to $s_{C,4}$ is used by some flow, then that same flow will always use the waiting arc from $s_{C,4}$ to $s_{C,5}$, so they might as well be joined into one arc.

From the above observations, it becomes clear that many nodes and arcs can be removed from the network in Figure 4.8 to obtain a reduced instance that has the same optimum, but involves a lot less arcs, thus a lot less variables, thus a lot less computation to solve. Most importantly, many redundant paths can be removed by allowing trucking only at the last moment, either to go to the (non-virtual) sink or to catch some transit. Rather than first generating a full space-time network and then removing arcs and nodes, it requires significantly less computation to already exclude all unnecessary arcs and nodes during generation. The employed way to generate such a reduced instance is described as Algorithm 1.

Using Algorithm 1, one could obtain a reduced instance to replace the instance shown in Figure 4.8. This reduced instance is shown in Figure 4.9. Aside from creating a reduced space-time network, one can also discard the flow variables for all but one commodity on the arcs that end in a virtual sink.

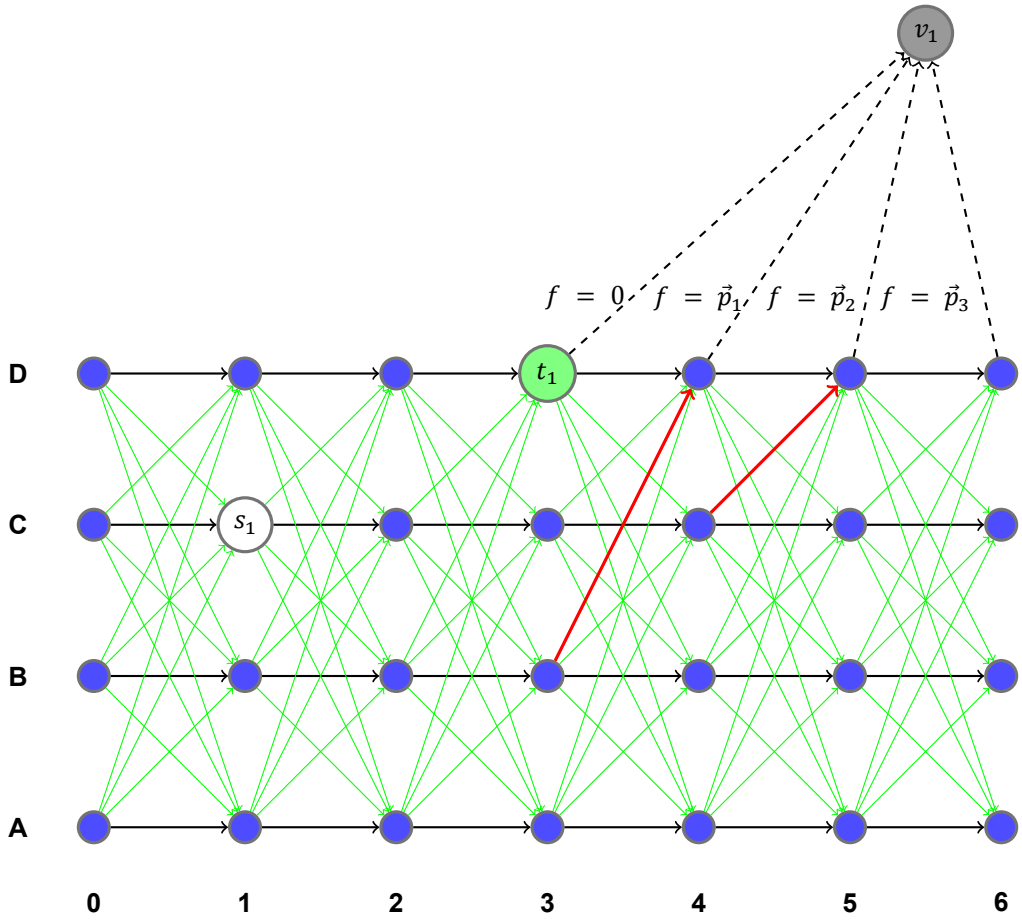


Figure 4.8: An unreduced space-time network with double matrix infinite resources: it is possible to truck with infinite capacity from any location to any other location at any time against some high price. This price depends only on origin-destination pair. Regardless of origin-destination pair, the travel time is always one time step. The thin green arcs are these expensive truck arcs with infinite capacity. This network contains a lot of redundancy under these assumptions: trucking immediately from C to D and then waiting one time step is equivalent to the converse.

It can be seen that more reduction is possible still: for example, the node at $(B, 2)$ is useless. More importantly, under the current method, more and more nodes will remain preserved as time progresses, because they could technically act as pre-sink nodes from which to depart to a virtual sink if indeed all nodes after the deadline are still connected for feasibility. However, one may see by inspection that the pre-sink node at $(D, 6)$ in Figure 4.9 is useless, assuming that lateness penalties are non-decreasing. Aside this issue of more and more nodes being marked as relevant, there should also be opportunities in not always allowing trucking to catch some transit, but only if the added truck price and transit price are lower than the cost of immediate trucking to the transit's destination.

But in spite of these possible improvements, the current algorithm already fares well in combating the growth of unnecessary truck arcs. The instance in Figure 4.8 has 102 arcs, thus 102 variables as there is only one commodity, whereas the instance in Figure 4.9 has only 19 arcs. While more ideas for instance reduction are imaginable, they will be left for further research.

Data: List of locations, list of requests (volume, due time, deadline, due location, release time, release location, lateness penalty vector $\vec{p}$) and list of vehicle transits (capacity, unit price, departure time, departure location, arrival time, arrival location), infinite resource travel time matrix M^1, infinite resource travel time matrix M^2
Result: Instance of MCMCF on a reduced space-time network
Initialise an empty space-time network $\mathcal{S}$;
Determine first time step T_1 as first release time among requests;
Request a final time step T_2, or generate it by some means;
Eliminate all locations that are not interesting, so that are not a request's source or sink node or a transit's departure or arrival location;
for *request in list of requests* **do**
  Add space-time source node to $\mathcal{S}$, if not already present;
  Add virtual sink node to $\mathcal{S}$;
  for τ *in* $due\ time, due\ time\ + 1, \dots, T_2$ **do**
    Add space-time pre-sink node at $(due\ location, \tau)$ to $\mathcal{S}$, if not already present;
    Add arc from space-time pre-sink node to virtual sink node with infinity capacity and weight 0 if $\tau = due\ time$ and with weight dictated by $\vec{p}$ otherwise;
  end
  Add a truck arc to $\mathcal{S}$, if not already present, from any $(i = location \neq due\ location = j, due\ time\ - M^1_{i,j})$ to $(due\ location, due\ time)$ with infinite capacity and weight dictated by $M^2_{i,j}$, adding nodes if necessary;
end
for *transit in list of transits* **do**
  if $departure\ time \geq T_1$ *and* $arrival\ time \leq T_2$ **then**
    Add arc to $\mathcal{S}$, with supplied weight and capacity, adding nodes if necessary;
    Add a truck arc to $\mathcal{S}$, if not already present, from any $(i = location \neq departure\ location = j, departure\ time\ - M^1_{i,j})$ to $(departure\ location, departure\ time)$ with infinite capacity and weight dictated by $M^2_{i,j}$, adding nodes if necessary;
    For every request with $due\ time \geq arrival\ time$, add a truck arc to $\mathcal{S}$ if not already present from $(i = arrival\ location, arrival\ time)$ to $(j = due\ location, arrival\ time\ + M^1_{i,j})$ with cost $M^2_{i,j}$;
  end
end
Add horizontal waiting arcs between space-time nodes ;

Algorithm 1: Generating a reduced instance of MCMCF on a space-time network, given that infinite resources are modelled as double matrix infinite resources.

4.3.2. Other or no infinite resources

If there are no trucks in the model, then of course, no truck arcs need to be generated. However, one could also choose not to generate truck arcs in the fashion of Section 4.3.1 because one of the key assumptions does not hold: perhaps trucks have limited capacity, or they cost more in the weekends than on weekdays, or the trucks have to be explicitly modelled to guarantee feasibility. Many truck models in this wider range of options could still be modelled by giving them space-time arcs as one would with any other modality. The necessity of generating a reduced instance would greatly depend on how many truck arcs are added this way, and the methodology of generating a reduced instance would greatly depend on the assumptions that do still hold. Therefore, further commentary on reduction methods is not provided here, save two remarks. A reduction that should still be applicable in most cases, is the conflation of waiting arcs: if a node has a waiting arc as its only incoming arc and a waiting arc as its only outgoing arc, the node may as well be removed. Additionally, one can always discard, for arcs going to a virtual sink, any flow variable that is not of the

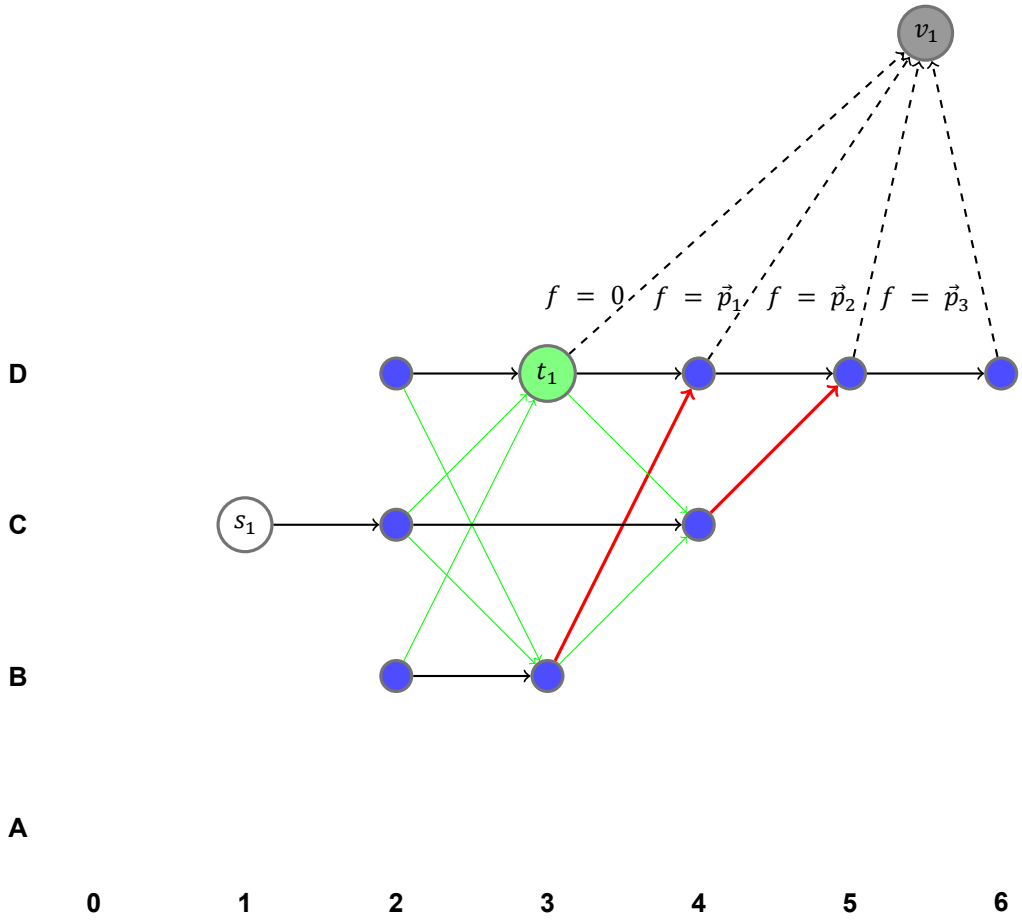


Figure 4.9: A reduced version of the instance in Figure 4.8, obtained using Algorithm 1. This reduced instance has only 19 arcs instead of the original 102. Further reduction should still be possible: for example, location A is useless here, as is the final pre-sink node at $(D, 6)$.

commodity that the sink belongs to. For the rest, the reader is advised to construct some reduction methodology, if necessary, by observing the given assumptions and Algorithm 1.

4.4. Numerical results

In this chapter, it was shown that Problem 1 can be solved to optimality by solving a minimum cost multi-commodity flow problem on a space-time network. It was argued that this can be done in relatively little time, because of the rarity of instances where the LP relaxation has a non-integral optimum. Furthermore, a problem reduction procedure was shown in the form of Algorithm 1 that should decrease computation time. In this section, numerical results are presented to support this claim.

In the experiments, 10 random instances were generated in each of four test classes. These instances were made by generating lists of random transits and requests, given a time-scale of the problem and a list of locations. The transits had random capacity, duration, departure time and price, though the prices were always considerably below the standard truck price. The orders had random volume, release time, due time and penalty vector, including instances where the penalty would become arbitrarily large to simulate hard deadlines. Each instance in test class $i = 1, 2, 3$ concerned $5i$ random requests and random transits over $5i$ locations, within a time scale of $5i + 1$ time steps. Test class 4 contained instances of 'real life size': based on data from a NWO-affiliated use case, instances in this class concerned 40 random requests and 60 random transits over 32 locations and 121 time steps. This was the minimum size specified to reflect operational use, where 121 time steps correspond to

the 120 hours of a five day working week. Though the instances in this class are scaled to the required size of the use case, the random placement of the transit arcs does not reflect scheduling procedures from the use case: rather, they reflect the 'pandemonium' a net operator may experience when supervising container-to-mode assignment over uncontrolled vehicles.

For each of these test classes, every instance was solved with and without the reduction from Algorithm 1, assuming double matrix infinite resources in both cases. The objective values, of course, were equal and optimal in both cases: rather, the solution two methods were compared in running time. When solving an instance, it was randomly determined which method to solve it with first, so as to mitigate cumulative slowdown bias. Also, for each of the 80 ILP's solved this way, it was monitored if indeed its LP relaxation had an integral optimum. The methods were implemented in Python 3.4, using the PuLP-library and its built-in non-commercial MILP solver. The experiments were run on an Intel(R) Core(TM) i7-6600U CPU @ 2.60GHz 2.80 GHz processor. The results can be viewed in Table 4.1. These

	Class 1	Class 2	Class 3	Class 4
Average running time with reduction (in seconds)	0.182	0.430	0.884	8.889
Variance of the above (in squared seconds)	$3.44 \cdot 10^{-4}$	$3.41 \cdot 10^{-3}$	$4.80 \cdot 10^{-3}$	0.948
Amount of instances with integral LP-relaxation optimum	10	10	10	10
Average running time without reduction (in seconds)	0.2972	1.046	3.061	36.33
Variance of the above (in squared seconds)	$1.07 \cdot 10^{-3}$	$4.90 \cdot 10^{-3}$	$3.07 \cdot 10^{-2}$	0.506
Amount of instances with integral LP-relaxation optimum	10	10	10	10

Table 4.1: Numerical results for deterministic container-to-mode assignment, or Problem 1. Class 4 contains instances of a size that was specified as useful within an operational use case.

results support the following valuable conclusions:

- Algorithm 1 significantly reduces computation time;
- Without or without reduction, this problem is solved to optimality fast, even in instances of 'real life size';
- The variation in computation time grows faster when using Algorithm 1, probably due to the variation in how much reduction is possible;
- Among the 80 ILP's, none was encountered of which the LP-relaxation had a non-integral optimum.

4.5. Discussion

The methods described in this chapter have certain strong merits:

- An optimal solution can be found in surprisingly little time, given that the problem is NP-hard;
- The method naturally models the option of consolidation.

However, there are still points of attention that could be investigated in future work:

- The given methods may still be not fast enough when used in iterative methods in an on-line setting. A heuristic or further graph reduction may still be required;
- Possibilities for further graph reduction were mentioned, but not explored;
- There is currently no penalty in cost or time for containers switching modality, which may not model reality well;

4.5.1. Added value

Solving this problem by solving a min-cost multi-commodity flow on a space-time network is not an entirely new idea [44, 53]. Where, then, lies the added value of this research?

- It has been proven that these methods can be translated well from tactical problems to operational problems and that they are fast enough for operational on-line use;
- The possibility to model soft deadlines was added;
- A significant reduction of computation time was achieved using Algorithm 1;
- A partial explanation for the low computation time was found by investigating graph theory, and the notion that an LP-solver is sufficient rather than an MILP-solver was shown to be true in 100% of the randomly generated instances.

4.6. Conclusion

This chapter sought to answer the following research sub-question:

2. How can a low cost net-centric container-to-transport assignment be found fast enough for on-line use if everything is known beforehand?

Deterministic container-to-mode assignment, or 'Problem 1', was defined to be the problem of assigning freight containers optimally to transport modalities with predetermined schedules, in the presence of time constraints and the absence of stochastic elements. It can be solved to optimality by solving the non-negative integral min cost multi-commodity flow problem on a space-time network. This model can be expanded to allow lateness, by introducing virtual sinks. Due to NP-hardness, solving the problem requires the use of an ILP solver. In all randomly generated instances, however, the LP-relaxation has an integral optimum. In a carefully constructed instance, this was not the case, but only one branching was required. Therefore, it is was hypothesised that ILP solvers will find an optimal solution relatively quickly in almost all cases.

The option was added to truck from anywhere to anywhere with infinite capacity and an origin-destination dependent unit cost. Algorithm 1 generates a reduced instance under this truck model, in order to exclude a large amount of superfluous arcs and variables. Further reductions are possible and recommended for future research.

Random instances of different sizes were generated to test this solution methodology on, with or without the reduction from Algorithm 1. They show that problems of a 'real life size' can be solved to optimality in an average 8.889 seconds, even when using a non-commercial MILP solver on a single computer. This makes the method suitable for on-line use.

To answer the sub-question: by solving the min cost multi-commodity flow problem on a space-time network with an MILP-solver, an optimal assignment can be found in a matter of seconds for problems of 'real life size'.

5

Stochastic container-to-mode assignment

In this chapter, the following research sub-question will be answered:

3. How can a low cost net-centric container-to-transport assignment be found fast enough for on-line use if new data is still expected to come in?

In the previous chapter, a method was developed to solve deterministic container-to-mode assignment relatively efficiently. One of the important classifiers of synchromodal transport is being able to adjust plannings in an on-line fashion as more information becomes available. A natural consequence of this is that decisions have to be made while some parameters are still uncertain. This is also often the case in operational planning in practice: if one cheap modality is 'probably' going to run into delays in the port area, and another modality is more expensive but also more likely to be on time, what decision should be made? Figure 5.1 shows how such a decision could be seen as a stochastic optimisation problem.

Though Figure 5.1 shows a problem where only the travel time of a modality is stochastic, uncertainties can manifest in many components of the transport chain. In literature, the travel time is often modelled as uncertain [54, 61], for example due to weather conditions, vehicle breakdowns, road congestion or general unpredictability. The demand patterns are also often considered uncertain [54, 61]. Sometimes, handling time at terminals are considered uncertain due to disturbances [61]. Aside from these more common factors, however, some models also study uncertainty in things like price and capacity of resources. Industry experts in operational planning affirm that 'any element' of transport comes with uncertainties, and Caris confirms that "real-life operational management is characterized by uncertainty" [19]. Even the total capacity of a barge, which one would almost always consider fixed, can be stochastic: for example, if a barge has to sail under a low bridge, then the maximum stacking height can become dependent on water levels. As such, a complete consensus on what uncertainties to model does not exist.

Where, then, does one draw the line? The move towards synchromodality is the move towards cooperation, which requires methods to remain applicable when different parties with different dynamics are integrated into the planning network. Therefore, in this thesis, methods are developed with a holistic view towards uncertainty: all elements proposed in Chapter 3 are modelled as stochastic, if not controlled. In particular, this chapter will study *stochastic container-to-mode assignment* (Problem 2), by which the following $\widehat{R}|\widehat{D},[D2R]||social(1)$-problem is meant:

> to assign freight containers to transports, so that the containers reach their destinations before a deadline against minimum total costs, given that the transports have fixed given schedules and some features of the problem are stochastic.

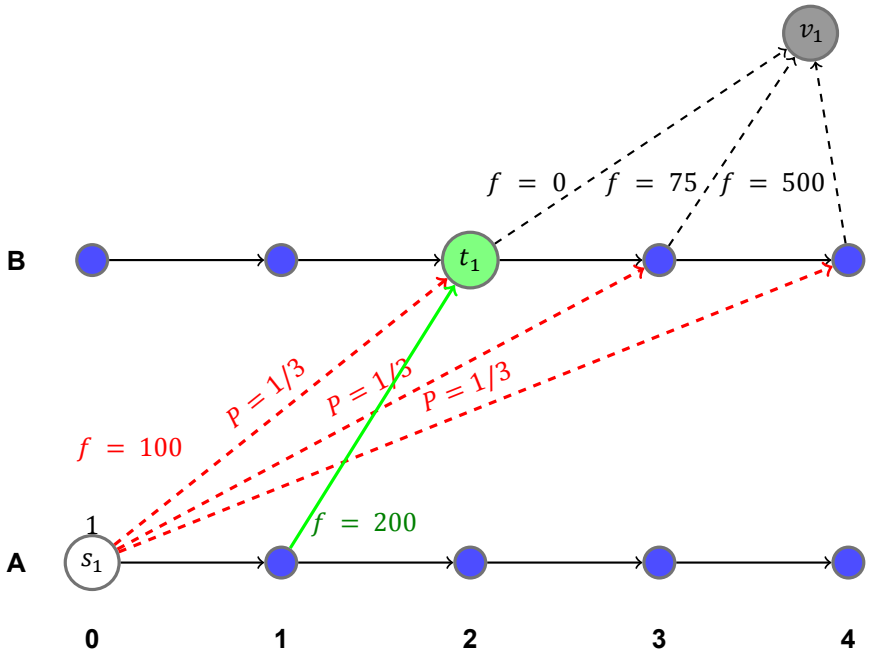


Figure 5.1: One container has to be assigned to a transportation modality. The red, dashed option has probability $1/3$ of taking two time steps, thus arriving just on time, probability $1/3$ of taking three time steps, causing a lateness penalty of 75 and probability $1/3$ of taking four time steps, causing a lateness penalty of 500. The green option is guaranteed to be on time, but has a base usage cost of 200 instead of 100. Both transports have capacity 10. At time $t = 0$, a planner is faced with the decision: red or green?

5.1. Concepts and definitions

In Chapter 4, the concept of solving min-cost multi-commodity flow on a space-time network was introduced. However, Figure 5.1 shows an example where a decision has to be made on some form of 'approximate' space-time network, where it is still uncertain which of its three possible outcomes it will eventually be. In this section, new concepts will be introduced from which to construct decision-making methods.

5.1.1. Transit Ideas and transit instances

The first gap that will be addressed, is that a planning algorithm can no longer plan based on a list of fixed transits, but on a list of transits with yet uncertain characteristics, as also illustrated in Figure 5.1.

In the philosophical work of Plato, the concept of an Idea is proposed: that all physical objects are instances or 'shadows' generated by some higher object. For example, in his theory of Ideas, every hammer on Earth is a different instance of one single Idea of a hammer [48]. This theory will not be discussed here, but it will be used as an analogy for the following definition.

A *transit Idea* $\mathfrak{T}$ is defined by a sextuple $\mathfrak{T} = (\mathfrak{T}_{OD}, \mathfrak{T}_{DT}, \mathfrak{T}_{TT}, \mathfrak{T}_C, \mathfrak{T}_P, \mathfrak{T}_\sim)$ where:

- $\mathfrak{T}_{OD}$ is a random variable representing the origin-destination pair of the transit;
- $\mathfrak{T}_{DT}$ is a random variable representing the departure time of the transport, so the time at which the transit starts;
- $\mathfrak{T}_{TT}$ is a random variable representing the travel time of the transit;
- $\mathfrak{T}_C$ is a random variable representing the capacity of the transport performing the transit;
- $\mathfrak{T}_P$ is a random variable representing the unit price of the transit per container transported by it;

- $\mathfrak{T}_{\sim}$ is a joint probability distribution on the random variables. If the random variables are independent, this probability distribution equals the product of marginal distributions.

