



NTNU – Trondheim
Norwegian University of
Science and Technology

Optimization of combined ship routing and inventory management in the salmon farming industry

Kristine S. Ivarsøy
Ida Elise Solhaug

Industrial Economics and Technology Management

Submission date: June 2014

Supervisor: Asgeir Tomasgard, IØT

Co-supervisor: Marielle Christiansen, IØT

Norwegian University of Science and Technology
Department of Industrial Economics and Technology Management

MASTERKONTRAKT

- uttak av masteroppgave

1. Studentens personalia

Etternavn, fornavn Ivarsøy, Kristine S.	Fødselsdato 05. jul 1989
E-post krisiva@stud.ntnu.no	Telefon 40401489

2. Studieopplysninger

Fakultet Fakultet for samfunnsvitenskap og teknologiledelse	
Institutt Institutt for industriell økonomi og teknologiledelse	
Studieprogram Industriell økonomi og teknologiledelse	Hovedprofil Anvendt økonomi og optimering

3. Masteroppgave

Oppstartsdato 15. jan 2013	Innleveringsfrist 11. jun 2013
Oppgavens (foreløpige) tittel Optimization of combined ship routing and inventory management in the salmon farming industry	
Oppgavetekst/Problembeskrivelse The purpose of this master thesis is to develop optimization models and methods which can support Marine Harvest Norway in their planning of fish feed transportation. Problems within an operational and tactical planning horizon will be considered taking ship routing and scheduling as well as inventory management at the fish farms into account. The formulation should be realistic, taking into consideration the most important restrictions. The focus is on exact solution methods. Due to the complexity of the problem, the emphasis will be on achieving near optimal solutions. Main contents: - Description of the problem - Presentation and discussion of relevant literature and solution methods - Development of mathematical models of the underlying problem suitable solution methods - Implementation of mathematical models and methods using commercial software - Testing of models and methods with relevant data provided by Marine Harvest - Discussion of the results and the usefulness of the model(s) and method(s).	
Hovedveileder ved institutt Professor Asgeir Tomasgard	Medveileder(e) ved institutt Marielle Christiansen
Merknader 1 uke ekstra p.g.a påske.	

4. Underskrift

Student: Jeg erklærer herved at jeg har satt meg inn i gjeldende bestemmelser for mastergradsstudiet og at jeg oppfyller kravene for adgang til å påbegynne oppgaven, herunder eventuelle praksiskrav.

Partene er gjort kjent med avtalens vilkår, samt kapitlene i studiehandboken om generelle regler og aktuell studieplan for masterstudiet.

Trondheim 28.04.14

Sted og dato

Kristine S. Larsøy
Student


Hovedveileder

MASTERKONTRAKT

- uttak av masteroppgave

1. Studentens personalia

Etternavn, fornavn Solhaug, Ida Elise	Fødselsdato 25. mai 1990
E-post idaeliso@stud.ntnu.no	Telefon 95148166

2. Studieopplysninger

Fakultet Fakultet for samfunnsvitenskap og teknologiledelse	
Institutt Institutt for industriell økonomi og teknologiledelse	
Studieprogram Industriell økonomi og teknologiledelse	Hovedprofil Anvendt økonomi og optimering

3. Masteroppgave

Oppstartsdato 15. jan 2013	Innleveringsfrist 11. jun 2013
Oppgavens (foreløpige) tittel Optimization of combined ship routing and inventory management in the salmon farming industry	
Oppgavetekst/Problembeskrivelse The purpose of this master thesis is to develop optimization models and methods which can support Marine Harvest Norway in their planning of fish feed transportation. Problems within an operational and tactical planning horizon will be considered taking ship routing and scheduling as well as inventory management at the fish farms into account. The formulation should be realistic, taking into consideration the most important restrictions. The focus is on exact solution methods. Due to the complexity of the problem, the emphasis will be on achieving near optimal solutions. Main contents: - Description of the problem - Presentation and discussion of relevant literature and solution methods - Development of mathematical models of the underlying problem suitable solution methods - Implementation of mathematical models and methods using commercial software - Testing of models and methods with relevant data provided by Marine Harvest - Discussion of the results and the usefulness of the model(s) and method(s).	
Hovedveileder ved institutt Professor Asgeir Tomasgard	Medveileder(e) ved institutt Marielle Christiansen
Merknader 1 uke ekstra p.g.a påske.	