In some problems, not all of these features are stochastic: for example, it is common to model transport capacity as fixed [15, 35, 45, 62]. This can be solved by making the corresponding random variable equal to this fixed constant with probability 1. In the notation of transit Ideas, such a random variable will then be replaced by its fixed outcome. Having remarked this, the red dashed link in Figure 5.1 correspond to the following transit Idea $\mathfrak{T}^{red}$:

$$\mathfrak{T}^{red} = ((A, B), 0, X, 10, 100, U_X\{2, 4\})$$

where $P(X = 2) = 1/3$, $P(X = 3) = 1/3$ and $P(X = 4) = 1/3$. So $\mathfrak{T}^{red}$ will certainly instantiate with origin-destination pair (A, B), departure time 0, capacity 10 and unit price 100, but the travel time X is drawn from a discrete uniform distribution.

When each of these five random variables are drawn, the result is henceforth called a *transit instance*. A transit instance corresponds to a fixed arc of a space-time network and is no longer subject to any randomness. In Chapter 4, exclusively transit instances were observed. The difference between transit Ideas and transit instances is also illustrated in Figure 5.2.

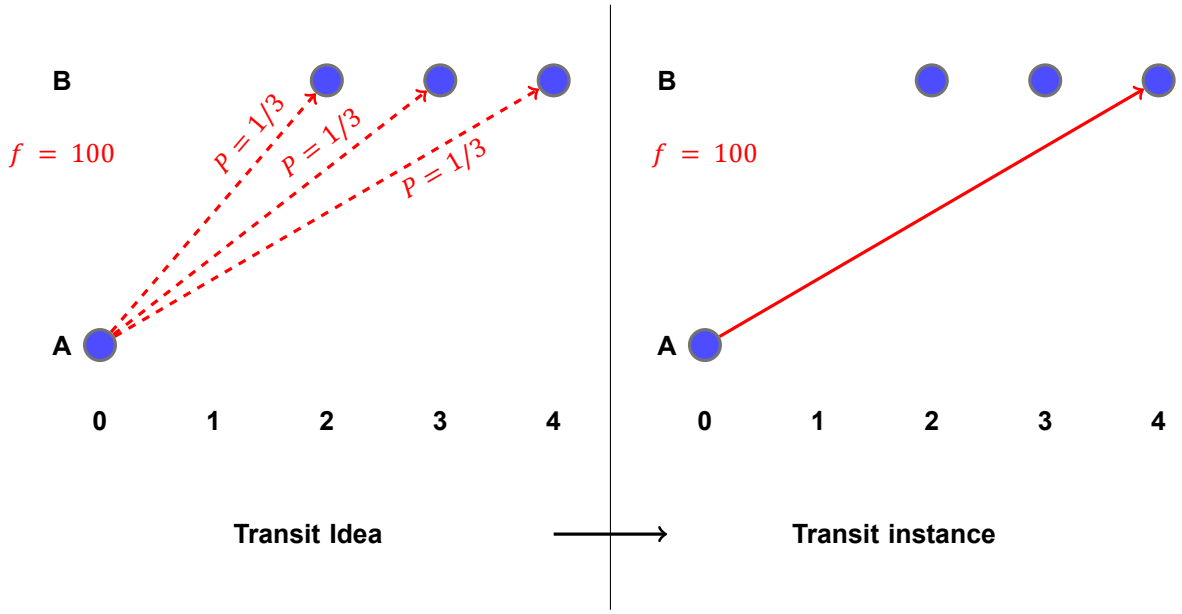


Figure 5.2: On the left, a transit Idea of which only the travel time is uncertain; on the right, one of the three possible instances of the Idea.

5.1.2. Request Ideas and request instances

By the same philosophy of transit Ideas, a *request Idea* $\mathfrak{R}$ is defined as a sextuple $\mathfrak{R} = (\mathfrak{R}_{OD}, \mathfrak{R}_{RT}, \mathfrak{R}_{DT}, \mathfrak{R}_A, \mathfrak{R}_P, \mathfrak{R}_{\sim})$ where:

- $\mathfrak{R}_{OD}$ is a random variable representing the origin-destination pair of the request;
- $\mathfrak{R}_{RT}$ is a random variable representing the release time of the request, so the from which the containers of the request can be picked up;
- $\mathfrak{R}_{DT}$ is a random variable representing the due time of the request;
- $\mathfrak{R}_A$ is a random variable representing the amount of containers in the request;
- $\mathfrak{R}_P$ is a random variable representing the penalty function of lateness of the request, as also described in Section 4.1.3;

- $\mathfrak{R}_\sim$ is a joint probability distribution on the random variables. If the random variables are independent, this probability distribution equals the product of marginal distributions.

Again, drawing these random variables creates an instance of the request Idea, which will be called a *request instance*.

5.1.3. Omnifutures

As transit Ideas generate all possible transit instances, and request Ideas generate all possible request instances, putting them together creates an object that generates all possible futures. Therefore, an *omnifuture* Ω is defined as a pair $\Omega = (\underline{\mathfrak{T}}, \underline{\mathfrak{R}}, \Omega_\sim^S)$ where:

- $\underline{\mathfrak{T}}$ is a list of transit Ideas;
- $\underline{\mathfrak{R}}$ is a list of request Ideas;
- $\Omega_\sim$ is a joint probability distribution on what instances are drawn from the Ideas. Ideas can be dependent: for example, if two transit Ideas correspond to the first and second transits of one vehicle, then if the former instantiates with large travel time, the latter may be more likely to instantiate with a late departure time. If all Ideas are independent, $\Omega_\sim$ equals the product of marginal distributions of the separate Ideas. Otherwise, it should be ensured that the final elements of the transit Ideas and request Ideas are correct marginal distributions of $\Omega_\sim$.

Using $\Omega_\sim$, one can draw values for all transit Ideas and request Ideas. The result, a *future*, can be interpreted as an instance of deterministic container-to-mode assignment and solved using the techniques from Chapter 4.

The example given in Figure 5.1 displays an omnifuture consisting of two transit Ideas and a request Idea. For one of the transit Ideas, there is uncertainty in its travel time and it could instantiate into one of three different values. The other transit Idea has no uncertainties: it generates only one fixed instance. The same goes for the request Idea. The three Ideas are independent and the omnifuture generates three futures with equal probability. In a sense, an omnifuture can be seen as a 'future Idea'.

The implicit assumption in this definition is that the probability distribution on possible futures is independent on the 'current' state. This conflicts directly with the synchromodal principal of adjusting decisions to updated information. This will be resolved in later sections by always basing an omnifuture on the 'current' state: when moving into a new state, a new omnifuture is defined and assumed.

5.1.4. Finite window methods and rolling window methods

Two important uncertainties have not been addressed in the previous sections:

1. In the definition of an omnifuture, the amount of transit Ideas and request Ideas was taken as fixed, while it may be difficult to predict how many requests will be placed;
2. None of the definitions take into account that new transit Ideas or request Ideas could enter the system during the 'resolution of the future'.

Of course, these two issues are closely related. Both are resolved by designing *rolling window methods*: methods that keep adding new Ideas as they enter the system and make decisions according to the presence or anticipation of new request Ideas and transit Ideas. If a method assumes that no new Ideas will enter the system at any point, this will be referred to as a *finite window method*.

5.1.5. Locked futures and future trees

As a final distinction, a model-maker must decide whether or not the future depends on the decisions made by the decision-maker: in other words, whether or not the omnifuture is independent of decisions. For example, some barge operators are known to wait with departure

until their barge is at least 70% filled with cargo, in order to make their trip worthwhile: in this case, their uncertain departure time is dependent on the decision-maker's container-to-mode assignment.

This chapter assumes *locked futures*, meaning that the instantiation of Ideas does not depend on choices in container-to-mode assignment. In other words: *there is only one true future, but the decision-maker does not yet know which one that is*. If one would not assume locked futures, one would have to make decisions regarding some form of *future tree*: a tree of possible futures that branches on decisions. Though the latter is more general and arguably more representative of real world practice, it is not investigated in this thesis due to expected issues of computational and conceptual complexity.

5.1.6. Demifutures

The following two assumptions could be made to keep the problem of stochastic container-to-mode assignment manageable:

- The window is finite. At the start of the problem, all transit Ideas and request Ideas are given. No new ones will enter the system in the given time window. See also Section 5.1.4.
- The future is locked. How the Ideas will instantiate is independent of the choices in container-to-mode assignment. See also Section 5.1.5.

Under these assumptions, the following situation must occur. At the start of the time window, some omnifuture Ω_0 is given. Because the future is locked, there is some 'true future' Φ among the futures generated by Ω_0, but it is probably not yet known which one this is. If Φ were known, the optimal container-to-mode assignment could be easily found using the techniques from Chapter 4. Instead, one has to make decisions at time step 0 against an uncertain future. At time step 1, more may be known: for example, transits may turn out to depart in this time step, making their departure times and destinations known, or requests may have been released, making their release time and other features known. Now, decisions have to be made at time step 1 against some induced omnifuture Ω_1.

Because the window is finite and all time-related features can be assumed to be finite as well, eventually, all transits and requests will have taken place, making all their features known. Φ will have revealed itself to be the true future, and hopefully, good decisions will have been made.

Under the two assumptions, a *demifuture* Ω_t^Φ can be defined as an omnifuture that was obtained in the following way: by taking some initial omnifuture Ω_0, and knowing some true future Φ, revealing everything that can be known of Φ at time t and limiting Ω_0 accordingly. If $t \to \infty$, Ω_t can only generate Φ as a possible future. See also Figure 5.3.

In practice, of course, Φ is not known. The main purpose of demifutures is to simulate the synchromodal principal of more becoming known over time and to be able to design and test decision processes using this. Though equivalent definitions and properties of demifutures may well be derived without these two assumptions, these are not necessary for this thesis.

5.2. Solving to optimality

In this section, multistage stochastic programming and Markov Decision Processes will be discussed as methods to solve Problem 2 to optimality. However, they will also be shown to be computationally far too heavy to be of practical use.

5.2.1. Two-stage stochastic programming

Multistage stochastic programming relies on a technique found in two-stage stochastic programming, which will be introduced here. *Stochastic programs* resemble linear programs,

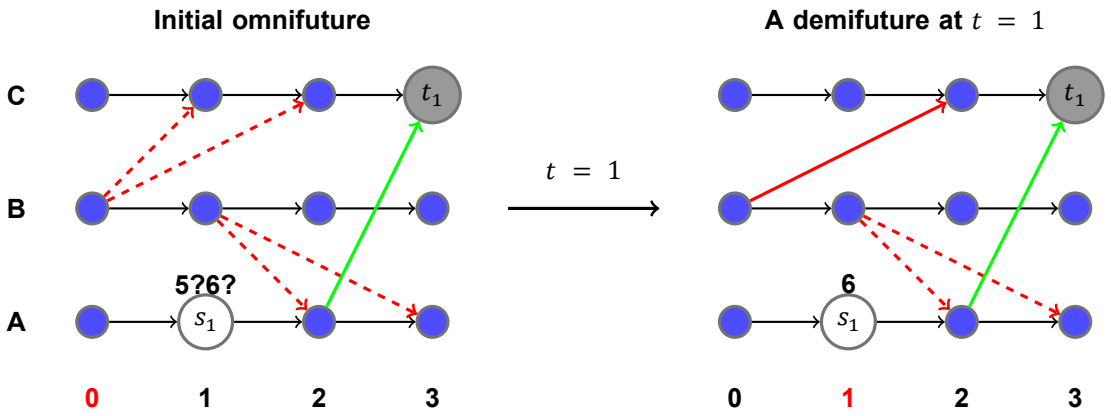


Figure 5.3: On the left, an omnifuture with three transit Ideas, of which only two have an uncertainty (travel time), and a request Idea with only uncertain volume. This omnifuture generates 8 possible futures. At $t = 1$, the request has been released and it turned out to have a volume of 6 containers. The transit that departed at $t = 0$ has not yet been finished, so it cannot have duration 1, so it must have duration 2, the only other option. In the demifuture at $t = 1$, only one Idea has uncertainty left. This demifuture generates 2 possible futures. Every demifuture after this will only generate Φ, the true future.

but with an expected value in the objective function based on randomness in the parameters that define the decision space and/or objective function.

If the stochastic component of the objective function can only take a finite amount of values against known probabilities, and these values are linear in decision variables, then the expected value can be substituted by these values times their probabilities to create one large linear program, which is called its *deterministic equivalent*. An example will be given later in this section.

In *two-stage stochastic optimisation*, some decisions have to be made before the random variables instantiate and some have to be made after. In the second stage, there are no more random variables and the choice becomes much simpler; the difficulty lies in the first stage, where decisions have to be made that influence the decision space of the second stage based on uncertainties. One then has to balance the direct cost of decisions in the first stage and the implied expected cost of decisions in the second stage.

A classical example of this is the newsvendor problem [23]: one wants to sell as many newspapers as possible in stage 2, but has to order stock in stage 1 without knowing what the demand will be in stage 2. Unsold newspapers become worthless at the end of the day. Suppose, for the sake on an example, that newspapers can be ordered at cost 2 and sold at cost 4 and that if D is the demand,

$$P(D = 1) = 0.1, \quad P(D = 2) = 0.2, \quad P(D = 3) = 0.3, \quad P(D = 4) = 0.4$$

If x is the amount of newspapers to order in stage 1, y is the amount to sell in stage 2 and y_i is the amount to sell knowing that $D = i$, problem 5.1 is a stochastic program that describes this and problem 5.2 is its deterministic equivalent.

$$\begin{aligned} \max_{x,y} \quad & -2x + 4\ \mathbb{E}[y] \\ s.t. \quad & y \leq x \\ & y \leq D \\ & x, y \in \mathbb{N} \end{aligned} \tag{5.1}$$

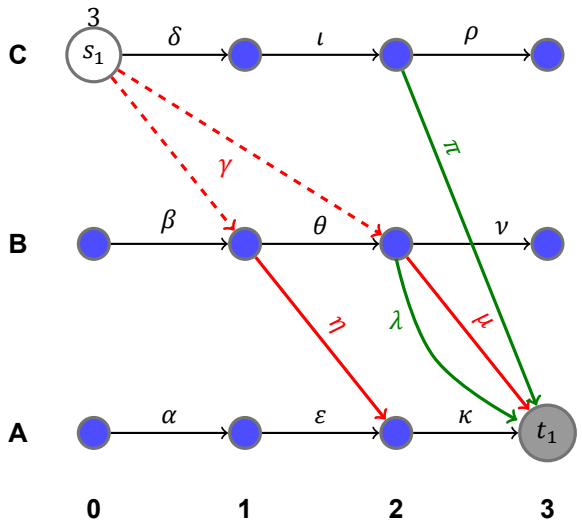
$$\begin{aligned} \max_{x,y_1,y_2,y_3,y_4} \quad & -2x + 4(0.1y_1 + 0.2y_2 + 0.3y_3 + 0.4y_4) && (5.2) \\ s.t. \quad & y_i \leq x && i = 1,2,3,4 \\ & y_i \leq i && i = 1,2,3,4 \\ & x, y_1, y_2, y_3, y_4 \in \mathbb{N} \end{aligned}$$

The deterministic equivalent can be solved using an ILP solver, giving as optimal solution $x = 3$, $y_1 = 1$, $y_2 = 2$, $y_3 = 3$, $y_4 = 3$. This result can be interpreted as a 'two stage policy', stating how many newspapers to order and how many to sell given a certain outcome of the demand.

5.2.2. Multistage stochastic programming: an illustrative example

Multistage stochastic optimisation is a generalisation of two-stage stochastic optimisation that involves more than two stages. This is suitable for synchromodal transport, because decisions have to be made at each time step that influence later decisions, but there is still uncertainty surrounding these later decisions. Multistage stochastic optimisation problems can be solved using backward dynamic programming, where each recursion step involves a two-stage stochastic program. This technique will be referred to as *multistage stochastic programming*. An applied example is given here. Note that an example was formulated where common sense supports the solution, so as to aid the reader, while in many actual instances, the optimal policy may not be easy to see without this technique.

Consider the instance of Problem 2 described in Figure 5.4:



Arc	Cost	Capacity	Duration
γ	1	100	X
η	2	2	1
μ	Y	Y	1
λ	4	∞	1
π	5	∞	1
rest	0	∞	1

$P(X = 1) = 0.6, P(X = 2) = 0.4$
$P(Y = 2) = 0.7, P(Y = 4) = 0.3$

Figure 5.4: A toy example of stochastic container-to-mode assignment, or Problem 2.

Three containers have to be sent from C to A. They have a deadline at $t = 3$, which is another way of saying they have a due date at $t = 3$ with arbitrarily large lateness penalty. One transit Idea departs at $t = 0$ from C to B, but its duration X is uncertain: it has probability 0.6 of being 1 and probability 0.4 of being 2. Another transit Idea, departing from B to A at $t = 2$, has uncertain price and capacity: with probability 0.7, they are both 2, and with probability 0.3, they are both 4. All other features are fixed. Admittedly, it is not common in practice to see that price and capacity are equal and random, but it is an allowed model within the general scope of this thesis and serves for an example with manageable computation. Note that the transits corresponding to arcs λ and π represent the option of trucking at the last moment under a double matrix infinite resource model.

In this toy example, one can manually deduce that the optimal policy is to send all three containers over arc γ, whichever its end point may be, then preferring η over μ over λ. If the

end point of γ is $s_{B,1}$, two containers can be sent over η, which is always at least as cheap as sending them over λ or μ. The third container then goes to $s_{B,2}$ by waiting. If the end point of γ is $s_{B,2}$, all three containers show up there. However many containers are at $s_{B,2}$, as much as possible are sent over μ, as sending containers over μ is at least as cheap as sending them over λ. A container sent via this policy contributes at most 5 to the cost, which is always at least as cheap as sending it over π instead.

Now, it will be shown how the same result can be derived by solving a multistage stochastic problem by means of backward dynamic programming. This requires a definition of states. Note that in this example, there are four possible futures, denoted as follows:

$$\Phi_1 : (X,Y) = (1,2); \quad \Phi_2 : (X,Y) = (1,4); \quad \Phi_3 : (X,Y) = (2,2); \quad \Phi_4 : (X,Y) = (2,4)$$

Further note that at any of the discrete-time moments, a container can be one of the six following 'places':

$$(unreleased, A, B, C, on\ board\ of\ \gamma, finished)$$

If the source node in this example would have been at $s_{C,2}$ instead of $s_{C,0}$, the containers would have been at the location 'unreleased' at time $t = 0$ and $t = 1$ and at location C at $t = 3$. In other words, containers are at location 'unreleased' before their release time; similarly, they are moved to the location 'finished' as soon as they reach their due location, A. Further note that containers can only be at the location 'on board of γ' at $t = 1$, if $X = 2$.

Using all this, a system state can be fully described by a 6-dimensional vector v indicating where the containers currently are, an integer τ that represents the current time step and a demifuture Ω_τ^ϕ that indicates both what has become certain so far and what uncertainties there still are. As such, let $S_\tau^\phi(v_1, v_2, v_3, v_4, v_5, v_6)$ denote the state that there are v_i containers at location i, $i = 1,2,3,4,5,6$, that the current time is $t = \tau$, $\tau \in \{0,1,2,3\}$ and that $\phi \in \{\Phi_1, \Phi_2, \Phi_3, \Phi_4\}$ is the true future, so the uncertainties are defined by the demifuture Ω_τ^ϕ.

Finally, denote $X_\tau^\phi(v_1, v_2, v_3, v_4, v_5, v_6)$ the multistage policy that, assuming that the system is in state $S_\tau^\phi(v_1, v_2, v_3, v_4, v_5, v_6)$, has minimal expected further cost $V_\tau^\phi(v_1, v_2, v_3, v_4, v_5, v_6)$ to get to the finishing state, where all containers are at location 'finished'.

In this example, there are no more uncertainties at $t = 2$, regardless of which future is the true future. As such, $X_2^\phi(v)$ can be calculated easily for any container distribution v and future ϕ: this is a matter of solving deterministic container-to-mode assignment, using the current container locations as source nodes. In futures Φ_3 and Φ_4, the containers on board of γ arrive at node $s_{B,2}$ and need to be absorbed into the system. One could thus solve problem 5.4 for futures Φ_3 and Φ_4 and problem 5.3 for futures Φ_1 and Φ_2 where Y is always known.

$$\begin{aligned} V_2^\phi(v_1, v_2, v_3, v_4, v_5, v_6) = \min \quad & \textstyle\sum_{i=\kappa,\lambda,\mu,\nu,\pi,\rho}^{n} c_i x_i & & \\ s.t. \quad & x_\kappa & = v_2 & \\ & x_\lambda + x_\mu + x_\nu & = v_3 & \\ & x_\pi + x_\rho & = v_4 & \\ & x_\kappa + x_\lambda + x_\mu + x_\pi & = 3 & \\ & x_\mu & \leq Y & \\ & x_\kappa, x_\lambda, x_\mu, x_\nu, x_\pi, x_\rho & \in \mathbb{N} & \end{aligned} \tag{5.3}$$

$$
\begin{aligned}
V_2^{\phi}(v_1, v_2, v_3, v_4, v_5, v_6) = \min \quad & \textstyle\sum_{i=\kappa,\lambda,\mu,\nu,\pi,\rho}^{n} c_i x_i && (5.4)\\
s.t. \quad & x_\kappa && = v_2\\
& x_\lambda + x_\mu + x_\nu && = v_3 + v_5\\
& x_\pi + x_\rho && = v_4\\
& x_\kappa + x_\lambda + x_\mu + x_\pi && = 3\\
& x_\mu && \leq Y\\
& x_\kappa, x_\lambda, x_\mu, x_\nu, x_\pi, x_\rho && \in \mathbb{N}
\end{aligned}
$$

Alternatively, it is easy to manually verify that all containers at C are transported over π, at most two containers at B are transported over μ and a potential third container over λ and nothing needs to be done with containers at A, so

$$V_2^{\Phi_1}(v_1, v_2, v_3, v_4, v_5, v_6) = V_2^{\Phi_2}(v_1, v_2, v_3, v_4, v_5, v_6) = 5v_4 + Y \cdot \min\{v_3, 2\} + 4 \cdot \max\{v_3 - 2, 0\}$$

$$V_2^{\Phi_3}(v_1, v_2, v_3, v_4, v_5, v_6) = V_2^{\Phi_4}(v_1, v_2, v_3, v_4, v_5, v_6) = 5v_4 + Y \cdot \min\{v_3 + v_5, 2\} + 4 \cdot \max\{v_3 + v_5 - 2, 0\}$$

Note that, at $t = 2$, no containers can be at the locations 'unreleased' or 'finished'. This coincides with how the values v_1 and v_6 have no meaning in problems 5.3 and 5.4. The allowed states are only those where the three containers are divided as integers over locations A, B and C, and 'on board of γ' which can be done in $4 + 12 + 4$ ways. Therefore, using the simple computation above for all 20 allowed container distributions and all 4 futures, all $20 \cdot 4$ relevant values $V_2^{\phi}(0, v_2, v_3, v_4, v_5, 0)$ can be computed.