4. Underskrift

Student: Jeg erklærer herved at jeg har satt meg inn i gjeldende bestemmelser for mastergradsstudiet og at jeg oppfyller kravene for adgang til å påbegynne oppgaven, herunder eventuelle praksiskrav.

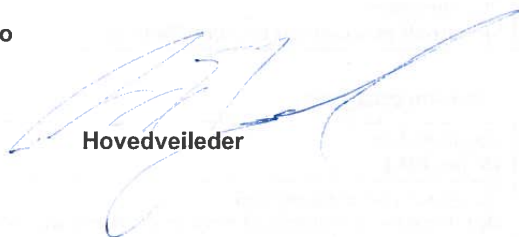
Partene er gjort kjent med avtalens vilkår, samt kapitlene i studiehåndboken om generelle regler og aktuell studieplan for masterstudiet.

28/4-14 Trondheim

Sted og dato

Ida Elise Solhaug

Student



Hovedveileder

SAMARBEIDSKONTRAKT

1. Studenter i samarbeidsgruppen

Etternavn, fornavn Ivarsøy, Kristine S.	Fødselsdato 05. jul 1989
Etternavn, fornavn Solhaug, Ida Elise	Fødselsdato 25. mai 1990

2. Hovedveileder

Etternavn, fornavn Tomasgard, Asgeir	Institutt Institutt for industriell økonomi og teknologiledelse
--	---

3. Masteroppgave

Oppgavens (foreløpige) tittel Optimization of combined ship routing and inventory management in the salmon farming industry

4. Bedømmelse

Kandidatene skal ha *individuell* bedømmelse
Kandidatene skal ha *felles* bedømmelse

<input type="checkbox"/>
<input checked="" type="checkbox"/>

Trondheim 28.04.14

Sted og dato



Hovedveileder



Kristine S. Ivarsøy



Ida Elise Solhaug

Originalen oppbevares på instituttet.

Preface

This master thesis is the last part of our Master of Science degree at the Norwegian University of Science and Technology. The degree specialization is Applied Economics and Optimization at the Department of Industrial Economics and Technology Management. The thesis was written in the spring of 2014, in collaboration with the feed division of Marine Harvest Norway.

We would like to thank all of those who have given of their time and expertise to help us during our work with this thesis. From Marine Harvest, we would like to thank Espen Moxnes, Vidar Myhre, Nora Hindar, and Eivind Osnes for explaining to us the salmon farming industry and the challenges related to planning of feed delivery. The meetings and email correspondence have been of great help to us. We would also like to thank Océane Balland, Associate Professor II at the Department of Marine Technology, NTNU, for the help with calculating LNG consumption and prices.

Our academic supervisors, Professor Marielle Christiansen and Professor Asgeir Tomasgard, have given us excellent guidance and feedback. A great thank you goes to Gerardo Perez Valdes, whom without, our work on parallelization would not have been completed. Also, we would like to thank Professor Agostinho Agra, for taking such genuine interest in our work and for his suggestions on how to improve our models. We are also grateful for the help with Mosel programming we have received from Associate Professor Henrik Andersson and for the help from Professor Kjetil Fagerholt on modeling maritime inventory routing problems.

Trondheim, June 6th, 2014

Kristine Stautland Ivarsøy

Ida Elise Solhaug

Sammendrag

Fiskefôr utgjør den største andelen av produksjonskostnader for en lakseoppdretter og tapte fôrdager er en stor kostnadsdriver. Norges største lakseoppdretter, Marine Harvest Norway, ønsker å kutte kostnader og redusere risikoen for sene fôrleveranser ved å begynne med egen fôrproduksjon. For å levere fôr til oppdrettsanleggene på en effektiv og pålitelig måte, trenger de kostnadseffektive og robuste transportplaner.

Denne masteroppgaven baserer seg på Marine Harvest Norway, heretter kalt Marine Harvest, sitt planleggingsproblem for fôrleveranser. Målet er å lage en modell som håndterer både lagerstyring på fabrikk og oppdrettsanlegg, samt ruteplanlegging for fôrleveranser. I Ivarsøy og Solhaug (2013) utviklet vi en modell for å løse et kombinert lagerstyrings- og ruteplanleggingsproblem, også kalt et Inventory Routing Problem (IRP). Her har vi videreutviklet modellen til tre nye matematiske formuleringer. Vi har også utviklet to rammeverk for parallell branch-and-bound i et forsøk på å søke gjennom større deler av løsningsrommet på kortere tid.

De tre nye formuleringene har blitt testet på tre datasett, der ulikt antall oppdrettsanlegg er inkludert. Det største datasettet er en realistisk representasjon av planleggingsproblemet. De fleste parameterne som er brukt er reelle data fra Marine Harvest, mens estimer har blitt gjort når det var nødvendig.

Den første formuleringen er en arc-load modell. De to andre formuleringene, arc-flow og multi-commodity flow, er en utvidelse av denne og inkluderer mer detaljert informasjon om last ombord på skip. Resultatene viser at arc-flow modellen presterer best for alle tre datasett. Arc-load modellen presterer bra for de to minste datasettene. Multi-commodity flow modellen skalerer dårlig og bruker lang tid på å løse hver branch-and-bound node.

De parallelle rammeverkene fungerte ikke helt som vi hadde håpet. Det første rammeverket paralleliserer arbeidet med å løse hver branch-and-bound node og klarte ikke å finne hverken gode nedre grenser eller heltallsløsninger. Det andre rammeverket paralleliserer arbeidet med å løse hele subtrær og virker mer lovende. Vår enkle synkroniserte implementasjon, fant en marginalt bedre løsning enn den sekvensielle kjøringen, men løsningstiden ble ikke redusert i forhold til økningen av antall prosessorer brukt.

Summary

Fish feed is the largest component of a salmon farmer's production costs and lost feed days are a large cost driver. Norway's largest salmon farmer, Marine Harvest Norway, hopes to cut costs and increase reliability of feed deliveries by starting in-house feed production and distribution. In order to deliver feed to fish farms from the new factory in an efficient and reliable way, they need to create cost-effective and robust transportation plans.

The basis for this thesis is the problem of planning feed deliveries faced by Marine Harvest Norway, hereby referred to as Marine Harvest. The goal is to provide a model to simultaneously aid in routing of feed deliveries and inventory management. We have further developed our Inventory Routing Problem (IRP) model from Ivarsøy and Solhaug (2013) into three mathematical formulations. We have also developed two frameworks for parallel branch-and-bound, in an attempt to search through a larger part of the solution space in shorter time.

The three formulations have been tested on three various-sized test cases, which differ in the number of fish farms considered, as well as production and transportation capacity. The largest test case is a realistic representation of the planning problem. Most of the parameters used are real data provided by Marine Harvest, while we have made our own estimations when needed.

The first formulation is an arc-load formulation, similar to the one given in Ivarsøy and Solhaug (2013). The arc-flow and multi-commodity formulations are extensions of this, and include more detailed information about loads on board ships. Results showed that the arc-flow model performed best for all test cases. The arc-load model performed well for the smallest test case, while the multi-commodity model suffers from long solution times for each branch-and-bound-node and has scaling issues due to a large number of variables.

The parallel frameworks did not work as well as hoped. The first framework parallelize the work of solving each branch-and-bound node, and did not manage to find any integer solutions or good lower bounds. The second framework is more promising, but our simple synchronous implementation only finds a marginally better solution than the sequential run, and the solution time was not decreased in accordance with the increased use of computing resources.