Next, the decision at $t = 1$ is examined. Herein lies a two-stage stochastic problem: the immediate costs made at $t = 1$ must be balanced against the expected costs $V_2^{\phi}(v)$ induced for $t = 2$. It is not yet known at $t = 1$, for example, if arc μ will have capacity and price 2 or 4. Note also that at $t = 1$, some containers might still be on board of γ if it turned out that $X = 2$. Therefore, in futures Φ_1 and Φ_2, the decision problem is given in problem 5.5 and the slightly different decision problem for Φ_3, Φ_4 is given in problem 5.6.

$$
\begin{aligned}
V_1^{\phi}(v_1, v_2, v_3, v_4, v_5, v_6) = \min \quad & \textstyle\sum_{i=\varepsilon,\eta,\theta,\iota}^{n} c_i x_i \quad + \sum_{j=1}^{4} P(\Phi = \Phi_j | \Omega_1^{\phi}) V_2^{\Phi_j}(v_1', v_2', v_3', v_4', v_5', v_6') && (5.5)\\
s.t. \quad & x_\varepsilon && = v_2\\
& x_\eta + x_\theta && = v_3 + v_5\\
& x_\iota && = v_4\\
& x_\eta && \leq 2\\
& v_1' && = 0\\
& v_2' && = x_\varepsilon + x_\eta\\
& v_3' && = x_\theta\\
& v_4' && = x_\iota\\
& v_5' && = 0\\
& v_6' && = 0\\
& x_i, v_k' && \in \mathbb{N} \quad i = \varepsilon, \eta, \theta, \iota, \quad k = 1, \ldots, 6
\end{aligned}
$$

$$V_1^{\phi}(v_1, v_2, v_3, v_4, v_5, v_6) = \min \quad \sum_{i=\varepsilon,\eta,\theta,\iota}^{n} c_i x_i \quad + \sum_{j=1}^{4} P(\Phi = \Phi_j | \Omega_1^{\phi}) V_2^{\Phi_j}(v_1', v_2', v_3', v_4', v_5', v_6') \quad (5.6)$$

$$\begin{aligned} s.t. \quad x_\varepsilon &= v_2 \\ x_\eta + x_\theta &= v_3 \\ x_\iota &= v_4 \\ x_\eta &\leq 2 \\ v_1' &= 0 \\ v_2' &= x_\varepsilon + x_\eta \\ v_3' &= x_\theta \\ v_4' &= x_\iota \\ v_5' &= v_5 \\ v_6' &= 0 \\ x_i, v_k' \in \mathbb{N} \quad & i = \varepsilon, \eta, \theta, \iota, \quad k = 1, \dots, 6 \end{aligned}$$

There are several things to note about these two problems:

- When computing $V_1^{\Phi_1}(v)$, so when assuming Φ_1 will eventually turn out to be the true future and so at $t = 1$ it has become clear that $X = 1$, every future where $X = 2$ becomes impossible. This gives conditional probabilities $P(\Phi = \Phi_3 | \Omega_1^{\Phi_1}) = P(\Phi = \Phi_4 | \Omega_1^{\Phi_1}) = 0$. The outcome of Y is independent of X, so this gives the other conditional probabilities $P(\Phi = \Phi_1 | \Omega_1^{\Phi_1}) = P(Y = 2) = 0.7$, $P(\Phi = \Phi_2 | \Omega_1^{\Phi_1}) = P(Y = 4) = 0.3$. All other conditional probabilities can be computed analogously.

- $V_2^{\Phi_j}(v)$, in its current form, is not linear in its arguments, making the programs non-linear. Because integral arguments are expected and each argument must be at least 0 and the sum must equal 3, the amount of possible inputs v are finite. Therefore, the program can be made linear again by adding an indicator variable for each allowed input v using techniques for logic operations in linear programming [18]. Though this makes the problems linear again, the amount of added variables and constraints can be exponential in the amount of containers and locations.

Using these two observations, it is possible to find all values $V_1^{\phi}(v)$ through linear programming. Instead, it is manually observed here that getting containers from B to A is cheapest over arc η if possible, and this is the only arc that gives direct costs in this stage, so

$$\begin{aligned} V_1^{\Phi_1}(0, v_2, v_3, v_4, v_5, 0) = \quad & 2 \cdot \min\{v_3 + v_5, 2\} + 0.7 V_2^{\Phi_1}(0, v_2 + \min\{v_3 + v_5, 2\}, \max\{v_3 + v_5 - 2, 0\}, v_4, 0, 0) \\ & + 0.3 V_2^{\Phi_2}(0, v_2 + \min\{v_3 + v_5, 2\}, \max\{v_3 + v_5 - 2, 0\}, v_4, 0, 0) \\ V_1^{\Phi_2}(0, v_2, v_3, v_4, v_5, 0) = \quad & 2 \cdot \min\{v_3 + v_5, 2\} + 0.7 V_2^{\Phi_1}(0, v_2 + \min\{v_3 + v_5, 2\}, \max\{v_3 + v_5 - 2, 0\}, v_4, 0, 0) \\ & + 0.3 V_2^{\Phi_2}(0, v_2 + \min\{v_3 + v_5, 2\}, \max\{v_3 + v_5 - 2, 0\}, v_4, 0, 0) \\ V_1^{\Phi_3}(0, v_2, v_3, v_4, v_5, 0) = \quad & 2 \cdot \min\{v_3, 2\} + 0.7 V_2^{\Phi_3}(0, v_2 + \min\{v_3, 2\}, \max\{v_3 - 2, 0\}, v_4, v_5, 0) \\ & + 0.3 V_2^{\Phi_4}(0, v_2 + \min\{v_3, 2\}, \max\{v_3 - 2, 0\}, v_4, v_5, 0) \\ V_1^{\Phi_4}(0, v_2, v_3, v_4, v_5, 0) = \quad & 2 \cdot \min\{v_3, 2\} + 0.7 V_2^{\Phi_3}(0, v_2 + \min\{v_3, 2\}, \max\{v_3 - 2, 0\}, v_4, v_5, 0) \\ & + 0.3 V_2^{\Phi_4}(0, v_2 + \min\{v_3, 2\}, \max\{v_3 - 2, 0\}, v_4, v_5, 0) \end{aligned}$$

This again yields 80 values $V_1^{\phi}(0, v_2, v_3, v_4, v_5, 0)$. Finally, the first decisions at $t = 0$ are determined in problem 5.7, which has only one feasible container distribution: namely, that all 3

containers have been released at location C.

$$V_0^{\phi}(0,0,0,3,0,0) = \min \quad \sum_{i=\alpha,\beta,\gamma,\delta}^{n} c_i x_i \quad + \sum_{j=1}^{4} P(\Phi = \Phi_j | \Omega_1^{\phi}) V_1^{\Phi_j}(v_1', v_2', v_3', v_4', v_5', v_6') \qquad (5.7)$$

$$\begin{aligned} s.t. \qquad x_\alpha &= v_2 = 0 \\ x_\beta &= v_3 = 0 \\ x_\gamma + x_\delta &= v_4 = 3 \\ x_\gamma &\leq 100 \\ v_1' &= 0 \\ v_2' &= x_\alpha \\ v_3' &= x_\beta \\ v_4' &= x_\delta \\ v_5' &= x_\gamma \\ v_6' &= 0 \\ x_i, v_k' \in \mathbb{N} \qquad & i = \varepsilon, \eta, \theta, \iota, \quad k = 1, \dots, 6 \end{aligned}$$

This program boils down to only one decision: how many of the three containers to send over γ, while sending the rest over δ. Therefore, only four values need to be checked:

- $0 \cdot 1 + \sum_{j=1}^{4} P(\Phi = \Phi_j) V_1^{\Phi_j}(0,0,0,3,0,0) = 15$;
- $1 \cdot 1 + \sum_{j=1}^{4} P(\Phi = \Phi_j) V_1^{\Phi_j}(0,0,0,2,1,0) = 13.24$;
- $2 \cdot 1 + \sum_{j=1}^{4} P(\Phi = \Phi_j) V_1^{\Phi_j}(0,0,0,1,2,0) = 11.48$;
- $3 \cdot 1 + \sum_{j=1}^{4} P(\Phi = \Phi_j) V_1^{\Phi_j}(0,0,0,0,3,0) = 10.64$;

The first value reflects the certainty that if all three containers are not sent over γ, they will be sent over π against cost 5. The second value reflects the certain cost of 10 for the two containers that do not travel over γ, plus the expected cost for the final container: 1 to traverse γ, then a further 2 against probability $P(X = 1) + P(X = 2, Y = 2) = 0.88$ or 4 against probability $P(X = 2, Y = 4) = 0.12$, so an expected cost of $1 + 0.88 \cdot 2 + 0.12 \cdot 4 = 3.24$. Similarly, the third value equals $5 + 2 \cdot 3.24$. Finally, the fourth value can be checked with the following computation: the first two containers each have expected cost 3.24, then the third container has cost $1 + 2$ only in Φ_1 and $1 + 4$ otherwise, so the third container has expected cost $0.42 \cdot 3 + 0.58 \cdot 5 = 4.16$ for a grand total expected cost of 10.64. Indeed, sending all containers over γ is the decision to take at $t = 0$ with the smallest expected cost.

5.2.3. Why multistage stochastic programming is not used

The example given in Section 5.2.2 could be generalised to solve stochastic container-to-mode assignment to optimality. Dynamic programming largely halts the growth of complexity in the length of the time window: if the example had a hundred time steps instead of only four, each step would still only require the 80 values of the next time step and produce 80 values before terminating, due to the *property of state* [20].

However, the required amount of values per time step may equal the amount of possible futures times the amount of allowed container distributions. In the example, this was 4 times 20, or 4 times 28 when not manually discarding the locations 'unreleased' and 'finished'. In general, the amount of possible futures may equal the product of the amount of values that each stochastic element can take. Take for example the problem instance sketched in Figure 5.5: there are only three request Ideas and two transit Ideas, one of the requests is already revealed at $t = 0$, the other request Ideas have release time X_1, X_2 respectively that may each take four different random values and an independent random volume Y_1, Y_2 of either 9 or

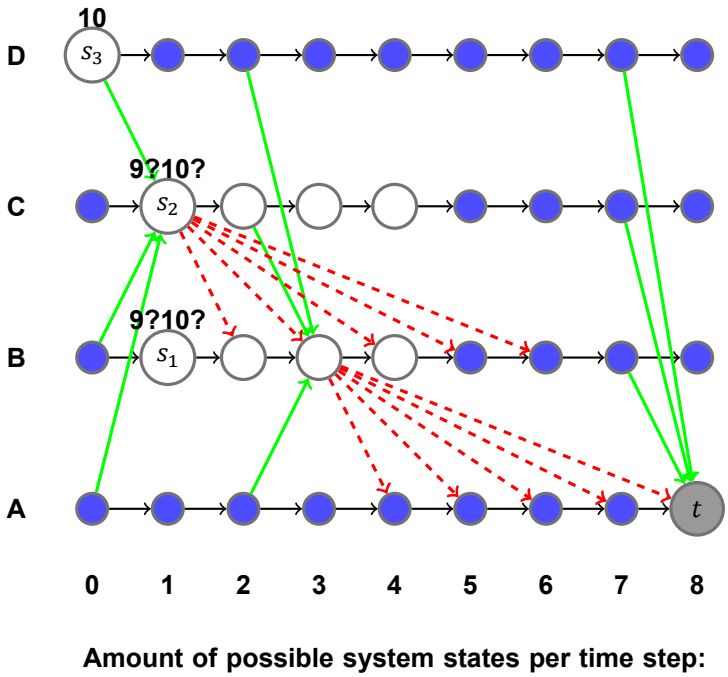


Figure 5.5: An instance of Problem 2 that shows the explosive growth of the state space. There are three request Ideas: one that is already revealed, the other two having 9 or 10 containers that are released at one of four possible times. The amounts are independent of the release time. All containers are due at the same place and time. There are two transit Ideas with uncertain travel time, one departing from $s_{B,3}$, the other from $s_{C,1}$; for the rest, it is possible to take a truck from any location to any other location in one time step to arrive just in time for the transit or the sink node. Though this instance is significantly smaller than 'real life', if the methodology from Section 5.2.2is applied, it would already involve having to keep track of values of over 10^{16} states, as explained in Section 5.2.3.

10 containers, and the transit Ideas have travel time Z_1, Z_2 respectively that may each take five different random values. Then that instance already has $4 \cdot 4 \cdot 2 \cdot 2 \cdot 5 \cdot 5 = 1600$ possible futures. If the instance concerns four real life locations A, B, C and D, so it concerns the 8 locations

$$(unreleased,\ A,\ B,\ C,\ D,\ on\ transit\ 1,\ on\ transit\ 2, finished),$$

and each request consists of 10 containers, then according to Theorem 5 in Appendix A, there are

$$\begin{pmatrix} 10+8-1 \\ 10 \end{pmatrix} = 19448$$

ways of distributing the 10 containers of the revealed request over the 8 locations, so if the others have 10 containers as well, $19448^3 = 7355714043392$ distributions of all containers over the locations. Multiplying this, the problem would have over $1.176 \cdot 10^{16}$ or 11 quadrillion allowed states it could be in at each time step. In general, unless some smart reduction is found, multistage stochastic programming could require keeping track of

$$\mathcal{O}\left(\prod_{stochastic\ elements} (amount\ of\ possible\ values) \prod_{request\ k} \begin{pmatrix} c_k + l - 1 \\ c_k \end{pmatrix} \right)$$

states, where c_k is the largest possible amount of containers in request k.

At each time step, for each state, a two-stage stochastic problem would have to be solved which can be made into an integer linear program at the cost of adding at least one variable and constraint for each state: thus, the growth of these integer linear programs is also exponential.

Long story short: unless some smart reduction is performed, the amount of states to keep track of grows uncontrollably, from 112 in Figure 5.4 to 11769142469427200 in Figure 5.5. For each of these states, an integer linear program with at least as many variables would have to be solved at each time step. This explosive growth of required space and time makes multistage stochastic programming unsuited for use beyond tiny instances.

5.2.4. Markov Decision Processes

The second way discussed here to solve Problem 2 is based on writing the decision process as a *Markov Decision Process*. This alternative to multistage stochastic programming appears to be more commonly discussed [35, 45]. In a Markov Decision Process (MDP), a number of actions can be taken in each state that yield a direct reward and move the system to another state with a certain probability. For examples, the reader is referred to White's survey [60].

In the context of this problem, one can again define a state based on a container distribution, a point in time and an omnifuture specifying the uncertainties that are still left. The allowed actions are moving from one container distribution v_1 to another v_2, which always succeeds, given that the modalities to do so are present. If containers are brought to their due location, they are immediately bounced to the 'finished' location. Due to the assumption of locked futures, the probability that an action will move the system from state A with demifuture Ω_A to state B with demifuture Ω_B is exactly the conditional probability that state B has demifuture Ω_B given that its previous state has demifuture Ω_A. Rewards are exclusively non-positive: they correspond to the arc weights used in Chapter 4 for transportation costs and lateness penalties. See also Figure 5.6.

Being able to write a decision process as a MDP depends on whether or not the decision process has the *Markov property* [47]: that probabilities and decisions depend only on the current state, not on previous ones. This property is closely related to the property of state mentioned in Section 5.2.3 and indeed, the methods appear related. These are at least three ways in which using a MDP differs from using multistage stochastic programming:

- MDP's are often defined with a *discount factor* $\gamma \in [0,1]$ that indicates how important immediate savings are when compared to expected future savings;
- Because of this discount factor, and because of the absence of backwards dynamic programming, MDP's lend themselves more naturally to rolling time window methods;
- MDP's have an action space with which to move around states, where multistage stochastic programming suggest moving from one state to another by means of two-stage stochastic programming. As a consequence of Theorem 5 in Appendix A, the action space may have size $\mathcal{O}(\prod_{commodity\ k} \begin{pmatrix} c_k + l - 1 \\ c_k \end{pmatrix})$, where c_k is the maximal amount of containers in request k and l is the number of locations.

It is noted that such a discount factor, within the context of synchromodal freight transport, is nice but not necessary. It was decided to limit this research to finite window methods, making the second difference also of lesser importance. The most important reason MDP's would be chosen over multistage stochastic programming, would be if MDP's could be solved in significantly less time.

Several solution methods exist for MDP's [13]. However, in the context of freight transportation, it is a common conclusion that the state space and action space are both too large to solve MDP to optimality [35, 45]. Since this method uses almost the same enormous state space that is used in Section 5.2.3 and also an exponentially large action space, the same conclusion is drawn here without numeric evidence. Solving Problem 2 to optimality, using either multistage stochastic programming or MDP's, is only possible in tiny instances. When looking for a method that is fast enough for on-line use, it appears necessary to either smartly reduce the state spaces of these methods, or to use a heuristic or meta-heuristic.

5.3. Single future iteration heuristics

The methods described in the previous section would probably not find solutions fast enough to be feasible in most practical applications. In particular, synchromodal planning depends on regular revaluations of the plannings, making it essential to formulate good plans in a relatively short amount of time. In this section, two basic ways of doing this are described.

5.3.1. Expected future iteration

If a barge has a travel time that is with equal probability 10, 11 or 12 time steps, then one could simplify the thought process by assuming the travel time will be its 'average', 11. Exactly this is the idea of computing an *expected future*: assume that all stochastic elements are independent, then for every numeric stochastic element, compute its rounded expected value and work with that. See also Figure 5.7.

For non-numeric stochastic elements, like release location, one could randomly use any of the values with highest probability instead.

The idea of *Expected Future Iteration* (EFI) is the following:

1. Compute the expected future;

2. Interpret the expected future as an instance of deterministic container-to-mode assignment and solve it using the techniques from Chapter 4;

3. Use this solution to decide how to assign containers in the current time step. Enact these decisions, then go to the next time step;

4. Go back to 1., working with the current container locations and the demifuture of this new time step, that is to say, using all information that has become known up until this new time step.

This process given in more detail in Algorithm 2 and Algorithm 3. Note that Algorithm 3 was designed for testing purposes: it simulates the decision process that, assuming locked futures and finite windows, would take place if 2 were used until all containers have reached their destinations. At the end of this process, the space-time network will be based on some true future: of this true future, the optimal container-to-mode assignment can be found using the techniques from Chapter 4, which then gives a lower bound on how well the decision-process could have performed.

In practice, decisions would be made using only Algorithm 2, which relies less on the assumptions of locked futures and finite windows. In particular, the temporary assumption made at each time step that the time window is finite is not expected to be a large problem, because when new requests do come in, they can simply be added to the next iteration. It may be interesting, however, to see if transportation should be 'pulled' closer to the current time, so as to reserve capacity near the end of the time window for new requests: this, and other questions concerning rolling time windows, are proposed as future research.

Data: Omnifuture Ω, double matrix infinite resource matrices M^1, M^2
Result: Demand-to-resource assignment for current time step
for *numeric stochastic variable* **do**
  compute its expected value;
end
for *non-numeric stochastic variable* **do**
  obtain its mode, breaking ties randomly;
end
Create instance I of deterministic container-to-mode assignment by replacing all Ideas in Ω by instances with parameters equal to expected values and modes;
Obtain an optimal freight plan by solving I with the techniques from Chapter 4, given M^1, M^2;
From this solution, return all arcs that emanate from a node at time $t = 0$ and translate this back to a container-to-mode assignment at $t = 0$. Return this assignment, STOP.;

Algorithm 2: Determining container-to-mode assignment for the current time step using Expected Future Iteration.

Data: Omnifuture Ω, true future Φ, double matrix infinite resource matrices M^1, M^2
Result: Synchromodal decisions over the entire time window
Initialise $t = 0$;
Initialise all containers at location 'unreleased';
while *there are still containers not yet at 'finished';*
do
  Obtain current demifuture Ω_t^Φ;
  for *request in request Ideas* **do**
    if *request release time equals* t *according to* Φ **then**
      Move all containers of request from 'unreleased' to release location according to Φ;
    end
    if *request released according to* Φ **then**
      for *chunk of containers of request* **do**
        if *chunk at due location of request according to* Φ **then**
          Move chunk to location 'finished';
        end
        if *chunk not at 'finished'* **then**
          Interpret chunk as request instance with source node $s_{v,t}$, where v is the location of chunk and t the current time step;
        end
      end
    end
  end
  Create instance I of stochastic container-to-mode assignment with the transit Ideas of Ω_t^Φ as transit Ideas and as request Ideas the request Ideas that are not yet released according to Φ and the chunks interpreted as request instances;
  Denote Ω_I the omnifuture based on taking Ω_t^Φ and redefining the request Ideas as described;
  Obtain a container-to-mode assignment for the current time step from Algorithm 2(Ω_I, M^1, M^2);
  Enact this assignment at the current time step;
  Update t := t + 1;
end
STOP;

Algorithm 3: Simulating synchromodal decision-making over the entire time window using Expected Future Iteration. Designed for testing purposes.

Expected Future Iteration has several merits: it is conceptually simple and it will be shown in Section 5.4 to give decent results in little time. It has one glaring downside, however: it fully trusts in the expected future and does not 'optimise its plan B', so if the actual values do not equal the expected values, the result can be arbitrarily bad. See also Figure 5.8. Furthermore, the underlying assumption that all stochastic elements are independent may not always be a realistic assumption, especially when destinations are random. It would be better to somehow use the final elements of the Ideas, the joint probability distributions, but it is harder to define an expected value on these.

5.3.2. Partially pessimistic future iteration

The method described in this section could be viewed as a 'safer' version of Expected Future Iteration. If X is a discrete random variable that may take values in the finite set $\chi \subset \mathbb{R}$, define

$$\sigma_{\leq}(X, \alpha) = \min\{x \in \chi | P(X \leq x) \geq \alpha\}$$

as the *α-upper value of X* and

$$\sigma_{\geq}(X, \alpha) = \max\{x \in \chi | P(X \geq x) \geq \alpha\}$$

as the *α-lower value of X*. For example:

$$P(X = 1) = P(X = 2) = P(X = 3) = P(X = 4) = P(X = 5) = 1/5 \quad \Rightarrow \quad \sigma_{\leq}(X, 70\%) = 4$$

Note that these values are defined for every $\alpha \in [0, 1]$: for example, $\sigma_{\leq}(X, 1) = \max\{x \in \chi\}$.

In general, if travel times turn out to be long, this is bad for minimising costs: it may cause containers to arrive at their destination late or it may cause expected transfers to be missed. In general, it makes the set of feasible modality-paths smaller: every solution that uses a transit with long travel time has an equivalent solution for the case that the travel time was short, namely by taking the same transit and waiting out the difference, but the opposite is not true. Similarly, late request release dates are worse than early ones, but early due dates are worse than late ones.

Following this logic, Table 5.1 states for each potentially random parameter if it is bad for the value to be high or low.

transit departure location: non-numeric	request release location: non-numeric
transit destination: non-numeric	request due location: non-numeric
transit capacity: bad when low	request volume: bad when high
transit departure time: bad when high	request release date: bad when high
transit travel time: bad when high	request due date: bad when low
resource price: bad when high	request lateness penalty function: non-numeric
terminal handling time: bad when high	container-to-mode assignment: non-numeric

Table 5.1: For each potentially random parameter, if it takes numeric values, whether it is generally bad for total transportation costs when the value is high or rather when it is low.

Then, *α-Pessimistic Future Iteration* (αPFI) is exactly the same as Expected Future Iteration, except that instead of taking expected values for numeric random variables, the α-upper bound is taken instead if Table 5.1 states that it is bad for the value to be high and and the α-lower bound is taken if it is bad for the value to be low. See also Figure 5.7.

Note that no simple robustness conclusions can be drawn from using this method: for example, if a modality-path exists in the 70%-pessimistic future, this does not have to mean that the path has a probability of at least 70% of existing, as demonstrated in Figure 5.9. The same figure illustrates how the expected future is not necessarily the same thing as the 50%-Pessimistic Future. Quantifying or correcting this dissonance, by reformulating the method to be based on the probability that a used path exists and using this for robust optimisation,

could be an insightful topic of future research.

Note also that α-Pessimistic Future Iteration retains the property of allowing for arbitrarily bad solutions, which can be seen by modifying Figure 5.8. Still, intuition dictates that the higher the pessimism parameter α is set, the more robust the solution will become, at the cost of discarding cheaper but riskier solutions.