Contents

1	Introduction	1
2	Salmon Farming Industry	5
2.1	Industry Players	6
2.2	Production of Salmon	7
2.2.1	Freshwater production	8
2.2.2	Seawater production	8
2.3	Production of Fish Feed	8
2.4	Salmon Feeding	9
2.5	Vertical Integration in the Salmon Industry	10
3	Problem Description	13
3.1	Problem Background	13
3.2	Detailed Problem Description and Assumptions	16
3.3	Problem Summary	18
4	Related Literature	21
4.1	Maritime and Land-Based Transportation	22
4.2	Inventory Routing Problems	22
4.2.1	Planning horizon	23
4.2.2	Time	24
4.2.3	Inventory policy	24
4.2.4	Fleet of vehicles	24
4.2.5	Structure of distribution network	25
4.2.6	Multiple products	25
4.2.7	Nature of demand	25
4.2.8	Objective	25
4.3	Solution Methods	26
4.3.1	Model formulations	27
4.3.2	Exact methods	28
4.3.3	Heuristic methods	29
4.4	Parallel Algorithms	29
4.4.1	Aspects of parallel branch-and-bound	30
4.4.2	Applications	32

5	Model Formulations	35
5.1	Model Introduction	35
5.1.1	Common model characteristics	35
5.1.2	Model applications	36
5.2	Arc-Load Formulation	37
5.2.1	Definitions	38
5.2.2	Model formulation	40
5.3	Arc-Flow Formulation	48
5.3.1	Definitions	49
5.3.2	Model formulation	49
5.4	Multi-Commodity Flow Formulation	49
5.4.1	Definitions	51
5.4.2	Model formulation	51
6	Implementation	55
6.1	Test Cases	56
6.2	Model Adjustments	57
6.2.1	Linearizing constraints	57
6.2.2	Slack in equality constraints	58
6.3	Problem Reduction	58
6.3.1	Simplifications	58
6.3.2	Sets of fish farms	59
6.3.3	Reduced number of visits	59
6.3.4	Aspect of service hours removed	59
6.3.5	Elimination of variables	60
6.4	Attempts of Model Improvement	60
6.4.1	Special ordered sets	60
6.4.2	Tightening constraints	60
6.4.3	Valid inequalities	62
6.4.4	Aggregated variables for branching	64
6.4.5	Stricter constraints	64
6.5	Parallel Branch-and-Bound Search	65
6.5.1	Parallel work on nodes	66
6.5.2	Parallel work on subtrees	67
7	Computational study	71
7.1	Test Instances	71
7.2	Small-Sized Test Case	72
7.2.1	LP bound	73
7.2.2	SOS1	74
7.2.3	Tightened constraints and valid inequalities	75
7.3	Medium-Sized Test Case	79
7.3.1	LP bound	79
7.3.2	Tightened constraints and valid inequalities	80
7.4	Large-Sized Test Case	81
7.4.1	LP bound	82

7.4.2	Tightened constraints and valid inequalities	82
7.4.3	Effects of further improvements	83
7.5	Comparison of Formulations	86
7.6	Parallelization	86
7.6.1	Node-based parallelization	87
7.6.2	Tree-based parallelization	87
7.7	Economical Results	88
8	Concluding Remarks	93
9	Future Research	95
	Bibliography	97
A	Complete Arc-Load Formulation	103
A.1	Definitions	103
A.1.1	Sets	103
A.1.2	Indices	103
A.1.3	Parameters	104
A.1.4	Decision variables	104
A.2	Model Formulation	105
A.2.1	Objective function	105
A.2.2	Routing constraints	105
A.2.3	Loading and unloading constraints	105
A.2.4	Inventory constraints	106
A.2.5	Timing constraints	107
A.2.6	Variable constraints	108
B	Complete Arc-Flow Formulation	109
B.1	Definitions	109
B.1.1	Sets	109
B.1.2	Indices	109
B.1.3	Parameters	110
B.1.4	Decision variables	110
B.2	Model Formulation	111
B.2.1	Objective function	111
B.2.2	Routing constraints	111
B.2.3	Loading and unloading constraints	111
B.2.4	Inventory constraints	112
B.2.5	Timing constraints	113
B.2.6	Variable constraints	114
C	Complete Multi-Commodity Flow Formulation	115
C.1	Definitions	115
C.1.1	Sets	115
C.1.2	Indices	116

C.1.3	Parameters	116
C.1.4	Decision variables	117
C.2	Model Formulation	117
C.2.1	Objective function	117
C.2.2	Routing constraints	117
C.2.3	Loading and unloading constraints	118
C.2.4	Inventory constraints	119
C.2.5	Timing constraints	120
C.2.6	Variable constraints	121
D	Test Case Data	123
D.1	Common Data	123
D.2	Small Test Case	123
D.3	Medium Test Case	125
D.4	Large Test Case	127
E	Cost Calculations	131
E.1	Transportation Cost	131
E.1.1	LNG consumption	131
E.1.2	LNG prices	132
E.2	External Feed Costs	132
E.2.1	Fixed transportation cost	132
E.2.2	Unit cost	133
F	Parallel Frameworks	135
F.1	Node-Based Parallelization	135
F.1.1	<i>main</i>	135
F.1.2	<i>bbclass</i>	136
F.1.3	<i>workerclass</i>	144
F.1.4	<i>nodeclass</i>	148
F.2	Tree-Based Parallelization	150
F.2.1	<i>main</i>	150
F.2.2	<i>bbclass</i>	151
F.2.3	<i>workerclass</i>	155
G	Contents of Enclosed CD	161

List of Tables

2.1	External feed supply versus internal feed production.	11
4.1	Characteristics of problems from related research.	23
4.2	Formulations and solution methods in related research.	26
4.3	Characteristics of algorithms with low and high scalability.	32
6.1	Specifications of computers in Solstorm.	55
7.1	Notation for test instances.	72
7.2	Small test case: Problem size	73
7.3	Small test case: LP optima	73
7.4	Small test case: Results with and without SOS1	74
7.5	Small test case: Results with and without presolve	75
7.6	Small test case: Results for all formulations and instances	76
7.7	Small test case: Results from combinations off effective cuts	78
7.8	Medium test case: Problem size	79
7.9	Medium test case: LP optima	80
7.10	Medium test case: Results for all formulations and instances	80
7.11	Medium test case: Results from combinations off effective cuts	81
7.12	Large test case: Problem size	81
7.13	Large test case: LP optima	82
7.14	Large test case: Results for all instances	82
7.15	Large test case: Results for combinations of cuts	82
7.16	Large test case: Results of problem reduction	83
7.17	Large test case: Results with aggregated variables for branching	84
7.18	Large test case: Results with larger minimum unloading quantities	84
7.19	Large test case: Results from adding EOH constraints	85
7.20	Utilization of production for different replanning points.	91
7.21	Utilization of ship capacity for different replanning points.	91
D.1	Small test case: Production rate, storage capacity and initial stock for the factory.	124
D.2	Small test case: Capacity and initial load for each ship.	124
D.3	Small test case: Initial stock, demand, safety stock and maximum storage capacity for each fish farm	125

D.4	Medium test case: Production rate, storage capacity and initial stock for the factory	125
D.5	Medium test case: Capacity and initial load for each ship.	126
D.6	Medium test case: Initial stock, demand, safety stock and maximum storage capacity for each fish farm	127
D.7	Large test case: Production rate, storage capacity and initial stock for the factory.	127
D.8	Large test case: Capacity and initial load for each ship.	127
D.9	Large test case: Initial stock, demand, safety stock and maximum storage capacity for each fish farm	129
E.1	Delivered power and energy consumption for six meter draft and controllable pitch propeller	131
E.2	Consumption with controllable pitch propeller	132
E.3	Transportation costs	132
G.1	Contents of enclosed CD.	161