5.4. Numerical results

		Class 1	Class 4
EFI cost over optimal:	mean	3.51%	3.54%
	variance	$28.52\%^2$	$7.76\%^2$
	worst	14.48%	8.56%
time for first step:	mean	$1.04s$	$23.98s$
	variance	$0.02s^2$	$1.64s^2$
	worst	$1.44s$	$26.35s$
70PFI cost over optimal:	mean	4.43%	3.34%
	variance	$38.41\%^2$	$7.57\%^2$
	worst	20.55%	8.67%
time for first step:	mean	$1.09s$	$25.60s$
	variance	$0.02s^2$	$2.45s^2$
	worst	$1.49s$	$27.36s$

Table 5.2: Numerical results for stochastic container-to-mode assignment, or Problem 2. Expected Future Iteration and 70%-Pessimistic Future Iteration are abbreviated to EFI and 70PFI respectively. Class 4 contains instances of a size that was specified as useful within an operational use case.

In this section, the methods of Expected Future Iteration and 70%-Pessimistic Future Iteration are tested on several random instances, under the same set-up as in Section 4.4.

If the true future is known completely at the start, one could solve for that future to attain the perfect assignment. The two tested methods discover this true future gradually, and one can only hope that the decisions they make while doing so are indeed the correct decisions. Understanding this, it is possible to give a tight lower bound on what the objective value could have been, and thus to express the efficiency of a decision process. This is one of the two results produced in this test.

The other result concerns computing time, rather than efficiency. In the tests, the iteration is performed until the end of the time window: however, in operational practice, one would only run the first iteration, seeing as one cannot iterate over a future that is not yet known. The relevant computing time, therefore, is the average computing time for the first iteration only. All results are shown in Table 5.2.

5.5. Discussion

Despite the relative naiveté of Expected Future Iteration and 70%-Pessimistic Future Iteration, the results in Section 5.4 show that both methods, in the testing environment, achieve quite good results in not too much time. More precisely: on problems of class 4, so 'real life size' problems, they found assignments with costs that were $3.34\% - 3.54\%$ over the optimum on average and $8.56\% - 8.67\%$ in the observed worst cases. In practice, so making one decision per hour, the time to take this decision with either method was not observed to exceed thirty seconds.

By design, one would expect 70%-Pessimistic Future Iteration to achieve lower worst case costs than Expected Future Iteration, but higher average costs; surprisingly, one might say that these numerical results argue the opposite. It is unclear why this occurs, but two explanations are suggested:

- The two methods are structurally more different than they seem, which also relates to how Expected Future Iteration does not map exactly to 50%-Pessimistic Future Iteration, as observed earlier: as such, they cannot be compared well on the gradient of risk versus robustness;
- The random generation of instances may not have included enough situations in which the short-sightedness of single future iteration heuristics truly creates problems, not allowing 70%-Pessimistic Future Iteration to 'shine' in this field.

Overall, there are several caveats to be addressed about these methods and results. For one, both methods have the theoretic property that they may yield arbitrarily bad results, although no spectacular excesses of cost were observed. Secondly, though the method of random instance creation may suit some chaotic problems well, it is difficult to promise upfront how efficiently these methods will perform on problems with more structurally defined system dynamics. For example, day and night cycles were not simulated in the environment, while these may cause great delays: if Expected Future Iteration blindly assumes that goods will come in just before a terminal's closing time, the consequences of arriving slightly late are major, and methods that consider such consequences may be more suitable. This second downside may well be solved by using the simulation method of Kooiman [35], though this requires solving many instances of Problem 1 during the decision process, which may add up to a decision process that would currently not be fast enough for on-line use. Thirdly, in practice, probability distributions for stochastic elements may likely be unknown: the developed methods depend greatly on reliable forecasting methods being available.

5.5.1. Added value

Stochastic container-to-mode assignment has already been studied under different forms and names, even in contexts that follow synchromodal paradigms [35, 45, 62]. However, none of the encountered literature assumes the holistic stance of this research, in viewing almost anything as potentially stochastic. Such holism may be essential in moving towards a synchromodal setting that encompasses the different dynamics of different parties. As a consequence, this research has formalised some concepts necessary to build these generalistic foundations, in the form of Ideas and omnifutures. Furthermore, this thesis has verified that reasonable solutions can be found for the stochastic container-to-mode assignment problem fast enough for operational decision-support use.

5.6. Conclusion

This chapter sought to answer the following research sub-question:

3. How can a low cost net-centric container-to-transport assignment be found fast enough for on-line use if new data is still expected to come in?

Stochastic container-to-mode assignment, or 'Problem 2', was defined to be the problem of assigning freight containers optimally to transport modalities with predetermined schedules, in the presence of time constraints and stochastic elements. It required the definition of not a singular request, but a request Idea that generates different instances of the request against different probabilities. Similarly, transits were replaced by transit Ideas and omnifutures were designed as objects that generate all possible futures.

It was illustrated that this problem can be solved to optimality using multistage stochastic programming, but also argued that multistage stochastic programming requires far too much computing time beyond tiny instances. Instead, two decision process heuristics were introduced: Expected Future Iteration, which at each time step assumes that all remaining unknowns will take on their marginalised expected values, and α%-Pessimistic Future Iteration, which instead assumes the unknowns will take on a variant of their 'α-percentile value'.

Random instances of different sizes were generated to compare these two methods. They show that assignments for the current time step can be found for problems of a 'real life

size' in an average 23.98 seconds and 25.60 seconds respectively, even when using a non-commercial MILP solver on a single computer. This makes the method suitable for on-line use. The numerical results suggest that the methods will find assignments with costs that are on average 3.54% and 3.34% above optimal respectively.

To answer the sub-question: using either Expected Future Iteration or 70%-Pessimistic Future Iteration serves as a near optimal decision policy as information becomes known. Both methods run fast enough for on-line problems of 'real life size'.

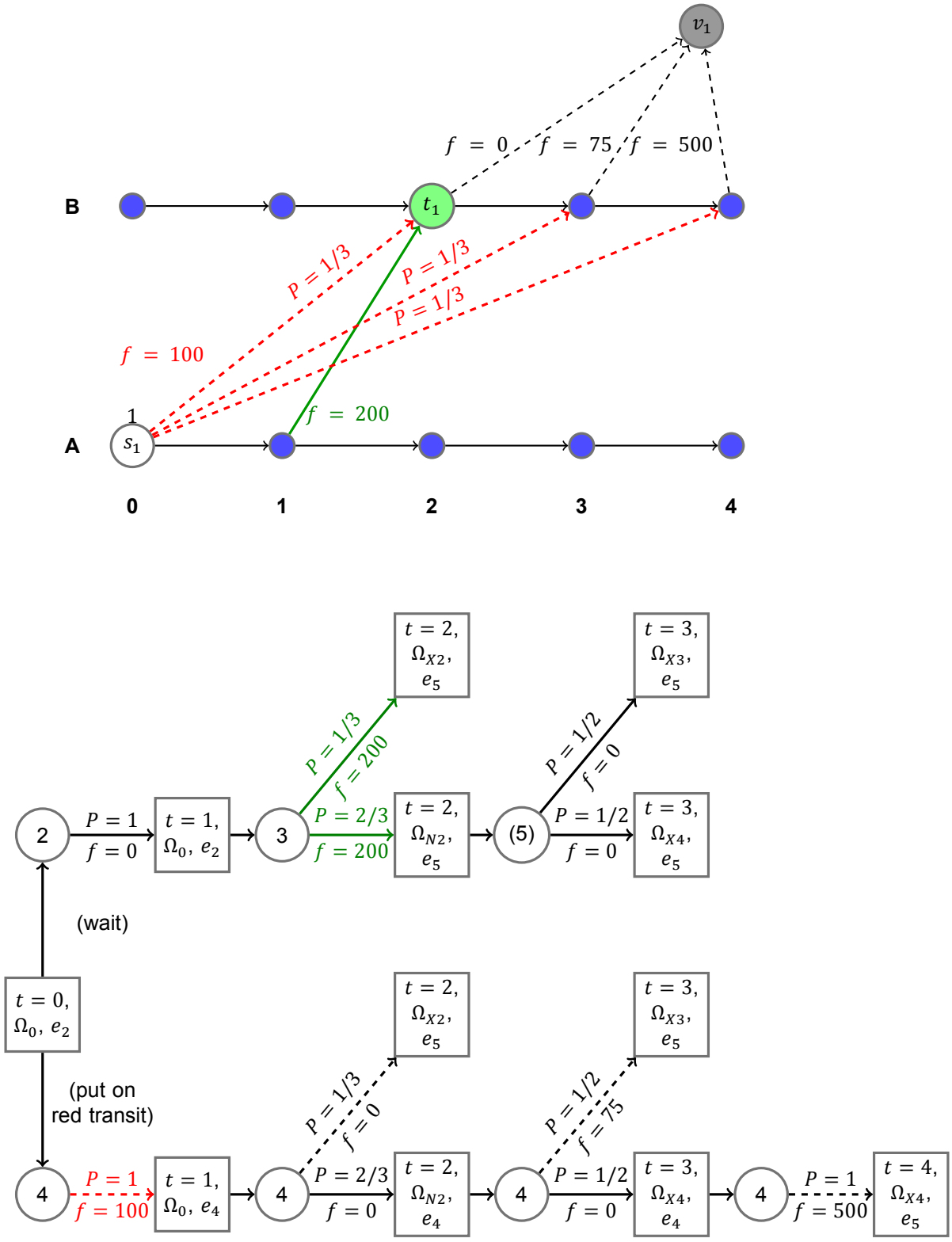


Figure 5.6: Above, Figure 5.1. Below, the same problem interpreted as a Markov Decision Process. There are five locations: (unreleased, A, B, on board of the uncertain transit, finished). There are five different demifutures: the initial omnifuture Ω_0, the demifuture Ω_{X2} in which the transit certainly has travel time 2, the analogously defined Ω_{X3} and Ω_{X4}, and the demifuture Ω_{N2}, in which the transit certainly does not have travel time 2, but the probability that it is 3 or 4 are both 0.5. The states are given as rectangular nodes; a circular node with text i represents moving from the current container distribution to the container distribution e_i, that is to say, moving the container to location i. Note that there is only one decision with multiple feasible answers: whether, at $t = 0$, to commit the container to the uncertain transit or not. The Markov Decision Process ends when container distribution e_5 is attained, so when the containers reaches location 'finished': in the case of state $t = 2; \Omega_{N2}; e_5$, it then turns into a Markov Process, in which Ω_{N2} is replaced by either Ω_{X3} or Ω_{X4}.

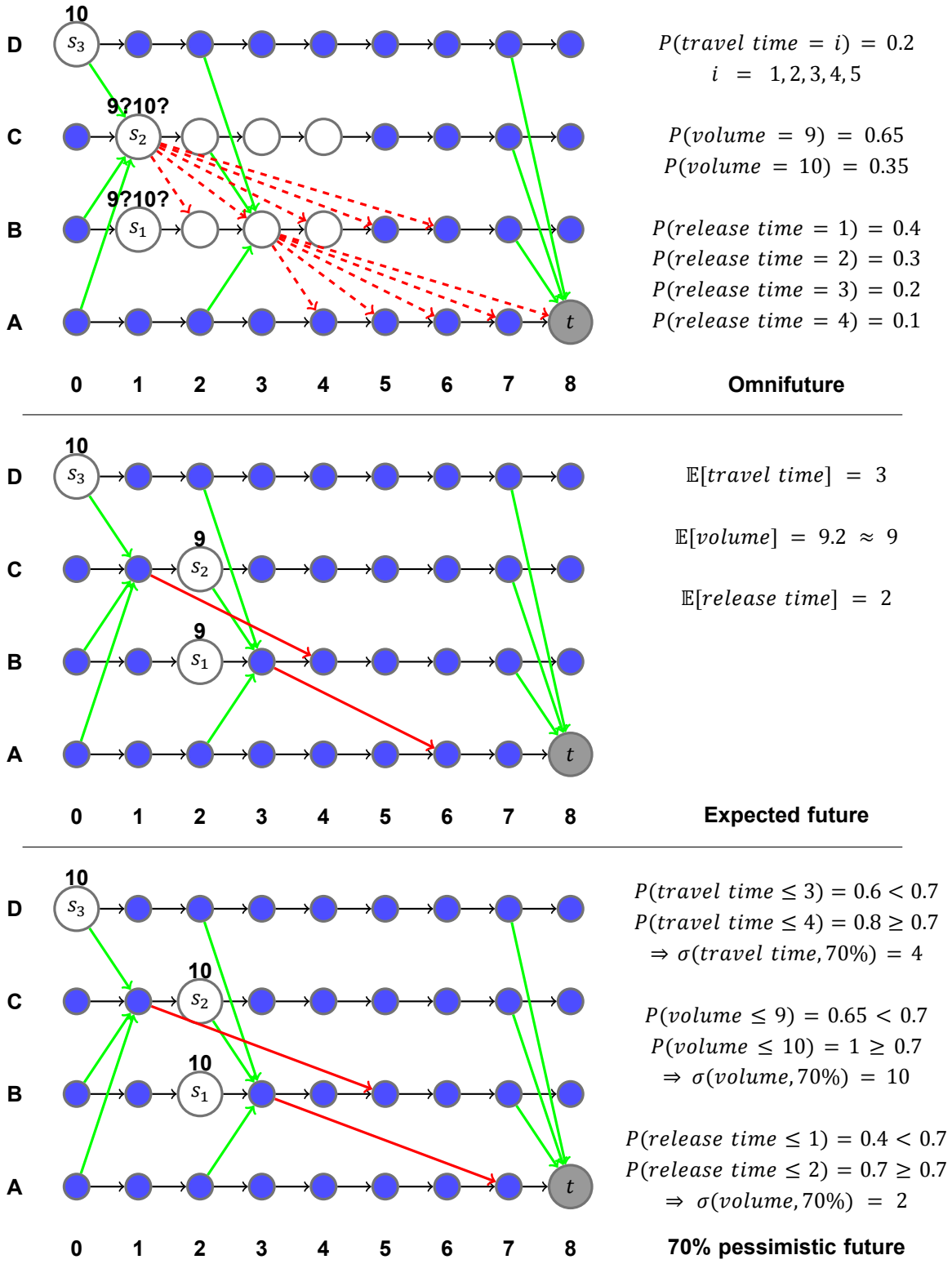


Figure 5.7: At the top, the instance from Figure 5.5. In the middle, its expected future, obtained by taking expected values for all numeric random variables. At the bottom, its 70% pessimistic future, obtained by taking all smallest values for which the probability is at least 70% that this value or a lower value is attained.

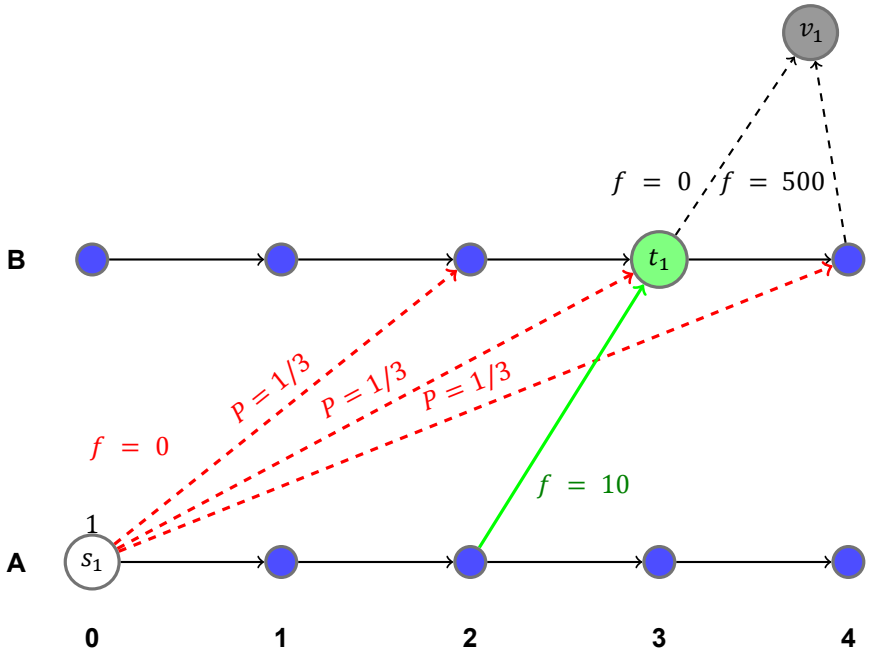


Figure 5.8: An instance that shows how Expected Future Fteration can give arbitrarily bad results. If the transit with uncertain travel time is used, this will have cost 0 if the travel time is 2 or 3, but have the arbitrarily high cost of 500 if the travel time is 4. Expected Future Iteration will choose this option, because it blindly assumes the travel time will be 3, the expected value. However, the expected cost of this option is $500/3$, whereas the expected cost of using the certain transit is only 10.

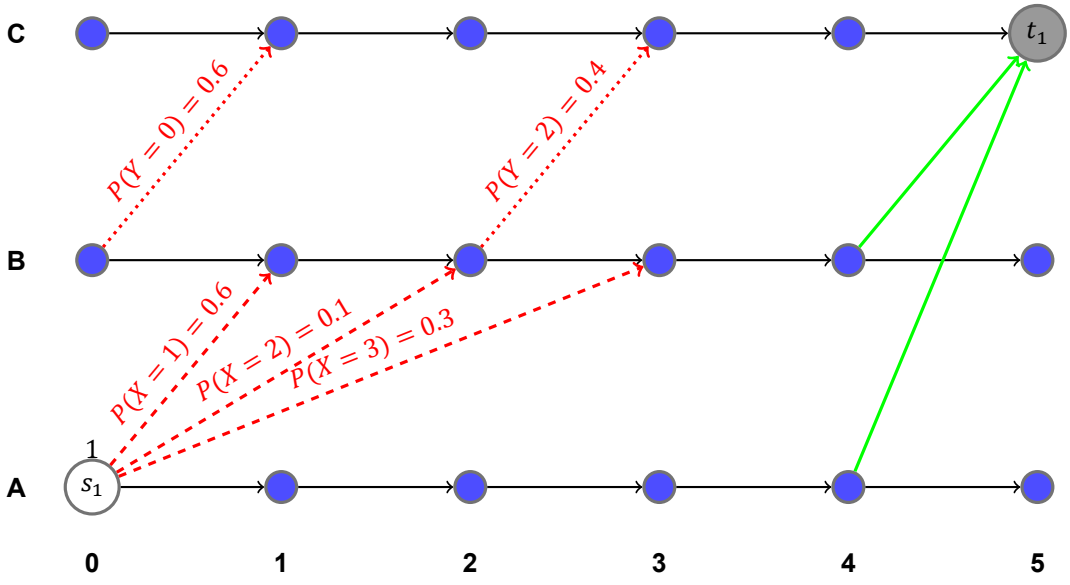


Figure 5.9: An instance with two uncertain transits: one, dashed, from A to B that has uncertain travel time X; another, dotted, from B to C that has uncertain departure time Y. X and Y are independent. $\sigma(70\%, X) = 2$, $\sigma(70\%, Y) = 2$, so in the 70%-pessimistic future, $X = 2$ and $Y = 2$ so it is possible to take the first transit and follow it up with the second. However, the probability that this modality-path indeed exists, is only $P(X \leq 2, Y = 2) = 0.28$, not 0.7. Note also how in the expected future, $X = 2$ and $Y = 1$, so expected future does not equal the 50%-pessimistic future.

6

Deterministic operational freight planning

In this chapter, the final research sub-question will be answered:

4. How can a low cost net-centric operational transport schedule be found fast enough for on-line use if everything is known beforehand?

In the two previous chapters, problems were studied where the timetables of barges and other resources were already fixed and only the container-to-mode assignment had to be optimised. The former chapter observed the deterministic case of this problem, the latter the stochastic case.

In this chapter, however, the decision-maker also fully or partially controls the timetables of the transports. The decision-maker can decide where transports will go and when. The optimisation, now, lies in simultaneous decision-making of fleet routing in space-time and container-to-mode assignment. The final problem of this thesis is the fast optimisation of *deterministic operational freight planning* (Problem 3), by which is the following $\bar{R}, [RD], [RDT] | \bar{D}, [D2R] | social(1)$-problem is meant:

> to simultaneously determine transport routes, determine transport departures times, assign freight containers to transports and determine when and where to load and unload said containers, so that the containers reach their destinations before a deadline against minimum total cost, given that all features of the problem are deterministic.

Again, trucks or subcontracted transportation can either be explicitly modelled, left out or modelled as double matrix infinite resources, as detailed in Section 4.3. A finite time window with finite discretisation is again observed.

An example of this problem is given in Figure 6.1. In the language of the previous chapters, Problem 3 could also be interpreted as follows: how can transit links be placed in a space-time network, respecting travel time and terminal handling time, such that the cost of the link placement and the corresponding minimum cost multi-commodity flow together are minimal? See also Figure 6.2.

This problem has direct applications in practice: for the optimisation of a social synchromodal network of different parties, but also for the more current problem of real-time planning within the transport chain of one party. However, these applications hold only if all stochasticity is discarded. The stochastic version of this problem may better suit operational reality, but is left as future research. The techniques developed here for the deterministic case may form the foundation for such research of the stochastic case, which is the final reason this problem is studied here.

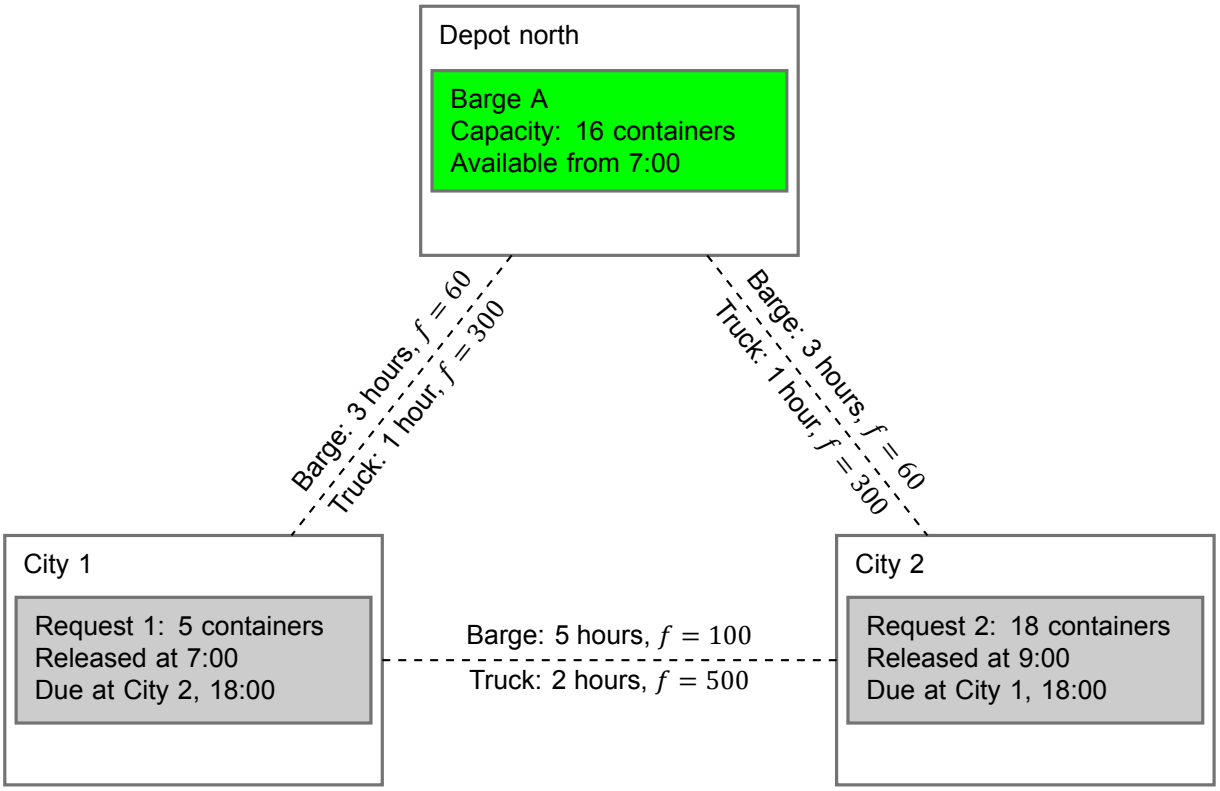


Figure 6.1: Two requests are placed: Request 1 consists of 5 containers that appear at City 1 and are due at City 2 and Request 2 consists of 18 containers the other way round. One barge with a capacity of 16 containers becomes available at the depot at 07:00. Completely loading or unloading the barge at a terminal always costs one hour. Under the assumption of double matrix truck modelling, any amount of trucks can be deployed at any time against given costs and travel times. The problem here is how to assign containers to either the barge or to trucks and how to move the barge around, knowing that trucks are always more expensive but there is not enough time to do everything by barge. Inspection shows that the optimal solution involves transporting 16 containers of Request 2 by only barge and all other containers by only truck.

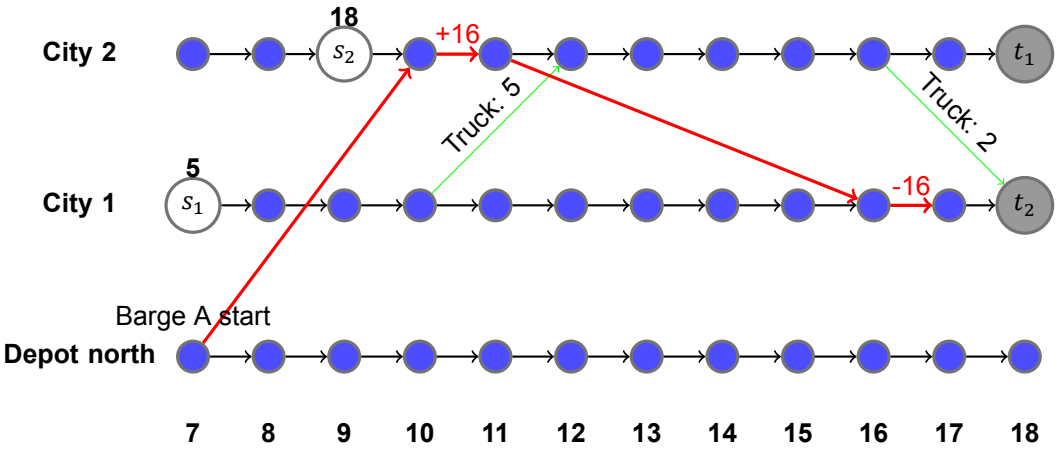


Figure 6.2: One of the optimal solutions of the problem in Figure 6.1 represented in a space-time network. Barge A, represented by the red arcs, starts at Depot north (7:00), departs (7:00), arrives at City 2 (10:00), loads 16 containers of Request 2, departs (11:00), arrives at City 1 (16:00) and unloads the 16 containers of Request 2. At 16:00, the remaining 2 containers of Request 2 are trucked. At 10:00, all 5 containers of Request 1 are trucked. Note that the moments of trucking are quite arbitrary, leading to many similar optimal solutions. The fact that the red barge could choose to arrive one time step later at City 2 or City 1 also creates similar solutions.

6.1. Notation of variables and parameters

The reader is reminded that controlled barges, trains and the like will be referred to as *vehicles*, while separate truck departments and subcontracted transportation will be referred

to as *infinite resources* with infinite capacity but typically high cost. The large and varied decision space of this problem merits a structured introduction of the variables used in this chapter, denoted with lower case letters, before any further discussion:

- $x_{w,k,t} \in \mathbb{N}$: the amount of containers from request k that vehicle w has on board at the start of time t;
- $y_{i,k,t} \in \mathbb{N}$: the amount of containers from request k present at location i at the start of time t;
- $z_{w,i,t} \in \{0,1\}$: indicator variable that equals 1 if and only if vehicle w is present at location i at the start of time t;
- $a_{w,i,t} \in \{0,1\}$: indicator variable that equals 1 if and only if vehicle w arrives at location i just before the start of time t;
- $b_{w,i,j,t} \in \{0,1\}$: indicator variable that equals 1 if and only if vehicle w departs from location i to location j just after the start of time t;
- $r_{w,t} \in \mathbb{N}$: the amount of remaining travel time vehicle w has at the start of time t;
- $h_{w,i,t} \in \{0,1\}$: indicator variable that equals 1 if and only if vehicle w stays at location i throughout time t to handle containers;
- $l_{w,i,k,t} \in \mathbb{N}$: the amount of containers from request k that are loaded from location i onto vehicle w during time t;
- $u_{w,i,k,t} \in \mathbb{N}$: the amount of containers from request k that are unloaded from vehicle w onto location i during time t;
- $q_{i,j,k,t} \in \mathbb{N}$: the amount of containers from request k that are sent by infinite resource from location i to location j, departing just after the start of time t;
- $v_{i,j,k,t} \in \mathbb{N}$: the amount of containers from request k that arrive at location j just before the start of time t, having been sent by infinite resource from location i.

Seeing how the departure variables $b_{w,i,j,t}$ are indexed on vehicle, origin, destination and departure time, one may as well allow for the travel time incurred to also depend on vehicle, origin, destination and departure time. Travel times can thus be defined as parameters $T_{w,i,j,t}$. If an instance has symmetric travel times that depend only on the origin-destination pair, one can simply set the values such that $T_{w,i,j,t} = T_{i,j} = T_{j,i} \quad \forall (w,i,j,t)$. Following this philosophy of generality, the parameters of this problem, denoted with upper case letters, are defined as follows, with $\mathbb{N}_+ = \{1,2,3,\ldots\}$:

- $E_{w,i,j,t} \in \mathbb{N}$: cost incurred if vehicle w travels from location i to location j, departing just after the start of time t;
- $F_{w,i,k,t} \in \mathbb{N}$: unit cost of loading containers from request k from location i onto vehicle w during time t;
- $G_{w,i,k,t} \in \mathbb{N}$: unit cost of unloading containers from request k from vehicle w onto location i during time t;
- $T_{w,i,j,t} \in \mathbb{N}_+$: travel time if vehicle w travels from location i to location j, departing just after the start of time t;
- $P_{w,i,t} \in \mathbb{N}$: processing speed, or total amount of containers that vehicle w can load or unload at location i during time t;
- $N_{i,t} \in \mathbb{N}$: total amount of containers that can be loaded or unloaded at location i during time t;
- $C_{w,t} \in \mathbb{N}_+$: total capacity of vehicle w at the start of time step t;

- $D_k \in \mathbb{N}_+$: volume of request k;
- $O_{i,j,k,t} \in \mathbb{N}$: unit cost of using infinite resources to move containers from request k from location i to location j departing just after the start of time t;
- $S_{i,j,k,t} \in \mathbb{N}_+$: amount of time steps required before containers from request k arrive at location j, having been sent by infinite resource from location i just after the start of time t.

Problem 3, then, is finding a feasible solution that minimises the sum over the vehicle travel costs $E_{w,i,j,t}b_{w,i,j,t}$, the loading costs $F_{w,i,k,t}l_{w,i,k,t}$, the unloading costs $G_{w,i,k,t}u_{w,i,k,t}$ and the costs $O_{i,j,k,t}q_{i,j,k,t}$ incurred by using infinite resources. The definition of 'feasible' will be more thoroughly discussed in Section 6.3.1, but it largely follows such intuitions as 'vehicles may only unload at a location if they are present at the location' and 'vehicles may unload at most as many containers as they have on board'. The discussion up until that section should be clear enough to follow without the formal definition.

As in Section 4.1.3, one can define a soft due date for a request, next to a hard deadline, by setting appropriate values for the time-dependent costs $G_{w,i,k,t}$ of unloading at the due location.

Notably absent from these parameters are, for example, release time and deadline of a request. In this model, it is assumed that if a request k is not released yet, all of its containers are present but stuck at the release location. So for every time step t before the release time, $y_{release-location,k,t} = D_k$ and $y_{not-release-location,k,t} = 0$. Similarly, for every time step t from the deadline onwards, $y_{due-location,k,t} = D_k$ and $y_{not-due-location,k,t} = 0$. By enforcing these variables to have these values, the behaviour of release times and deadlines is achieved. Therefore, the final 'parameter' used in this model is a set Δ of fixed value pairs: for example, $(x_{Barge\ A,101,0}, 2) \in \Delta$ denotes that $x_{Barge\ A,101,0} = 2$ must hold, so that the vehicle $Barge\ A$ has 2 containers from request 101 on board at the start of time 0. If $\Delta = \emptyset$, no variables are enforced to have fixed values.

These fixed value pairs in Δ are important to model the following requirements, among others:

1. At the first time step, vehicles and containers can be at any place undergoing any activity. Δ must enforce the starting situation on any variable;
2. In particular, if using an infinite resource during the time window to transport containers from request k from location i to location j can get them there as early as some time τ according to parameters $S_{i,j,k,t}$, then $v_{i,j,k,t}$ has a fixed value until τ. This fixed value is only greater than 0 if transport was started some time before the start of the time window;
3. As discussed, the requests have release times and deadlines and Δ must enforce values D_k and 0 on variables $y_{i,k,t}$;
4. If a terminal i has closing times, then Δ must enforce $z_{w,i,t} = 0$ for every vehicle w if i is not open at time t;
5. If a terminal has specified the remaining time slots in which vehicles can still be handled, Δ must enforce $z_{w,i,t} = 0$ outside of these time slots;
6. If a terminal i cannot service barges, perhaps because it has no water connection, then Δ must enforce $z_{w,i,t} = 0$ for every vehicle w if w is a barge and for every time t.

Note that in item 4, one could also choose to enforce that $a_{w,i,t} = 0$ rather than $z_{w,i,t} = 0$, seeing how a vehicle cannot be at a location without arriving there and vice versa. In every such case, it is encouraged to fix both values, to speed up the computation in the case of integer linear programming and to make the boundaries of the decision space more explicit.

Note finally that, in this formulation, the amount of containers a vehicle can load or unload in a time step is not request-dependent and the amount of containers a terminal can load or unload depends on neither requests nor vehicles. This may not exactly represent operational reality, but improvements would require some concept of the amount of 'work' loading or unloading certain containers requires, which is left as a topic of future research.

With a notation for this problem introduced, the discussion on the features and solution methods of this problem can continue more efficiently.

6.2. Problem features

This problem is similar to the Capacitated Multicommodity Network Design (CMND) problem investigated by Pedersen [44]. It is quite different, however, from the more well-known Dial-A-Ride-Problem (DARP) [21] and other Vehicle Routing Problems (VRP) [52], in two important senses:

- In intermodal transportation, it may be optimal for goods to be picked up by one vehicle and dropped off by another, changing vehicles along the way any amount of times. However, most DARP-formulations assume that the entire journey from pick-up to drop-off is performed by one vehicle;
- In Problem 3, it is not necessary for vehicles to start and end at some depot location. In fact, the real-time flexibility property of synchromodal planning demands that in the starting situation, vehicles and containers can be at any location undergoing any type of action. However, most VRP-formulations assume that vehicles start and end at some depot.

Therefore, Problem 3 is a departure from many classical optimisation problems. It would seem prudent, thus, to supply a proof why it is still a strongly NP-hard problem. This can be done by a reduction from 3-partition, which is known to be a strongly NP-complete problem [27].

Theorem 1. *Deterministic operational freight planning is a strongly NP-hard optimisation problem.*

Proof. First, observe the decision variant of deterministic operational freight planning: given some instance of deterministic operational freight planning, does it have a feasible solution with cost smaller than or equal to some threshold value Y? This problem is obviously in NP:

- Solutions of this problem can be encoded in polynomial time and space with respect to the input size: the variables, described in Section 6.1, are polynomially many in the amount of vehicles, locations, requests and time steps;
- Whether or not a solution gives YES to the decision problem, can be checked in polynomial time and space with respect to the input size: this is merely a matter of computing whether

$$\sum_{w,i,j,t} E_{w,i,j,t} b_{w,i,j,t} + \sum_{w,i,k,t} F_{w,i,k,t} l_{w,i,k,t} + \sum_{w,i,k,t} G_{w,i,k,t} u_{w,i,k,t} + \sum_{i,j,k,t} O_{i,j,k,t} q_{i,j,k,t} \leq Y$$

Now, take any instance I_1 of 3-partition: thus, take any $m, A_1, A_2, \dots, A_{3m}, B \in \mathbb{Z}$ such that $\sum_{j=1}^{3m} A_j = mB$ and let I_1 be the decision problem 'Can the list of numbers $A_1, A_2, \dots, A_{3m}$ be partitioned into m triplets S_i such that $\sum_{j \in S_i} A_j = B$ for $i = 1, 2, \dots, m$?'

Next, construct the instance I_2 of deterministic operational freight planning illustrated in Figure 6.3 as follows. Let there be m homogeneous vehicles $\{1, 2, \dots, m\}$ with capacity B. Let there be $3m$ requests named $1, 2, \dots, 3m$ respectively, with volume $D_j = A_j$ for $j = 1, 2, \dots, 3m$. Let there be $3m + 1$ locations named $0, 1, 2, \dots, 3m$ respectively. Let the time steps be $0, 1, 2, \dots, 8$. Employ Δ as follows:

- Let all vehicles start empty;

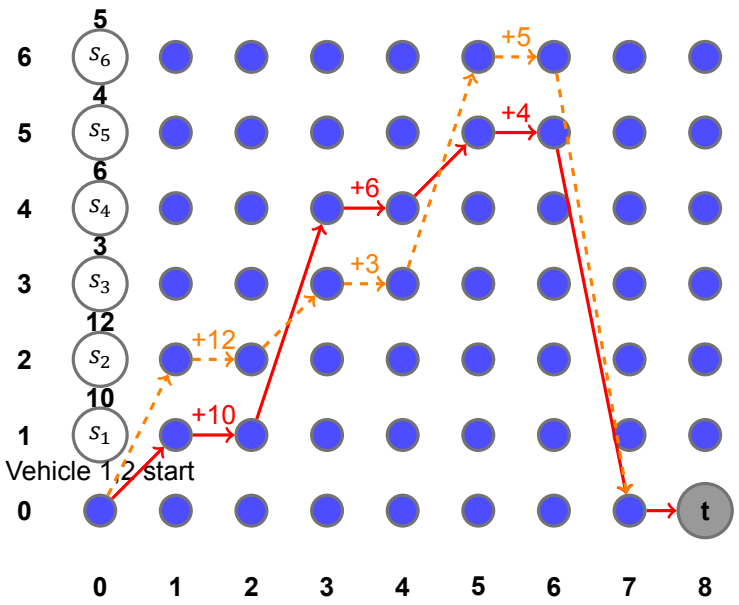


Figure 6.3: A special instance of deterministic operational freight planning can be based on a random instance of 3-partition by giving each item of the partition problem its own request and its own location for release and letting all requests be due at time 8 at location 0. At location 0, m vehicles with capacity B start empty at time 0. Travelling, loading and unloading always costs one time step and has cost 0. Using infinite resources always costs one time step and has cost 1. A resource, thus, has time to take care of at most three requests. This instance has a solution with cost less than or equal to 0, so a solution that uses no infinite resources, if and only if each vehicle fully loads three requests before unloading them all, which is possible if and only if a 3-partition over the m vehicles with capacity B exists. To illustrate, this figure is based on the 3-partition instance where $m = 2$, $B = 20$, $A_1 = 10$, $A_2 = 12$, $A_3 = 3$, $A_4 = 6$, $A_5 = 4$ and $A_6 = 5$, for which the partitioning $S_1 = \{1, 4, 5\}$, $S_2 = \{2, 3, 6\}$ gives YES. This partitioning corresponds to a set of routes in which all vehicles fill up exactly all of their capacity (20) before unloading; no infinite resources are required so the total cost is 0. The first route is indicated in red arcs, the second in orange dashed arcs. For every three items added to the 3-partition problem, three new locations and one new vehicle would appear in this figure.

- Let request k have release time 0, release location k, deadline 7 and due location 0 for $k = 1, 2, \ldots, 3m$;
- Let each vehicle w start at location 0, with $r_{w,0} = 0$ and $a_{w,i,0} = 0$ for each location i;
- Let $v_{i,j,k,0} = 0$ for each location i, each location j and each request k, which will imply that no infinite resources were called upon before time 0 that effect this time window.

Let all vehicle travel costs $E_{w,i,j,t}$, all loading costs $F_{w,i,k,t}$ and all unloading costs $G_{w,i,k,t}$ equal 0. Let all infinite resource costs $O_{i,j,k,t}$ equal 1. Let all vehicle travel times $T_{w,i,j,t}$ and all infinite resource delivery times $S_{i,j,k,t}$ equal 1. Let $P_{w,i,t} = C_w = B$ and $N_{i,t} = mB$ for each vehicle w, each location i and each time step t, which implies that loading and unloading at a location always costs 1 time step, regardless of the amount of containers. With all parameters set, note that each vehicle can fully or partially service at most three different requests, by performing this sequence of actions that each cost one time step:

$$(go\ to\ first\ pickup, load, go\ to\ second\ pickup, load, go\ to\ third\ pickup, load, go\ to\ 0, unload\ all)$$

Note also that for any solution, the objective value has costs equal to the amount of containers transported anywhere by infinite resource.

Finally, let I_3 be the decision problem "Does I_2 have a solution with value less than or equal to 0?" Clearly, decision problem I_3 can be constructed from decision problem I_1 in polynomial time and space in the input size of I_1: this is simply a matter of creating m vehicles, $3m + 1$ locations, $3m$ requests and 9 time steps, assigning a polynomial amount of parameters their given values and creating a polynomially sized list Δ.

If I_1 has answer YES, then let $S_1, S_2, \ldots, S_m$ be a 3-partition. Construct the following solution of I_2: for $w = 1, 2, \ldots, m$, if $S_w = \{\alpha, \beta, \gamma\}$, let vehicle w go to location α, load all containers of request α, do

the same for β and γ, then arrive at location 0 at time 7 and unload all containers so they are in by their deadline 8. As argued earlier, this is feasible in time. By choice of parameters $P_{w,i,t}$ and $N_{i,t}$, this is feasible in the amount of processed containers. Because $\sum_{j \in S_w} = B = C_w$, the capacity of the vehicle is at no time exceeded. All in all, this solution is feasible and requires no infinite resources so its total cost is 0. So then I_3 has answer YES.

If I_3 has answer YES, then observe a solution of I_2 with total cost lower than or equal to 0. This solution, then, must use no infinite resources. Each request must be fully serviced by vehicles. Because of the deadlines, a vehicle can service at most three requests fully or partially. There are m vehicles and $3m$ requests, so if no infinite resource are used, each vehicle must service three requests and each request is serviced by exactly one vehicle. So each vehicle fully picks up three requests, then drops all three off at location 0. It must be possible, then, to partition the requests into triples S_w, where vehicle w takes full responsibility for the requests in S_w. Because the vehicles have capacity B, the sum of volumes D_k within in a triple cannot exceed B. The sum of all volumes is $\sum_{k=1}^{3m} D_k = \sum_{j=1}^{3m} A_j = mB$, so the sum of volumes D_k within a triple must be exactly B. Therefore, observing the three jobs serviced by vehicle w gives a set S_w such that $S_1, S_2, \ldots, S_m$ is a 3-partitioning of $A_1, A_2, \ldots, A_{3m}$ into m partitions of size B, so I_1 has answer YES.

To conclude: I_3 is in NP and can be constructed in polynomial time and space from the decision problem I_1, I_1 and I_3 are equivalent and I_1 is known to be strongly NP-complete. Therefore, I_3 is also strongly NP-complete. I_3 is a decision variant of I_2, so I_2 is strongly NP-hard, and I_2 is an instance of deterministic operational freight planning, so deterministic operational freight planning is strongly NP-hard. □

Therefore, unless $P = NP$, deterministic operational freight planning has no general solution method that runs in poly-time. Additionally:

Theorem 2. *Unless $P = NP$, no Fully Polynomial-Time Approximation Scheme (FPTAS) exists for deterministic operational freight planning.*

Proof. This follows directly from the fact that deterministic operational freight planning is a strongly NP-hard optimisation problem [41]. □

So unless $P = NP$, each general solution method for deterministic operational freight planning runs in an exponential amount of time and each non-exponential general approximation method is a Polynomial-Time Approximation Scheme (PTAS) at best.

6.2.1. A note on labour conditions

Depending on the time-scale of the problem instance, it may be important to also model that barge teams cannot work around the clock for weeks: sleep and refuelling may be necessary. It is likely possible to add rest periods with time windows to this model for Problem 3 by creating 'requests' that represent a rest period, which can only be executed by a specific vehicle cheaply, so as to force them to go to the resting place within a specific time window. It is also likely that labour conditions can be modelled in with some extra variables and constraints instead, and that this is a less far-fetched solution. Expanding the model to encompass labour conditions is left as a topic of future research.

6.3. Solving to optimality

The decision space of Problem 3 is obviously much larger and more convoluted than that of Problem 1. However, Problem 3 may still be solved to optimality for small problems in a reasonable amount of time, using the ILP developed in this section.