List of Figures

1.1	Sales volume and value of Norwegian farmed salmon, 2002-2012	1
2.1	Illustration of the farmed salmon value chain	5
2.2	Export price of farmed salmon, 2012-2014	6
2.3	Expected market share of salmon feed producers in 2013	7
2.4	Illustration of a salmon's life cycle	7
2.5	Components of production costs for fish farmers in Norway.	10
3.1	Marine Harvest's fish farms located along the Norwegian coast	14
3.2	Illustration of Marine Harvest's planning situation.	19
4.1	Illustration of a shared, distributed and hybrid data model	31
5.1	Illustration of the arc-load formulation.	38
5.2	Illustration of the arc-flow formulation.	48
5.3	Illustration of the multi-commodity flow formulation.	50
6.1	Illustration of a parallel branch-and-bound framework.	66
6.2	Classes implemented in the node-based parallel branch-and-bound framework.	67
6.3	Classes implemented in the tree-based parallel branch-and-bound framework	68
7.1	Illustration of how the solution and lower bound improves with time.	85
7.2	Illustration of the solutions found by the sequential and parallel run.	89
7.3	Illustration of the best solution found by the sequential run of the arc-flow instance TM.	90

Chapter 1

Introduction

Norway is the world's largest producer of farmed salmon and the industry is growing, as illustrated in Figure 1.1. In 2012 the production in Norway was 1.2 million metric tons, valued to approximately 28 billion NOK. Continued growth is, among other things, dependent on sustainable solutions to environmental challenges, change of regulations, increased demand and innovations in feed production (Olafsen et al., 2012). If this is achieved, salmon farming will become an increasingly important part of Norwegian exports in the future.

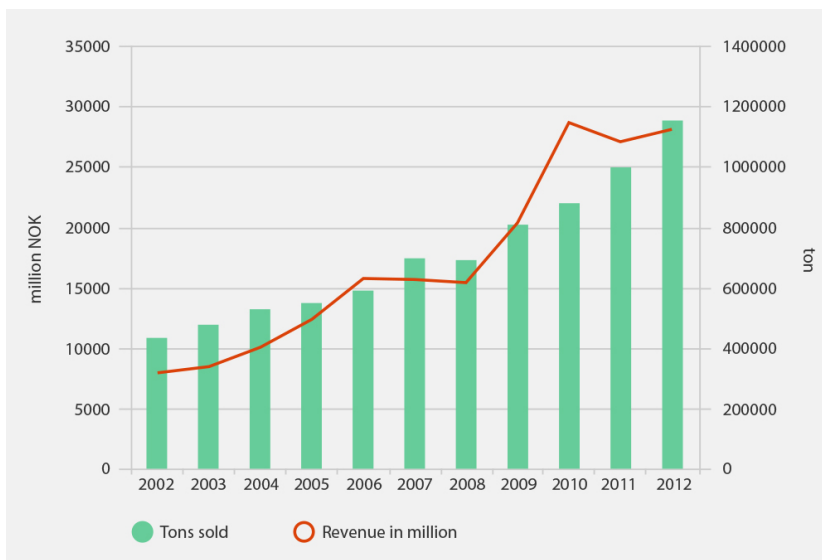


Figure 1.1: Sales volume and value of Norwegian farmed salmon, 2002-2012 (Statistisk sentralbyrå, 2013).

Marine Harvest is a vertically integrated salmon farmer with business ranging from egg production to processing, distribution and sales of finished products. Salmon farmers are not traditionally involved in feed production, but Marine Harvest is now aiming to integrate into both feed production and delivery. They will be the only salmon farmer in Norway controlling the entire value chain. This allows for implementation of vendor managed inventory (VMI), as opposed to the order-based feed delivery common in the industry. VMI is a popular business practice in supply chain management, and can give benefits such as lower inventory and transportation costs and reduced risk of empty inventories (Simchi-Levi et al., 1999, chap.8).

Costs related to feed account for approximately half of salmon production costs. Another large cost driver is lost feed days. Marine Harvest hopes to lower costs and secure stable deliveries of feed by gaining control over this critical part of the value chain. In Ivarsøy and Solhaug (2013), we developed two alternative models for the planning of feed deliveries and evaluated them on a reduced data set. The first model is based on an inventory routing problem (IRP) and is able to achieve all the flexibility of VMI by taking into account both routing and inventory management. The other is based on a periodic vehicle routing problem (PVRP) where weekly schedules with fixed delivery quantities are created. This gives predictable delivery dates, but much of the flexibility and possible gains from VMI are lost.