6.3.1. ILP formulation

Denote W the set of vehicles. Denote I the set of locations. Denote K the set of requests. Without loss of generality, denote $T = \{0, 1, 2, \ldots, end\} \subset \mathbb{N}$ the set of time steps. Using the notation

from Section 6.1, Problem 3 can be formulated as the following integer linear program:

$$\begin{aligned}
\min \quad & \textstyle\sum_{w,i,j,t} E_{w,i,j,t} b_{w,i,j,t} + \sum_{w,i,k,t} F_{w,i,k,t} l_{w,i,k,t} \\
+ \quad & \textstyle\sum_{w,i,k,t} G_{w,i,k,t} u_{w,i,k,t} + \sum_{i,j,k,t} O_{i,j,k,t} q_{i,j,k,t} \\
s.t. \quad & var = val && \forall (var, val) \in \Delta && (6.1)\\
& x_{w,k,t} = x_{w,k,t-1} + \textstyle\sum_{i \in I} l_{w,i,k,t-1} - \sum_{i \in I} u_{w,i,k,t-1} && (\forall w \in W)(\forall k \in K)(\forall t \in T \backslash \{0\}) && (6.2)\\
& y_{i,k,t} = y_{i,k,t-1} - \textstyle\sum_{w \in W} l_{w,i,k,t-1} + \sum_{w \in W} u_{w,i,k,t-1} \\
& \quad - \textstyle\sum_{j \in I} q_{i,j,k,t-1} + \sum_{j \in I} v_{j,i,k,t} && (\forall i \in I)(\forall k \in K)(\forall t \in T \backslash \{0\}) && (6.3)\\
& z_{w,i,t} = z_{w,i,t-1} + a_{w,i,t} - \textstyle\sum_{j \in I} b_{w,i,j,t-1} && (\forall w \in W)(\forall i \in I)(\forall t \in T \backslash \{0\}) && (6.4)\\
& \textstyle\sum_{i \in I} u_{w,i,k,t} \leq x_{w,k,t} && (\forall w \in W)(\forall k \in K)(\forall t \in T) && (6.5)\\
& \textstyle\sum_{j \in I} q_{i,j,k,t} + \sum_{w \in W} l_{w,i,k,t} \leq y_{i,k,t} && (\forall i \in I)(\forall k \in K)(\forall t \in T) && (6.6)\\
& v_{i,j,k,t} = \textstyle\sum_{\tau \in T^*} q_{i,j,k,\tau} \ (T^* = \{\tau \in T : \tau + S_{i,j,k,\tau} = t\}) && (\forall i,j \in I)(\forall k \in K)(\forall t \in T) && (6.7)\\
& \textstyle\sum_{k \in K} x_{w,k,t} \leq C_{w,t} && (\forall w \in W)(\forall t \in T) && (6.8)\\
& \textstyle\sum_{k \in K} l_{w,i,k,t} + \sum_{k \in K} u_{w,i,k,t} \leq P_{w,i,t} h_{w,i,t} && (\forall w \in W)(\forall i \in I)(\forall t \in T) && (6.9)\\
& \textstyle\sum_{w \in W} \sum_{k \in K} l_{w,i,k,t} + \sum_{w \in W} \sum_{k \in K} u_{w,i,k,t} \leq N_{i,t} && (\forall i \in I)(\forall t \in T) && (6.10)\\
& h_{w,i,t} \leq z_{w,i,t} && (\forall w \in W)(\forall i \in I)(\forall t \in T) && (6.11)\\
& a_{w,i,t} \leq \textstyle\sum_{j \in I} \sum_{\tau < t} b_{w,j,i,\tau} - \sum_{\tau < t} a_{w,i,\tau} && (\forall w \in W)(\forall i \in I)(\forall t \in T \backslash \{0\}) && (6.12)\\
& r_{w,t} \geq r_{w,t-1} - 1 && (\forall w \in W)(\forall t \in T \backslash \{0\}) && (6.13)\\
& r_{w,t} \geq \textstyle\sum_{i \in I} \sum_{j \in I} T_{w,i,j,t} b_{w,i,j,t} && (\forall w \in W)(\forall t \in T) && (6.14)\\
& r_{w,t} \leq end - end \cdot \textstyle\sum_{i \in I} a_{w,i,t} && (\forall w \in W)(\forall t \in T) && (6.15)\\
& \textstyle\sum_{j \in I} b_{w,i,j,t} \leq z_{w,i,t} && (\forall w \in W)(\forall i \in I)(\forall t \in T) && (6.16)\\
& \textstyle\sum_{j \in I} b_{w,i,j,t} + h_{w,i,t} \leq 1 && (\forall w \in W)(\forall i \in I)(\forall t \in T) && (6.17)\\
& \textstyle\sum_{i \in I} \sum_{j \in I} b_{w,i,j,t} \leq 1 && (\forall w \in W)(\forall t \in T) && (6.18)\\
& b_{w,i,i,t} = 0 && (\forall w \in W)(\forall i \in I)(\forall t \in T) && (6.19)\\
& q_{i,i,k,t} = 0 && (\forall i \in I)(\forall k \in K)(\forall t \in T) && (6.20)\\
& v_{i,i,k,t} = 0 && (\forall i \in I)(\forall k \in K)(\forall t \in T) && (6.21)\\
& a_{w,i,t}, b_{w,i,j,t}, h_{w,i,t}, z_{w,i,t} \in \{0,1\} && \forall (w,i,j,k,t) \in W \times I^2 \times K \times T && (6.22)\\
& l_{w,i,k,t}, q_{i,j,k,t}, r_{w,t}, u_{w,i,k,t}, v_{i,j,k,t}, x_{w,k,t}, y_{i,k,t} \in \mathbb{N} && \forall (w,i,j,k,t) \in W \times I^2 \times K \times T && (6.23)
\end{aligned}$$

The cost function minimises the total costs of moving the vehicles around, loading containers, unloading containers and employing infinite resources. The reader is reminded that unit penalties for delivering after a soft due date can be embedded into the time-dependent cost parameters. Equalities (6.1) state that some variables have fixed values, as detailed in Section 6.1. Equalities (6.2) state that the amount of containers from request k a vehicle has on board at any time equals the amount it held in the previous time step, plus the amount it has loaded in the previous time step, minus the amount it has unloaded in the previous time step. Equalities (6.3) state that the amount of containers from request k present at a location at any time equals the amount present in the previous time step, minus the amount vehicles took away in the previous time step, plus the amount vehicles have unloaded here in the previous time step, minus the amount sent away by infinite resource in the previous time step, plus the amount that appears here from having been sent by infinite resource. Equalities (6.4) state that whether or not a vehicle is present at a location at any time depends on whether it was present in the previous time step, whether it arrived just before the start of this time step and whether it has departed to another location during the previous time step. Inequalities (6.5) state that a vehicle cannot unload more containers from request k than it has on board. Inequalities (6.6) state that no more containers from request k can be sent away by infinite resource or loaded onto vehicles than there are present. Equalities (6.7) ensure that if containers are sent away by infinite resource, they arrive at the right point in space-time. Inequalities (6.8) state the total amount of containers on board of a vehicle may never exceed the vehicle capacity. Inequalities (6.9) state that the total amount of containers

that a vehicle can load or unload, or process, at a location is limited by a processing speed. Furthermore, processing can only be done if the vehicle has explicitly decide to spend this time step on handling goods at this location. Inequalities (6.10) state that a location can handle only so many containers in one time step. Inequalities (6.11) state that a vehicle can only stay at a location to handle goods if it is present. Inequalities (6.12) state that a vehicle can only arrive at a location if it has departed towards that location more often then it has arrived there; in other words, a vehicle can only arrive somewhere if it has travelled to that location, not counting previous trips. Inequalities (6.13) enact a 'remaining travel time counter': each time step, it may be decremented by 1, but it always remains greater than or equal to 0. When departing from one location to another, inequalities (6.14) set that counter equal to the travel time. Inequalities (6.15) make it so that a vehicle cannot arrive somewhere at any time step while it has more than zero remaining travel time on its counter; if some $a_{w,i,t}$ equals 1, then $r_{w,t}$ must be less than or equal to 0, while if all $a_{w,i,t}$ equal 0, $r_{w,t}$ must be less than or equal to the length of the time window, which is a reasonable upper bound on any $r_{w,t}$. Inequalities (6.16) state that a vehicle can only depart from a location if it is present at that location. Inequalities (6.17) state that is not allowed for a vehicle to both leave from a location and stay at the location to handle goods in the same time step. Inequalities (6.18) state that a vehicle can depart to only one location at a time; if this were not present, it could start 'manifesting' at multiple locations. Equalities (6.19), (6.20) and (6.21) disallow that anything 'goes' from location i to location i by setting the appropriate variables equal to 0; the same could be achieved by leaving the variables out or enforcing fixed values upon them with Δ. Finally, (6.22) and (6.23) state that some variables are binary decision variables and some variables are non-negative integer decision variables.

Notably absent from this formulation are inequalities that a vehicle can only be at one place at a time and an analogue for containers. However, if the starting situation is properly defined with Δ, the constraints disallow for vehicles and containers to 'manifest' at several places at the same time, so the proper behaviours are implicitly present. Whether or not to make these explicit, is discussed in Section 6.3.2.

Furthermore, it must be noted that the amount of variables and constraints grows polynomially, but still unfavourably fast, in the amount of vehicles, locations, requests and time steps. For example, the amount of variables $l_{w,i,k,t}$ is equal to the amount of vehicles times the amount of locations times the amount of requests times the amount of time steps; additionally, each of these variables may take any integer between 0 and D_k as its value. Future research may have to investigate ways to make this model better scalable, possibly by employing reasoning similar to those used in simplifying instances of Problem 1.

6.3.2. Speed-up from additional constraints

This section describes a number of constraints that can be added to the ILP from Section 6.3.1. These do not change the integral solution set, because they are explicit versions of constraints that are already implicitly true. However, it will be shown in Section 6.6 that including them significantly reduces computation time. Most likely, this is because they constrain the solution polytope in such a way that the LP-relaxations can be solved more easily, but explanations were not further investigated.

The added constraints are as follows:

$$s.t. \quad \sum_{i \in I} z_{w,i,t} \leq 1 \quad (\forall w \in W)(\forall t \in T) \tag{6.24}$$

$$\sum_{i \in I} a_{w,i,t} \leq 1 \quad (\forall w \in W)(\forall t \in T) \tag{6.25}$$

$$\sum_{i \in I} h_{w,i,t} \leq 1 \quad (\forall w \in W)(\forall t \in T) \tag{6.26}$$

Inequalities (6.24) state that a vehicle cannot be at more than one location at a time. Inequalities (6.25) state that a vehicle cannot arrive at more than one location at a time. Inequalities (6.26) state that a vehicle cannot handle goods at more than one location at a time.

Although the results in 6.6 show that these added constraints significantly reduce computation time, they also show that the achieved computation times are still too long to make this method useful for on-line use in problems of 'real life size'.

6.4. Greedy Gain heuristic

As discussed earlier, heuristics will have to be used to find reasonable solutions for Problem 3 fast enough for on-line use. A first step is often to develop a simple, greedy algorithm: while not sophisticated, they are often fast and serve as a good starting point for further development.

The idea of the *Greedy Gain heuristic* (GG) is the following. Suppose for now that the infinite resources are trucks and none of the vehicles start with containers on board. The vehicles are not under way to some location; they are simply waiting somewhere. Requests may already be released and scattered over locations, but they are not on board of vehicles or trucks: they are simply waiting. Each of these batches is considered a separate request. Then, an initial solution can be found by simply trucking everything at its cheapest possible time: this solution is easy to find, but probably very expensive, because infinite resources are typically more expensive than vehicles and the vehicles are not being used at all. In the Greedy Gain heuristic, each request is then assigned to one vehicle, one at a time. To determine which request to assign to which vehicle next, it is checked how much can be gained immediately from each assignment, then the assignment with the highest gain is chosen, as long as it is positive. When such an assignment is done, the request is 'erased' from the system: if 9 containers of this request are on board of a vehicle w, this is no longer described by variables, but rather by w having 9 less capacity at that time in future problems. This way, throughout most of the algorithm, only instances of Problem 3 are solved with just one vehicle and just one request: this counteracts the fast scaling of the ILP computation time. An example of the GG process is shown in Figure 6.4, Figure 6.5 and Figure 6.6.

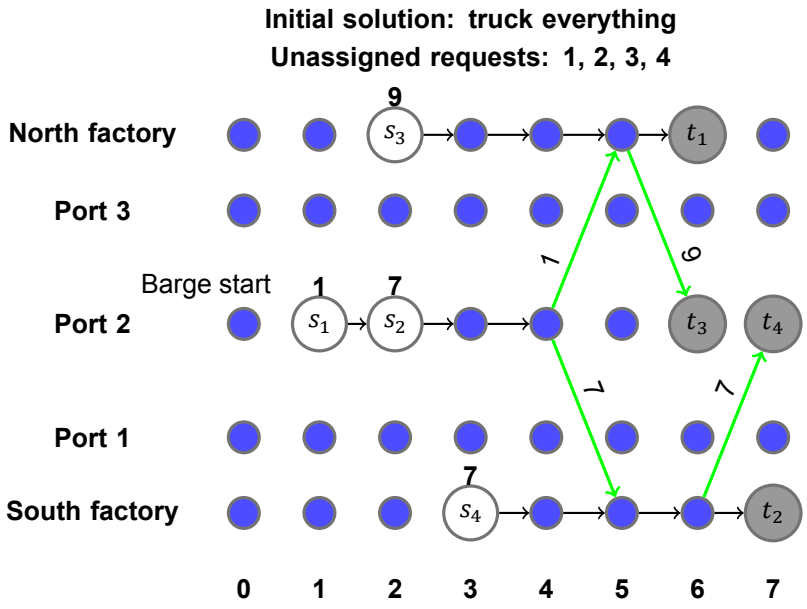


Figure 6.4: An instance of Problem 3 with one barge and four requests. The barge can visit ports, but not factories. South factory is very close to Port 1 and North factory very close to Port 3; for the rest, distances and prices are all the same. Infinite resources, in the form of trucks, are more expensive than barge usage. The barge starts at Port 2 with nothing on board. The barge has capacity 20: $C_{barge,t} = 20$ for $t = 0, 1, \ldots, 7$. In the Greedy Gain heuristic, the initial solution is to just truck everything; in each iteration, one of the requests will be added to a vehicle's schedule.

If, however, in the starting situation a vehicle is not waiting but under way, let it become

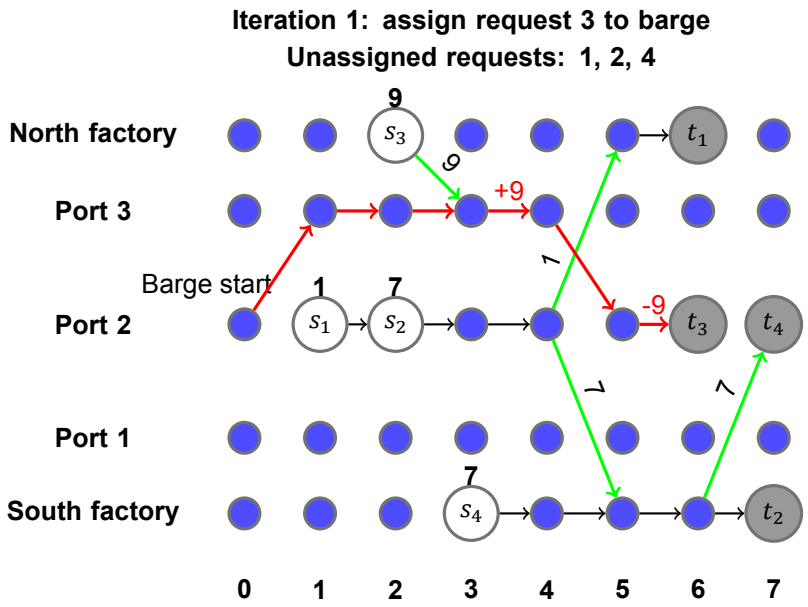


Figure 6.5: In the first iteration, request 3 is assigned to the barge, because this assignment has the highest immediate gain: namely, 9 containers are now largely transported by barge instead of truck. In future iterations, request 3 no longer computationally 'exists'; instead, the barge has a mysterious appointment at Port 3, time 3, in which its capacity changes from $C_{barge,3} = 20$ to $C_{barge,4} = 11$, and a mysterious appointment at Port 2, time 5, in which its capacity changes from $C_{barge,5} = 11$ to $C_{barge,6} = 20$. This way, the sub-problem of adding request 1 or another request to the current schedule of the barge, involves only variables of the barge and the new request, not of the old request; this makes it computationally more manageable.

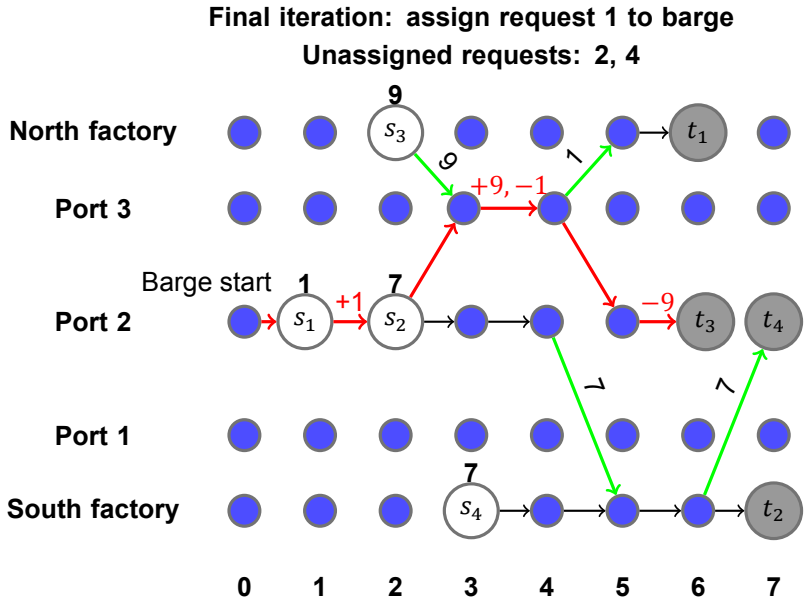


Figure 6.6: In the second iteration, the algorithm would like to assign one of the requests of 7 containers to the barge, but it cannot: the barge already has appointments at time 3 and time 5, so it cannot wait for the release of request 2 or 4. The containers of request 2 could theoretically be trucked to Port 3 so they can be loaded during the existing appointment and unloaded during the second appointment, but this would be more expensive than just trucking them straight to South factory, so this assignment would have negative gain. The only assignment left with positive gain, is to add request 1 to the schedule of the barge; it consists of only one container, but the barge is already taking a favourable route for it anyway. After this assignment, the next iteration has only negative gains, so the algorithm stops. The final solution is sub-optimal: the barge handles request 1 and 3 for a total of 10 containers, while it would have been smarter to handle request 2 and 4 for a total of 14 containers.

available to the algorithm only as soon as it arrives at its destination by enforcing values of arrival variables $a_{w,i,t}$ and presence variables $z_{w,i,t}$ with Δ. Given an instance of Problem 3, these values should already be specified in its set Δ. If containers are under way on an

infinite resource to some location j, arriving at t, ignore them if j is their due location, and interpret them as a batch that is released at j, time t otherwise. If containers start on board of a vehicle, interpret them as already assigned to that vehicle; in the initial solution, that vehicles handles only those containers and the resulting schedule is again expressed in parameter changes rather than variables.

With these details addressed, the Greedy Gain heuristic is formally presented in Algorithm 4.

The most potent upside of the GG heuristic is that, throughout most of the algorithm, only sub-problems are solved with just one vehicle and one request; this counteracts the fast computational growth in the amount of vehicles and requests. However, it does nothing to reduce the amount of locations and time steps in a sub-problem; in Section 6.6, it will become clear that these still greatly contribute to the computation time. Reducing the amount of locations and time steps in a sub-problem, or solving a sub-problem without the use of the ILP, may be worthwhile venues for future research.

An obvious downside of the GG heuristic, being a greedy algorithm, is that it may make 'bad choices' in early stages that influence the decision space of later iterations. This happens in Figure 6.5, where greedily picking the request of size 9 makes it so that in the next iteration, the algorithm can only pick a 'close-by' request of size 1; a better solution would have been attained if the two 'close-by' requests of size 7 were picked. It would be smarter, thus, if an algorithm could somehow recognise the 'compatible' requests of size 7 as one cluster and the other two as another cluster, then assign the vehicle to the cluster with the best value. This idea is the basis of the next heuristic discussed.

Data: Instance of Problem 3 (set of vehicles W, set of locations I, set of requests K, list of time steps; set Δ that forces fixed values to describe starting distribution and system properties; parameters as described in Section 6.1)
Result: Solution of Problem 3, possibly non-optimal
for *k in set of requests* **do**
  Interpret each separate batch of containers belonging to k as a 'separate request' κ;
end
for *w in set of vehicles* **do**
  Initialise $AR(w)$, the set of 'separate requests' assigned to w, as only those already on board in the starting situation ;
  if $AR(w) = \emptyset$ **then**
    Initialise $H(w) =$, the set of fixed values of h that enforce the current schedule of w;
    Initialise $f(w)$, the cost of the current schedule of w, as 0;
  end
  else
    Solve, using the ILP, the sub-problem where only w handles only the requests in $AR(w)$, denote the solution X;
    Initialise $H(w) = \{(h_{w,i,t}, 1) | h_{w,i,t} = 1 \ in \ X\}$, initialise $f(w)$ the cost of X;
    Update $C_{w,t} := C_{w,t} - \sum_{\kappa \in AR(w)} x_{w,\kappa,t}$;
    Update $P_{w,i,t} := P_{w,i,t} - \sum_{\kappa \in AR(w)} (l_{w,i,\kappa,t} + u_{w,i,\kappa,t})$;
    Update $N_{i,t} := N_{i,t} - \sum_{\kappa \in AR(w)} (l_{w,i,\kappa,t} + u_{w,i,\kappa,t})$;
  end
end
Initialise the unassigned requests,
$UR = (all \ separate \ requests) \backslash \{AR(w) | w \ in \ set \ of \ vehicles\}$;
for *unassigned request κ in UR* **do**
  Compute infinite resource cost $IRC(\kappa)$ the cheapest cost of sending all of κ from its release location to its due location by infinite resource ;
  for *vehicle w in set of vehicles* **do**
    Create, solve and store sub-problem where only w handles only κ, respecting the current $H(w)$, $C_{w,t}$, $P_{w,i,t}$ and $N_{i,t}$;
    Denote $f(w + \kappa)$ the cost of this sub-problem ;
    Compute $gain(w, \kappa) = (f(w) + IRC(\kappa)) - f(w + \kappa)$;
  end
end
while $UR \neq \emptyset$ *and* $\max_{\kappa \in UR} gain(w, \kappa) > 0$ **do**
  Determine $(w, \kappa) = \arg\max gain(w, \kappa)$ with corresponding solution X ;
  Update $AR(w) := AR(w) \cup \{\kappa\}$, $UR := UR \backslash \{\kappa\}$;
  Update $H(w) := H(w) \cup \{(h_{w,i,t}, 1) | h_{w,i,t} = 1 \ in \ X\}$, $f(w) := f(w + \kappa)$;
  Update $C_{w,t} := C_{w,t} - \sum_{\kappa \in AR(w)} x_{w,\kappa,t}$;
  Update $P_{w,i,t} := P_{w,i,t} - \sum_{\kappa \in AR(w)} (l_{w,i,\kappa,t} + u_{w,i,\kappa,t})$;
  Update $N_{i,t} := N_{i,t} - \sum_{\kappa \in AR(w)} (l_{w,i,\kappa,t} + u_{w,i,\kappa,t})$;
  for *unassigned request κ in UR* **do**
    Create, solve and overwrite new sub-problem where only w handles only κ, respecting the current $H(w)$, $C_{w,t}$, $P_{w,i,t}$ and $N_{i,t}$;
    Denote $f(w + \kappa)$ the cost of this sub-problem ;
    Compute $gain(w, \kappa) = (f(w) + IRC(\kappa)) - f(w + \kappa)$;
  end
end
Compute and return solution where all $\kappa \ in \ UR$ are trucked as in the initial solution and the solved (w, κ)-sub-problems enforce values on h, l, q, u, v, x, y through Δ.

Algorithm 4: The Greedy Gain heuristic: completely assign requests to vehicles, one at a time, picking the one with highest immediate gain. This gain is computed by observing sub-problems of one vehicle and one request, where the other requests handled by this vehicle are embedded as parameters: capacities are decreased and handling appointments enforced.

6.5. Compatibility Clustering heuristic

The idea of the *Compatibility Clustering heuristic* (CC heuristic) is as follows: if there are n requests and m vehicles, divide the requests into m clusters of requests that are 'compatible', that is to say, likely to be handled together by one vehicle efficiently. If two requests have releases at the same location at virtually the same time, and the same goes for their due points in space-time, they are extremely compatible: it is very likely efficient that one vehicle handles both these requests. If the due node in space-time of the first request almost coincides with the release node of the second request, so dropping off the first request smoothly leads into picking up the second request, these requests are also quite compatible, but not as compatible as in the previous situation: in the first situation, the requests share two places in space-time where the vehicle is needed, where in the latter, they share only one. See also Figure 6.7.

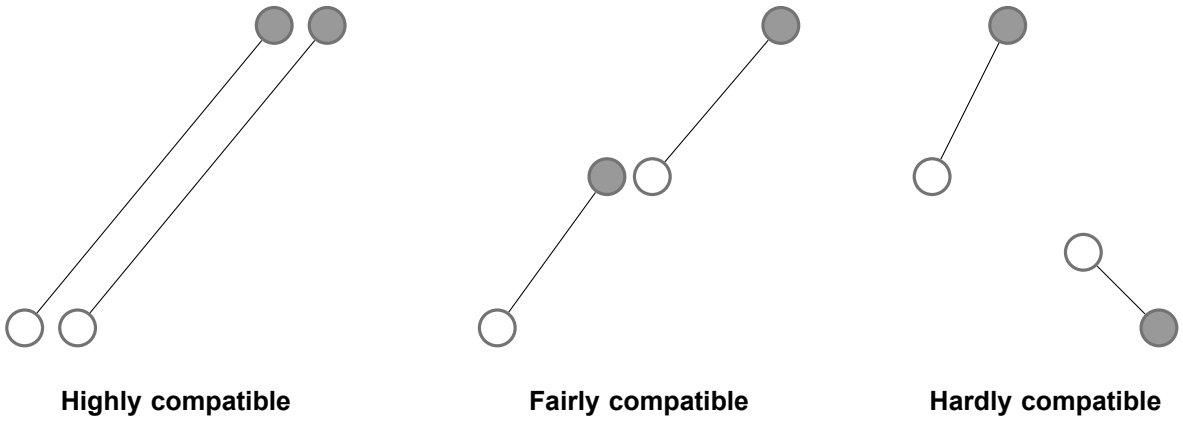


Figure 6.7: Pairs of requests with different 'compatibility', that is to say, likeliness that it is efficient to assign them both to the same vehicle. The white nodes signify release points in space-time, the grey nodes signify where they are due. The left pair consists of two requests that have to be picked up around the same place in space-time and dropped of around the same place as well: they share two places in space-time where a vehicle would have to go to. The middle pair shares only one point in space-time and the right pair shares none.

In order to actually use a existing clustering algorithms to cluster requests on their 'compatibility', the compatibility of requests must be properly expressed as a metric. Before this is done, however, some remarks and an outline of the CC heuristic are given, to show that some other metrics are necessary as well.

Compatibility must depend on both space and time: if two requests have the same origin-destination pair, but one has its release today and the other next week, their shared origin-destination pair loses all value. Understanding this, suppose the time window is $2m$ days. If the requests are blindly divided into m clusters, it may well happen that the first cluster consists of all requests of the first two days, the second cluster comprises the second two days, et cetera. Assigning vehicles to clusters, then, means telling one vehicle to try and take care of everything that happens in two days and not contributing anything in all other days: this is not the desired planning behaviour. So the CC heuristic consists of first splitting up the instance into a given amount of periods, based on time, then within each period finding m clusters, based on space-time. Vehicles should then be assigned to a sequence of clusters, one cluster for each period. To determine these sequences smartly, one would have to know how 'connectible' a cluster of one period is with a cluster of the next period, disregarding time: if there are three remote major areas of activity and three vehicles, one would try to sequence the clusters so the vehicles can stay within one area as much as possible. Thus, the CC heuristic needs a variety of different metrics, which are developed in Section 6.5.1.

6.5.1. Used metrics

This section introduces the metrics necessary for clustering. The reader is reminded that any distance function $d : X \times X \to [0, \infty)$ is a metric and the pair (X, d) a metric space if and only if they satisfy the following properties:

1. $d(x, y) \geq 0$ for all $x, y \in X$;
2. $d(x, x) = 0$ for all $x \in X$;
3. $d(x, y) = 0 \Rightarrow x = y$ for all $x, y \in X$;
4. $d(x, y) = d(y, x)$ for all $x, y \in X$;
5. $d(x, z) \leq d(x, y) + d(y, z)$ for all $x, y, z \in X$.

Famous metrics include Euclidean distance and the taxicab metric. Furthermore, Fujita formulated the following crucial result: if (X, d) is a metric space, and A and B are finite subsets of X, then the 'average distance' between points in A and points in B is a function that again satisfies the properties of a metric [26].

First, if $s_1 = (v_1, t_1)$ and $s_2 = (v_2, t_2)$ are two space-time nodes in an instance of Problem 3, the *vehicle-based space-time distance between space-time nodes*, $\sigma_w((v_1, t_1), (v_2, t_2))$ is defined as

$$\sigma_w(s_1, s_2) = \max\{T_{w,v_1,v_2,t_1}, |t_2 - t_1|\}$$

It can be interpreted as follows: if vehicle w wants to perform an infinitely short action at one place in space-time and then another action at another location in space as soon as possible, how much time steps will there be between these actions? If these actions are picking up containers but their release times are far apart, σ_w equals that difference in time, $|t_2 - t_1|$. If instead the actions occur very close together in time, the difference is the time it takes to get from the first location to the second, T_{w,v_1,v_2,t_1}.

Theorem 3. *Under the assumptions that for all locations i and j, $T_{w,i,j,t}$ is time-independent, is symmetric in i, j, equals 0 if $i = j$ and is positive otherwise, and satisfies the triangle inequality, $\sigma_w(s_1, s_2)$ is a metric on the space-time nodes of the instance.*

Proof.

1. By assumption, $T_{w,i,j,t} \geq 0$ for all locations i, j. Also, $|x - y| \geq 0$ for all $x, y \in \mathbb{R}$. So $\sigma_w(s_1, s_2) \geq 0$ for all space-time nodes s_1, s_2.
2. $\sigma_w(s_1, s_1) = \max\{T_{w,v_1,v_1,t_1}, |t_1 - t_1|\} = \max\{0, 0\} = 0$.
3. Suppose $\sigma_w(s_1, s_2) = 0$ for some space-time nodes $s_1 = (v_1, t_1)$, $s_2 = (v_2, t_2)$.
   So $T_{w,v_1,v_1,t_1}, |t_2 - t_1| \leq 0$. $|t_2 - t_1| \leq 0 \Rightarrow t_1 = t_2$, because the Euclidean distance is a metric.
   By assumption, $T_{w,v_1,v_2,t} \leq 0$ implies that $v_1 = v_2$. So $s_1 = (v_1, t_1) = (v_2, t_2) = s_2$.
4. Because $T_{w,i,j,t}$ is assumed to be symmetric in i, j and time-independent and $|t_2 - t_1| = |t_1 - t_2|$, it follows that $\sigma_w(s_1, s_2) = \max\{T_{w,v_1,v_2,t_1}, |t_2 - t_1|\} = \max\{T_{w,v_2,v_1,t_2}, |t_1 - t_2|\} = \sigma_w(s_2, s_1)$.
5. Let s_1, s_2 and s_3 be space-time nodes of the instance. Suppose $\sigma_w(s_1, s_3) = T_{w,v_1,v_3,t_1}$. By the assumptions of triangle inequality and time independence, $T_{w,v_1,v_3,t_1} \leq T_{w,v_1,v_2,t_1} + T_{w,v_2,v_3,t_2}$, so

   $$\sigma_w(s_1, s_3) = T_{w,v_1,v_3,t_1} =\leq T_{w,v_1,v_2,t_1} + T_{w,v_2,v_3,t_2} \leq \sigma_w(s_1, s_2) + \sigma_w(s_2, s_3)$$

   If instead $\sigma_w(s_1, s_3) = |t_3 - t_1|$,

   $$\sigma_w(s_1, s_3) = |t_3 - t_1| = |(t_3 - t_2) + (t_2 - t_1)| \leq |t_3 - t_2| + |t_2 - t_1| \leq \sigma_w(s_1, s2) + \sigma_w(s_2, s_3)$$

□

A request is largely described by two space-time nodes: its release node in space-time and its due node in space-time. Theoretically, it is not at all necessary that a a request be picked up near its release time and delivered near its deadline, nor at the release or due locations, but it may often be the case in efficient synchromodal practice. If a request J_1 is seen as a pair of two nodes $\{s_1, s_2\}$, and another request as $J_2 = \{s_3, s_4\}$, then if every pair of request with the same release node and due node are considered as one large request, the result of Fujita can be used to define the *vehicle-based space-time distance between requests*, $d_w(J_1,J_2)$:

$$d_w(J_1,J_2) = \frac{1}{2|\{s_1,s_2,s_3,s_4\}|}\left(\sum_{s_i\in\{s_1,s_2\}}\sum_{s_j\in S_2\setminus S_1}\sigma_w(s_i,s_j) + \sum_{s_j\in\{s_3,s_4\}}\sum_{s_j\in S_1\setminus S_2}\sigma_w(s_i,s_j)\right)$$

Theorem 4. *If σ_w is a metric on space-time nodes, d_w is a metric on requests.*

Proof. This follows immediately from Fujita. The concerned reader may note that if $J_i = \{release\ node\ i, due\ node\ i\}$ and $J_1 = J_2$, then it might be that $release\ node\ 1 = due\ node\ 2$ and $due\ node\ 1 = release\ node\ 2$; however, assuming that due nodes are always strictly further in time than release nodes, $J_1 = J_2$ implies that the requests have equal release nodes and equal due nodes. □

It was explained earlier in Section 6.5 that requests will first be divided over a specified amount of time periods. This can also be done with clustering, using a metric that depends only on time, namely the *temporal distance between requests*:

$$\tau(J_1,J_2) = \frac{1}{2|\{s_1,s_2,s_3,s_4\}|}\left(\sum_{s_i\in\{s_1,s_2\}}\sum_{s_j\in S_2\setminus S_1}|t_j - t_i| + \sum_{s_j\in\{s_3,s_4\}}\sum_{s_i\in S_1\setminus S_2}|t_j - t_i|\right)$$

From Fujita and the fact that $|t_j - t_i|$ is an Euclidean distance, it immediately follows that τ is a metric on the requests.

Finally, the need was argued to express connectability of request clusters from different time periods, based mainly on spatial distance. This is done by first describing the *vehicle-based spatial distance between space-time nodes* s_1, s_2 as $\pi_w(s_1,s_2) = T_{w,v_1,v_2,t_1}$, then the *vehicle-based spatial distance between requests*, denoted $\pi'_w(J_1,J_2)$, as the average spatial distance between the sets $\{s_1,s_2\}$ and $\{s_3,s_4\}$, then describing the *vehicle-based spatial distance between clusters* $C_1 = \{J_1,\dots,J_2\}$ and $C_2 = \{J_3,\dots,J_4\}$, denoted $\pi''_w(C_1,C_2)$, as the average spatial distance between the sets $\{J_1,\dots,J_2\}$ and $\{J_3,\dots,J_4\}$. So

$$\pi''_w(C_1,C_2) = \frac{1}{|C_1\cup C_2||C_1|}\sum_{J_i\in C_1}\sum_{J_j\in C_2\setminus C_1}\pi'_w(J_i,J_j) + \frac{1}{|C_1\cup C_2||C_2|}\sum_{J_j\in C_2}\sum_{J_i\in C_1\setminus C_2}\pi'_w(J_i,J_j)$$

If the same assumptions on $T_{w,i,j,t}$ apply as in Theorem 3, π'' is a metric on request clusters, though this is not actually necessary for the functioning of the algorithm described in Section 6.5.2.

With these metrics, it becomes possible to cluster requests before assigning them. Three downsides were discovered, however, in having to use metrics. For one, the functions are currently only guaranteed metrics if the travel times $T_{w,i,j,t}$ 'behave metrically' and are time-independent: though this is often quite a natural assumption to make, it is still a loss of generality. Secondly, if two requests are released and due at exactly the same times but slightly different locations, they may in practice be exactly *not* compatible, because the vehicle can only be present at one location for loading and miss the loading window for the other; if however compatibility were modelled to decrease when requests come too close together, but be perfect when the requests exactly overlap, this would lead to an inevitable loss of the triangle inequality. Fortunately, this 'non-decreasingness' of compatibility is often not a problem, assuming that infinite resources can be used to get both batches in one place cheaply, because they are close together. Finally, basing distances on average distances may

even out the distinctiveness of requests too much, making all requests 'about as compatible'; though intuitively one may want to use a minimum distance rather than an average distance, this leads to a loss of property 3 of metric functions as soon as requests may share a node. If any of these behaviours are deemed too undesirable, it may be interesting to study whether the algorithm in Section 6.5.2, or an appropriate adjustment of it, still works when the undesirable behaviours are modelled out at the possible cost of non-metricity.

6.5.2. Description of algorithm

Finally, the CC heuristic is described as Algorithm 5. Figure 6.8 illustrates the desired result of the algorithm and may serve as a visual aid when reading it.

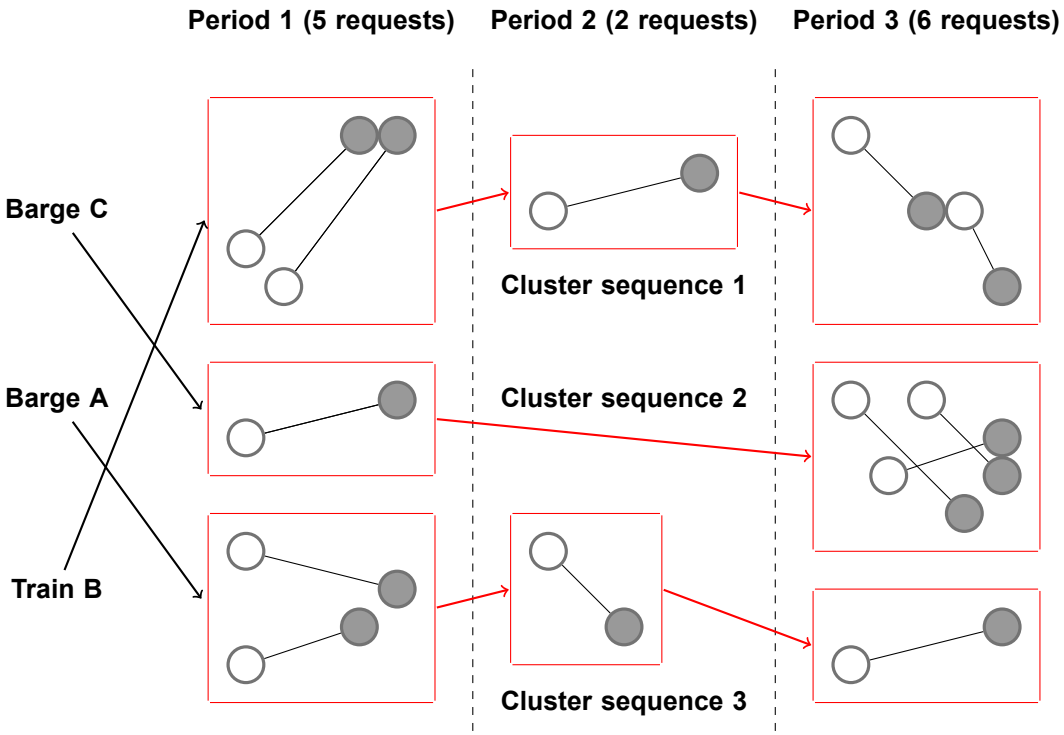


Figure 6.8: The desired result of the Compatibility Clustering heuristic: first, the requests are divided into a given number of time periods. Then, within each time period, the requests are divided in m clusters based on compatibility, with m the amount of vehicles; if the amount of requests in a time period is less than or equal to m, each request forms its own cluster, as here in Period 2. Each period then has at most m clusters: these are stringed together into cluster sequences, by looking at the spatial cluster distance between each cluster in one period and each cluster in the next, then finding a minimum weight perfect matching using the Hungarian algorithm. Finally, this yields m cluster sequences, and the m vehicles are assigned to the m cluster sequences, again with a minimum weight perfect matching: the assignment costs are determined by solving a sub-problem where the given vehicle executes the requests in the given cluster sequence.

Note that for its clustering, it uses the UPGMA method [42], because among clustering methods with non-Euclidean distances, it is known to produce clusters relatively equal in size [6], which is beneficial in dividing work load fairly over vehicles and thus preventing having to resort to infinite resources unnecessarily. Note furthermore that, though the metrics developed in Section 6.5.1 are defined vehicle-based, it is not yet known at some points in the algorithm which vehicle will service certain requests or clusters; at these points, the distance is averaged out over vehicles. In some real life instances, one may expect travel times to be similar for different vehicles, and if not, this is partially compensated in the final stages of the algorithm, where specific vehicles are taken into account. Further discussion on properties of this algorithm is postponed to Section 6.7.

Data: Instance of Problem 3 (set of vehicles W, set of locations I, set of requests K, list of time steps, amount of time periods Ξ; set Δ that forces fixed values to describe starting distribution and system properties; parameters as described in Section 6.1)
Result: Solution of Problem 3, possibly non-optimal
for *request k in K* **do**
  Interpret each separate batch of containers belonging to k as a 'separate request' κ;
end
Cluster all requests into Ξ time periods, using UPGMA with the $\tau(J_i, J_j)$ temporal distance between requests and place the clusters into a chronological list ;
for *cluster in list of time period clusters* **do**
  if *amount of requests in this time period* $\leq |W|$ **then**
    Make every request its own cluster;
  end
  else
    Cluster the requests within this time period into $|W|$ clusters, using UPGMA with the average (over vehicles) $d_w(J_i, J_j)$ space-time distance between requests ;
  end
end
Initialise $|W|$ empty cluster sequences;
Add each cluster in the first time period to its own cluster sequence ;
for *time period in* $\{2, \dots, \Xi\}$ **do**
  Compute each spatial cluster distance $\pi''(C_1, C_2)$ between the clusters at the end of the current cluster sequences and the clusters in this time period;
  if *amount of non-empty cluster sequences* $\geq$ *amount of clusters in this time period* **then**
    Assign each cluster in this time period to a cluster at the end of a non-empty cluster sequence, using the Hungarian algorithm ;
    Set each cluster in this time period at the end of the sequence to which it is assigned ;
  end
  else
    Assign the last cluster in each non-empty cluster sequence to a cluster in this time period, using the Hungarian algorithm ;
    Set each unassigned cluster in this time period at the end of a currently empty cluster sequence ;
    Set each assigned cluster in this time period at the end of the sequence to which it is assigned ;
  end
end
for *vehicle w in W* **do**
  for *cluster sequence in set of cluster sequences* **do**
    Compute the cost of assigning w to this cluster sequence by solving the sub-problem where only w handles only the requests in this cluster sequence, using the ILP and storing the solution;
  end
end
Use the Hungarian algorithm to assign vehicles to cluster sequences ;
Return the solution where each vehicle w and each request k does exactly what it does in the solved sub-problem if w is assigned to the cluster sequence that contains k.

Algorithm 5: The Compatibility Clustering heuristic: in each time period, cluster the requests into $|W|$ clusters according to their $d(J_1, J_2)$-distance, then make $|W|$ strings of clusters over the time periods by using the Hungarian algorithm between each time step and the next, then solve each sub-problem where only vehicle w handles only the requests in a given cluster sequence, then use the results to assign vehicles to cluster sequences using the Hungarian algorithm.

6.6. Numerical results

		Class A (2, 5, 4, 14)	Class B (3, 6, 5, 14)	Class C (4, 8, 8, 14)
ILP cost over optimal:	mean	0%	–	–
	variance	$0\%^2$	–	–
	worst	0%	–	–
ILP computation time:	mean	$67.9s$	–	–
	variance	$9549.6s^2$	–	–
	worst	$288.9s$	–	–
ILP+ cost over optimal:	mean	0%	0%	–
	variance	$0\%^2$	$0\%^2$	–
	worst	0%	0%	–
ILP+ computation time:	mean	$57.7s$	$26.2m$	–
	variance	$4197.8s^2$	$2149.1m^2$	–
	worst	$162.8s$	$2.56h$	–
GG cost over optimal:	mean	210.5%	152.5%	unknown
	variance	$72973\%^2$	$40821\%^2$	unknown
	worst	887.5%	508.3%	unknown
GG computation time:	mean	$15.4s$	$31.5s$	$113.5s$
	variance	$9.0s^2$	$14.8s^2$	$191.9s^2$
	worst	$22.0s$	$40.0s$	$140.2s$
CC cost over optimal:	mean	243.8%	249.4%	unknown
	variance	$150233\%^2$	$74553\%^2$	unknown
	worst	1200.0%	720.0%	unknown
CC computation time:	mean	$15.1s$	$28.8s$	$116.9s$
	variance	$14.0s^2$	$43.3s^2$	$349.0s^2$
	worst	$22.4s$	$40.7s$	$145.1s$

Table 6.1: Numerical results for random instances of Problem 3. Class A concerns instances with 2 vehicles, 5 locations, 4 requests and 14 time steps; the other classes can be interpreted the same way from their descriptions at the top of the table.

In this section, the various methods discussed in this chapter are tested for objective value efficiency and computation time. 'ILP' represents solving the instance to optimality with the ILP as given in Section 6.3.1, while 'ILP+' represents solving it with the additional constraints from Section 6.3.2. Though the hardware set-up is still the same as in Section 4.4, different class definitions are used: Class A consists of random problem 3 instances with 2 vehicles, 5 locations, 4 requests and 14 time steps, Class B has 3 vehicles, 6 locations, 5 requests and 14 time steps and Class C has 4 vehicles, 8 locations, 8 requests and 14 time steps. Unfortunately, no results are given for a class that represents 'real life size': this would involve 121 time steps rather than 14, and even the heuristics were unable to find a single solution in under five hours, which is sufficient to prove that these methods cannot yet be applied in on-line practice.

As problem classes become computationally more intense, certain solution methods are left out. In the first two classes, the exact optima are known, but not in Class C; here, it can only be stated that the Compatibility Clustering heuristic achieves results that have a cost of, on average, 129.0% over the cost attained with the Greedy Gain heuristic. Among these instances, none were found where CC found a cheaper solution than GG did.

The results can be viewed in Table 6.1 and are discussed in Section 6.7.

6.7. Discussion

From the results in Section 6.6, several things can be concluded.

1. Using the additional constraints from Section 6.3.2 leads to a significant decrease in ILP computation time;

2. Going from Class A to Class B, which is 'only slightly larger', increases the average ILP+ computation time by a factor 26;
3. Class B has an average ILP+ computation time of 26.2 minutes, with worst observed case 2.56 hours, making the ILP unsuitable for decision-support in cases of size B or up under the current hardware set-up;
4. On average, the GG heuristic achieves costs that are a factor 2.5 to 3.1 of the optimal cost, though this gap appears to become smaller as problem instances grow;
5. The computation time of both heuristics is much more stable and predictable than that of solving the ILP, and the growth of computation time is more manageable;
6. The computation times of the two heuristics are surprisingly similar, but the GG heuristic attains better results in cost;
7. Class C consists of instances that have an average computation time of two minutes for both heuristics. Depending on goals and preferences, this is 'pushing the limit' of how much computation time is allowed in decision-support;
8. The growth of computation time observed in all methods makes it clear that instances of the last class in previous chapters, consisting of 6 vehicles, 32 locations, 40 requests and 121 time steps, cannot yet be solved efficiently with these methods.

Conclusion 1 is probably due to the LP-relaxation solution polytopes being smaller, leading to better bounds in the ILP process. Conclusion 4, the non-optimality of the GG heuristic, is explained in Section 6.4. Conclusion 6, or rather that the GG heuristic performs better than the CC heuristic, is unfortunate, as the CC heuristic was designed to deal with the non-optimality of the GG heuristic. It should be investigated if other clustering methods and other distance metrics yield better results, perhaps even methods that discard metricity as suggested in Section 6.5.1, and studying optimal solutions may lead to other ideas of how to cluster, or other ideas for heuristics in general. Additionally, both heuristics assign a full request to at most one vehicle, which limits options of intermodality and eliminates options of consolidation in order to find a solution more quickly.

All other conclusions are about how the computation time grows too quickly for any of these methods to be useful in operational decision-support for problems of 'real life size'. This is easy to explain: the variables $b_{w,i,j,t}$, $q_{i,j,k,t}$ and $v_{i,j,k,t}$ and their corresponding inequalities grow enormously in amount, though these are only the worst offenders. Not only does this mean that the ILP solver has to handle thousands of variables and constraints, but also that the decision space is very large: the more time steps there are, the more random sequences of a vehicle visiting locations there are to investigate. More broadly, the solution set contains 'nonsensical' solutions like a vehicle picking up one container, going to some random location, dropping the container, picking it up again and repeating this until trucking it at the end, but the non-optimality of such strategies should be picked up very quickly in the branch-and-bound process; however, whether or not it is non-optimal to visit certain locations under certain conditions may require a deeper search tree. The heuristics solve sub-problems with only one vehicle w and a limited amount of requests, but even then, the amount of variables $b_{w,i,j,t}$ is still the amount of locations squared times the amount of time steps and the same or worse goes for $q_{i,j,k,t}$ and $v_{i,j,k,t}$.

Though speed-ups can probably be attained using commercial ILP solvers, parallelisation, cloud computing and streamlining of the implementations, this may cost a considerable amount of money before the current methods are useful in operational practice. Instead, there are still several mathematical innovations and refinements that are likely to reduce computation time:

- In the field of heuristics, develop heuristics that solve ILP sub-problems more rarely and carefully or not at all. This may not be a trivial task: the great benefit of solving

a sub-problem using the ILP is not so much the minimal cost, but that the ILP solver looks at the ILP and works out all details to make the solution feasible. When not using an ILP solver, much attention may have to be spent on ensuring solution feasibility;

- When working with the GG heuristic or other iterative heuristics, try an ILP solver that allows passing initial solutions: when checking the gain of adding request k to the schedule of vehicle w, it may be very useful to pass the current schedule of w and the current trucking decision for k as an initial solution;
- When solving sub-problems of one vehicle and one request, investigate ways to limit the amount of relevant locations and time steps as well, as these still make the sub-problems too large for 'real life size' instances;
- When solving the ILP, look into column generation. In a given solution, the amount of variables that have a value greater than 0 are comparatively small, because one only needs a handful of positive variables to describe the route of a vehicle or the used routes of a request. Applying column generation, however, is again not trivial, because the complex system dynamics may make it so that a variable cannot be added one at a time;
- Find lesser-indexed versions of variables. For example, it appears to be possible to replace variables $b_{w,i,j,t}$ (barge w departs from location i to location j at time t) with variables $b_{w,j,t}$ by splitting each equality

$$z_{w,i,t} = z_{w,i,t-1} + a_{w,i,t} - \sum_{j \in I} b_{w,i,j,t-1}$$

into four inequalities

$$a_{w,i,t} \leq z_{w,i,t} \leq a_{w,i,t} + 2z_{w,i,t-1}, \quad a_{w,i,t} + 2z_{w,i,t-1} - \sum_{j \in I} b_{w,j,t} - 1 \leq z_{w,i,t} \leq a_{w,i,t} - \sum_{j \in I} b_{w,j,t} + 1$$

and splitting each inequality $r_{w,t} \geq \sum_{i \in I} \sum_{j \in I} T_{w,i,j,t} b_{w,i,j,t}$ into $|I|^2$ inequalities $r_{w,t} \geq T_{w,i,j,t}(z_{w,i,t} + b_{w,j,t} - 1)$ and treating the contribution to the cost function the same, but with $E_{w,i,j,t}$;

- Similar to how Algorithm 1 eliminates redundant information in Problem 1, investigate how redundant information can be eliminated in Problem 3;
- When solving the ILP, see if it is beneficial to add more inequalities like the ones in Section 6.3.2.

Each of these are recommended for future research.

6.7.1. Added value

In this chapter, a problem was formulated that is a departure from many classical VRP problems, despite it having direct applications in practice. Its strong NP-hardness was proven, an ILP was formulated to solve it and two heuristics were proposed. Though the high computation time of these methods make them only applicable to small instances without resorting to more costly hardware configurations, clear venues were set out for how to find solutions more quickly through mathematical innovation.

6.8. Conclusion

This chapter sought to answer the following research sub-question:

4. How can a low cost net-centric operational transport schedule be found fast enough for on-line use if everything is known beforehand?

Deterministic operational freight planning, or 'Problem 3', was defined to be the problem of making the time tables for vehicles for travelling to locations and handling goods at terminals and simultaneously assigning containers to vehicles and infinite resources, all in such a way that the total cost of getting the containers to their destinations is minimal.

Though one would expect Problem 3 to resemble VRP problems, it is actually quite a departure from them, in that goods may switch vehicles any amount of times and vehicles do not have to start or end at a depot. Problem 3 was proven to still be strongly NP-hard, thus to not have an FPTAS.

An ILP formulation of this problem was developed, complete with the options to specify what the vehicles are doing at time 0 and to disallow certain vehicles to arrive at certain locations at certain times, so as to model opening times, terminal time slot availabilities and whether a certain terminal can even handle a certain modality. It was made slightly faster by adding additional inequalities.

Because the amount of variables in this ILP is polynomial but high in the amount of vehicles, locations, requests and time steps, a Greedy Gain heuristic was developed which solves sub-problems of only one vehicle and one request to see how much gain there is to be had in adding this request to the schedule of this vehicle, then iteratively adding requests to vehicle schedules based on the highest immediate gain. To counteract the obvious non-optimality this strategy may attain, a Compatibility Clustering heuristic was developed, which first clusters requests on how efficient it is to handle them together, then checks for each vehicle how much it would cost to handle this cluster by solving the corresponding sub-problem, then uses the Hungarian algorithm to assign vehicles to clusters.

Though the computation time of the CC heuristic is surprisingly similar to that of the GG heuristic and both are much faster than solving the complete ILP, the CC heuristic finds worse solutions than the GG heuristic, and none of the methods can find a solution fast enough for operational decision-support in instances of 'real life size' on modest hardware, because even the sub-problems are too large for those instances. Both heuristics find solutions with significantly non-optimal costs, possibly because they sacrifice some intermodality and all consolidation for speed, but the gap appears to become smaller for the GG heuristic as instances grow. A number of recommendations were given to find a solution more quickly: most importantly, a heuristic may have to be developed that does not solve sub-problems with the ILP at all.

To answer the sub-question: small instances may be solved to optimality fast enough and slightly larger instances may be solved with the Greedy Gain heuristic. For instances of 'real life size', however, more research will have to be done to find solutions fast enough for operational decision support, and a number of venues for this were formulated.

7

Conclusion

The following research question was posed:

How can on-line container-to-transport assignment and operational transport scheduling in synchromodal freight transport be optimised against their corresponding definition of 'optimal'?

This was answered along the following sub-questions:

1. How can different scenarios of synchromodal freight transport be classified within an exhaustive framework?

2. How can a low cost net-centric container-to-transport assignment be found fast enough for on-line use if everything is known beforehand?

3. How can a low cost net-centric container-to-transport assignment be found fast enough for on-line use if new data is still expected to come in?

4. How can a low cost net-centric operational transport schedule be found fast enough for on-line use if everything is known beforehand?

The provided answers are as follows:

1. Chapter 3 provides the framework asked for in the first sub-question. Though classification within this framework is still partly subject to subjectivity, it provides a formulation that captures many problem elements and whether they are controlled, fixed, stochastic, dynamic or irrelevant and also how the interaction between the decision-maker and the rest of the system is modelled.

2. A low cost net-centric container-to-transport assignment can be found by modelling the $\bar{R}|\bar{D},[D2R]||social(1)$-problem as a minimum cost multi-commodity flow problem on a space-time network. When removing redundant information using Algorithm 1, this method finds the optimal assignment for instances of 'real life size' in an average 9 seconds on modest hardware. The developed method allows for modelling of simultaneous soft due dates and hard deadlines.

3. Though it is theoretically possible to find an assignment policy with minimum expected cost for the $\widehat{R}|\widehat{D},[D2R]||social(1)$-problem using multi-stage stochastic programming, this is not possible in practice because of the enormous amount of computation it would require. Instead, Expected Future Iteration and 70%-Pessimistic Future Iteration are proposed: though they achieve near optimality in an average 24 seconds respectively 26 seconds per use, it is still unknown if they work as well under less uniform system dynamics and full use case data may be needed to examine this.

4. For small instances, a minimum cost net-centric operational schedule for the $\bar{R}, [RD], [RDT] | \bar{D}, [D2R] | social(1)$-problem, in the form of vehicle schedules and container-to-mode assignments, can be found with integer linear programming. For slightly larger instances, the computation time might become too high for integer linear programming, but solutions may still be found using the Greedy Gain heuristic or Compatibility Clustering heuristic, preferably the former. Though these methods could also be used to solve instances of 'real life size', additional research or hardware investments will have to be done to find a solution fast enough for operational decision support.

The added value of this thesis, more in-depth conclusions and remaining challenges can be found in the final sections of Chapters 3, 4, 5 and 6.

7.1. Recommendations for future research

This thesis ends with a number of recommendations for future research, divided on which problem or problems they relate to.

Problem 1:

- Refine Algorithm 1 to remove the redundancies it still leaves, or find some other minimal representation;
- Find solutions for this problem more quickly, possibly using heuristics, so that the simulation method used by Kooiman [35] can be applied in Problem 2;
- Investigate if combining shortest path algorithms with Lagrange relaxation on the capacity constraints can give input to heuristics or improvements to finding exact solutions.

Problem 2:

- Study problems where just the travel times, request release times or terminal handling times are stochastic: this may lead to more specialised versions of the synchromodal algorithms proposed here, and the large amount of attention these problems receive in literature justify the existence of specialised algorithms;
- Make single future iteration heuristics more suitable to rolling time windows, by quantifying if certain containers should be transported earlier than optimal to reserve capacity near the end of the time window for new requests;
- Apply the simulation method used by Kooiman, given that enough speed-up was achieved in solving or approximating instances of Problem 1;
- Redefine the 70%-Pessimistic Future Iteration method to be based not on separate percentiles, but on actual probabilities that the chosen path turns out to exist, or more accurately quantify this probability for the current method. This will move the method more clearly into the field of robustness;
- Further develop the proposed decision processes to deal well with dependent random variables.

Problem 3:

- Incorporate labour conditions by planning rest periods as well, either with virtual 'requests' at the rest stop or by adding appropriate variables and constraints;
- Speed up the ILP with column generation, clever reformulations, powerful additional constraints and removal of redundant options;
- Develop heuristics that find decent solutions of Problem 3 fast enough for operational decision-support for instances of 'real life size' as well;
- Find a way to solve sub-problems of one vehicle and one request without using the ILP, for iterative heuristics like the GG heuristic;

- Find a way to solve sub-problems of one vehicle without using the ILP, for heuristics like the GG heuristic and the CC heuristic;
- Investigate what metric properties are truly necessary for clustering and use this knowledge to improve the CC heuristic;
- Add components of intermodality and consolidation to existing or new heuristics.

'Problem 4':

- Investigate the stochastic version of Problem 3, given that enough advancements were made in Problem 2 and Problem 3 or that enough supporting literature already exists.

All problems:

- Test the methods against full data use cases and iterate on the results;
- Investigate the alternatives in meta-heuristics, where the rigorous problem definitions from this thesis may serve as input to define neighbourhoods.

A

Amount of distributions

Theorem 5. *Let $a \in \{0, 1, 2, ...\}$ and $b \in \{1, 2, ...\}$. The amount of ways to distribute a identical items over b recipients equals* $\left(\begin{array}{c} a+b-1 \\ b-1 \end{array} \right) = \left(\begin{array}{c} a+b-1 \\ a \end{array} \right)$.

	$b=1$	$b=2$	$b=3$	$b=4$	$b=5$	⋯
$a=0$	1	1	1	1	1	⋯
$a=1$	1	2	3	4	⋱	
$a=2$	1	3	6	⋱		
$a=3$	1	4	⋱			
$a=4$	1	⋱				
⋮	⋮					

Figure A.1: The amount of distributions of a identical items over b recipients follows the behaviour seen in Pascal's Triangle.

Proof. First, note that if $a = 0$, then the amount of distributions is 1: namely, all of the recipients get nothing. Indeed, the amount of distributions is then

$$\left(\begin{array}{c} a+b-1 \\ b-1 \end{array} \right) = \left(\begin{array}{c} b-1 \\ b-1 \end{array} \right) = 1$$

Second, note that if $b = 1$, the amount of distributions is 1: namely, the one recipient gets all a items. Indeed, the amount of distributions is then

$$\left(\begin{array}{c} a+b-1 \\ a \end{array} \right) = \left(\begin{array}{c} a \\ a \end{array} \right) = 1$$

Now suppose $a \neq 0$, $b \neq 1$, so $a > 0$, $b > 1$. Denote C the amount of distributions of $a-1$ items over b recipients. Denote D the amount of distributions of a items over $b-1$ recipients. It will be proven here that the amount of distributions of a items over b recipients equals $C + D$.

To this end, observe the distributions of a items over $b-1$ recipients. How many distributions are

there when another recipient is added? In each distribution, this new recipient either gets no items, or gets at least one item. The amount of distributions in which it gets none of the items is the amount of distributions of a items over $b-1$ recipients, so D. If the new recipient gets at least one item, the other $a-1$ items must still be distributed over the b recipients, which can be done in C ways. So the amount of distributions of a items over b recipients equals $C+D$.

Observe Figure A.1. It is well-known from the theory of Pascal's Triangle that this two-dimensional recursion, with the given base case, implies that the amount of distributions of a items over b recipients equals $\begin{pmatrix} a+b-1 \\ b-1 \end{pmatrix} = \begin{pmatrix} a+b-1 \\ a \end{pmatrix}$.

□

Bibliography

[1] Synchromodale cool port control. http://www.synchromodaliteit.nl/case/synchromodale-cool-port-control/. Accessed: 2017-02-27.

[2] Ontwikkeling van een synchromodale planningstool. http://www.synchromodaliteit.nl/case/ontwikkeling-van-een-synchromodale-planningstool/. Accessed: 2017-01-04.

[3] Synchromodaal transport biedt kansen. http://www.nederlandlogistiek.com/toekomst/synchromodaal-transport-biedt-volop-kansen-voor-logistieke-dienstverleners. Accessed: 2017-01-04.

[4] Lean and green synchromodal. http://www.synchromodaliteit.nl/case/lean-and-green-barge/. Accessed: 2017-02-27.

[5] Complexity in transport and logistics. http://www.nwo.nl/financiering/onze-financieringsinstrumenten/magw/complexity-in-transport-and-logistics/complexity-in-transport-and-logistics.html. Accessed: 2017-01-04.

[6] Clustering methods in python. http://scikit-learn.org/stable/modules/clustering.html. Accessed: 2017-07-29.

[7] Rotterdam – moerdijk – tilburg; een pilot met synchromodaal vervoer. http://www.synchromodaliteit.nl/case/rotterdam-moerdijk-tilburg-een-pilot-met-synchromodaal-vervoer/. Accessed: 2017-02-27.

[8] Synchromodal control tower. http://www.synchromodaliteit.nl/case/synchromodale-control-tower/. Accessed: 2017-02-27.

[9] Synchro-gaming: spelen met containers. http://www.synchromodaliteit.nl/synchro-gaming-spelen-containers/, . Accessed: 2017-01-04.

[10] Synchromodaliteit ervaren met een logistieke serious game. https://www.tno.nl/nl/over-tno/nieuws/2015/8/synchromodaliteit-ervaren-met-een-logistieke-serious-game/, . Accessed: 2017-01-04.

[11] Synchromodaily. http://www.synchromodaliteit.nl/case/synchromodaily/, . Accessed: 2017-02-27.

[12] Jardar Andersen, Teodor Gabriel Crainic, and Marielle Christiansen. Service network design with management and coordination of multiple fleets. *European Journal of Operational Research*, 193(2):377–389, 2009.

[13] Aristotle Arapostathis, Vivek S Borkar, Emmanuel Fernández-Gaucherand, Mrinal K Ghosh, and Steven I Marcus. Discrete-time controlled markov processes with average cost criterion: a survey. *SIAM Journal on Control and Optimization*, 31(2):282–344, 1993.

[14] William J Baumol. Economic theory and operations analysis. 1977.

[15] Behzad Behdani, Yun Fan, Bart Wiegmans, and Rob Zuidwijk. Multimodal schedule design for synchromodal freight transport systems. 2014.

[16] Tolga Bektas and Teodor Crainic. *A brief overview of intermodal transportation*. CIRRELT, 2007.

[17] Thorsten Blecker, Wolfgang Kersten, and Carsten Gertz. Management in logistics networks and nodes, 2008.

[18] Gerald G Brown and Robert F Dell. Formulating integer linear programs: A rogues' gallery. *INFORMS Transactions on Education*, 7(2):153–159, 2007.

[19] An Caris, Cathy Macharis, and Gerrit K Janssens. Planning problems in intermodal freight transport: accomplishments and prospects. *Transportation Planning and Technology*, 31(3):277–302, 2008.

[20] John W Chinneck. Practical optimization: a gentle introduction. *Systems and Computer Engineering), Carleton University, Ottawa. http://www. sce. carleton. ca/faculty/chinneck/po. html*, 2006.

[21] Jean-François Cordeau and Gilbert Laporte. The dial-a-ride problem (darp): Variants, modeling issues and algorithms. *4OR: A Quarterly Journal of Operations Research*, 1(2): 89–101, 2003.

[22] Teodor Gabriel Crainic and Kap Hwan Kim. Intermodal transportation. *Handbooks in operations research and management science*, 14:467–537, 2007.

[23] James D Dana Jr and Nicholas C Petruzzi. Note: The newsvendor model with endogenous demand. *Management Science*, 47(11):1488–1497, 2001.

[24] Bolin Ding, Jeffrey Xu Yu, and Lu Qin. Finding time-dependent shortest paths over large graphs. In *Proceedings of the 11th international conference on Extending database technology: Advances in database technology*, pages 205–216. ACM, 2008.

[25] Shimon Even, Alon Itai, and Adi Shamir. On the complexity of time table and multicommodity flow problems. In *Foundations of Computer Science, 1975., 16th Annual Symposium on*, pages 184–193. IEEE, 1975.

[26] Osamu Fujita. Metrics based on average distance between sets. *Japan Journal of Industrial and Applied Mathematics*, 30(1):1–19, 2013.

[27] Michael R Garey and David S Johnson. "strong"np-completeness results: Motivation, examples, and implications. *Journal of the ACM (JACM)*, 25(3):499–508, 1978.

[28] R.L. Graham, E.L. Lawler, J.K. Lenstra, and A.H.G. Rinnooy Kan. Optimization and approximation in deterministic sequencing and scheduling: a survey. *Annals of discrete mathematics*, 5:287–326, 1979.

[29] Marcia P Helme. Reducing air traffic delay in a space-time network. In *Systems, Man and Cybernetics, 1992., IEEE International Conference on*, pages 236–242. IEEE, 1992.

[30] Alan J Hoffman and Joseph B Kruskal. Integral boundary points of convex polyhedra. In *50 Years of Integer Programming 1958-2008*, pages 49–76. Springer, 2010.

[31] D. Huizing and M. Ortega del Vecchyo. The current worldwide state of synchromodal transport in academics and practice. 2016.

[32] LCM Kallenberg. Besliskunde 4. *Universiteit Leiden*.

[33] D.G. Kendall. Stochastic processes occurring in the theory of queues and their analysis by the method of the imbedded markov chain. *The Annals of Mathematical Statistics*, pages 338–354, 1953.

[34] Jeff L Kennington. A survey of linear cost multicommodity network flows. *Operations Research*, 26(2):209–236, 1978.

[35] K. Kooiman. A classification framework for time stamp stochastic assignment problems and an application to inland container shipping. 2015.

[36] L. et al. Li. Distributed model predictive control for cooperative synchromodal freight transport. *Transport. Res. Part E*, 2016.

[37] Le Li. Coordinated model predictive control of synchromodal freight transport systems. 2016.

[38] Xiao Lin, Rudy R Negenborn, and Gabriël Lodewijks. Towards quality-aware control of perishable goods in synchromodal transport networks. *IFAC-PapersOnLine*, 49(16): 132–137, 2016.

[39] Arnt-Gunnar Lium, Teodor Gabriel Crainic, and Stein W Wallace. A study of demand stochasticity in service network design. *Transportation Science*, 43(2):144–157, 2009.

[40] Martijn RK Mes and Maria-Eugenia Iacob. Synchromodal transport planning at a logistics service provider. In *Logistics and Supply Chain Innovation*, pages 23–36. Springer, 2016.

[41] R Garey Michael and S Johnson David. Computers and intractability: a guide to the theory of np-completeness. *WH Free. Co., San Fr*, pages 90–91, 1979.

[42] Charles D Michener and Robert R Sokal. A quantitative approach to a problem in classification. *Evolution*, 11(2):130–162, 1957.

[43] Asuman E Ozdaglar and Dimitri P Bertsekas. Optimal solution of integer multicommodity flow problems with application in optical networks. In *Frontiers in global optimization*, pages 411–435. Springer, 2004.

[44] Michael Berliner Pedersen, Oli BG Madsen, and Otto Anker Nielsen. *Optimization models and solution methods for intermodal transportation*. PhD thesis, Technical University of DenmarkDanmarks Tekniske Universitet, Department of TransportInstitut for Transport, Traffic ModellingTrafikmodeller, 2005.

[45] Arturo Pérez Rivera and Martijn Mes. Service and transfer selection for freights in a synchromodal network. *Lecture notes in computer science*, 9855:227–242, 2016.

[46] S. et. al. Pfoser. Critical success factors of synchromodality: results from a case study and literature overview. 2016.

[47] Martin L Puterman. *Markov decision processes: discrete stochastic dynamic programming*. John Wiley & Sons, 2014.

[48] CDC Reeve. Plato, cratylus. *Indianapolis/Cambridge*, 1998.

[49] Vasco Reis. Should we keep on renaming a+ 35-year-old baby? *Journal of Transport Geography*, 46:173–179, 2015.

[50] B Reissen and R. et. al. Negenborn. Service network design for an intermodal container network with flexible due dates/times and the possibility of using subcontracted transport. *Int. J. of Shipping and Transport Logistics*, 2013.

[51] B Reissen and R. et. al. Negenborn. Synchromodal container transportation: An overview of current topics and research opportunities. *Computational Logistics*, 2015.

[52] Marius M Solomon. Algorithms for the vehicle routing and scheduling problems with time window constraints. *Operations research*, 35(2):254–265, 1987.

[53] M SteadieSeifi, Nico P Dellaert, W Nuijten, Tom Van Woensel, and R Raoufi. Multimodal freight transportation planning: A literature review. *European journal of operational research*, 233(1):1–15, 2014.

[54] Agachai Sumalee, Kenetsu Uchida, and William HK Lam. Stochastic multi-modal transport network under demand uncertainties and adverse weather condition. *Transportation Research Part C: Emerging Technologies*, 19(2):338–350, 2011.

[55] Éric Taillard, Philippe Badeau, Michel Gendreau, François Guertin, and Jean-Yves Potvin. A tabu search heuristic for the vehicle routing problem with soft time windows. *Transportation science*, 31(2):170–186, 1997.

[56] Christophe Theys, Wout Dullaert, and Theo Notteboom. Analyzing cooperative networks in intermodal transportation: a game-theoretic approach. In *Nectar Logistics and Freight Cluster Meeting, Delft, The Netherlands*, 2008.

[57] B van Riessen. Impact and relevance of transit disturbances on planning in intermodal container networks using disturbance cost analysis.

[58] Bart van Riessen, Rudy R Negenborn, and Rommert Dekker. Synchromodal container transportation: An overview of current topics and research opportunities. In *International Conference on Computational Logistics*, pages 386–397. Springer, 2015.

[59] PJ Vinke. Dynamic consolidation decisions in a synchromodal environment: Improving the synchromodal control tower. 2016.

[60] Douglas J White. A survey of applications of markov decision processes. *Journal of the Operational Research Society*, 44(11):1073–1096, 1993.

[61] Ya Xu, Qiushuang Chen, and Xiongwen Quan. Robust berth scheduling with uncertain vessel delay and handling time. *Annals of Operations Research*, 192(1):123–140, 2012.

[62] M Zhang and AJ Pel. Synchromodal hinterland freight transport: model study for the port of rotterdam. *Journal of Transport Geography*, 52:1–10, 2016.