In this thesis we will continue our work on the IRP model, in accordance with Marine Harvest's decision to implement VMI and abandon order-based feed delivery. Our aim has been to develop a model which can support the process of planning feed deliveries to fish farms. The model minimizes transportation costs and avoids inventories below safety stock levels, resulting in a robust transportation plan. The model decides on volumes loaded at the factory and discharged at each fish farm, as well as the order of visits. Production capacity is not enough to supply all fish farms and as demands vary, the model gives an indication of how many fish farms it is realistic to serve at a given time of the year. The model is flexible to allow for future changes in Marine Harvest's planning situation, such as an increased number of factories, fish farms and ships.

As opposed to many IRPs from maritime industry, Marine Harvest's problem includes several consecutive deliveries and no tight time windows. Another complicating characteristic is that they do not have capacity to supply all fish farms internally. Solution methods that have proven to work well on other applications, may not be viable for this problem. We have focused on exact methods and experimented with different formulations and valid inequalities. In an effort to solve the real-sized problem, we have also combined the use of a commercial mixed integer programming (MIP) solver with frameworks for parallel branch-and-bound.

Firstly, we give an introduction to the farmed-salmon value chain in Chapter 2. In Chapter 3 we provide a detailed description of the transportation problem we have based our modeling on. Chapter 4 is a review of research done on the IRP, presenting important problem characteristics and solution methods along with

relevant papers. Here, we also give an introduction to work done on frameworks for parallel branch-and-bound. Chapters 5 and 6 presents our various model formulations and the implementation of these. Chapter 7 presents the results from running the models using three various-sized test cases, along with a comparison of the formulations. We also present results from testing the two parallel frameworks and explore the best solution found. Lastly, we provide concluding remarks in Chapter 8 and ideas for future research in Chapter 9.

Chapter 2

Salmon Farming Industry

This chapter gives relevant background information for the IRP model described in Chapter 5, starting with an introduction of the industry players in Section 2.1. In Section 2.2 we describe the production of salmon, while Sections 2.3 and 2.4 give an introduction to the production of salmon feed and feeding of salmon. Lastly, we provide thoughts on vertical integration of the farmed-salmon value chain in Section 2.5. We will focus on the production cycle of salmon and production of salmon feed, but the characteristics of the value chain apply to other species as well. The farmed-salmon value chain is illustrated in Figure 2.1.

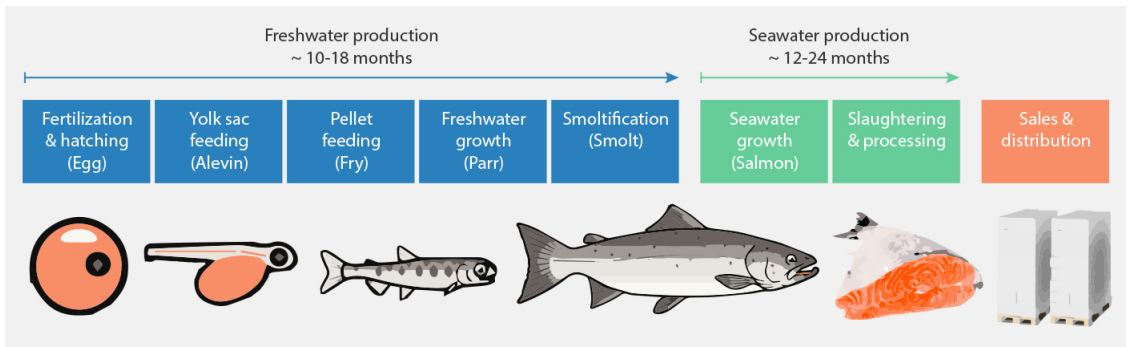


Figure 2.1: Illustration of the farmed salmon value chain (Marine Harvest, 2010).

2.1 Industry Players

Salmon farmers have experienced rapid growth since the industry developed in the 1970s. Salmon farming requires specific environmental conditions and the industry is therefore mainly located in Norway, Chile, North America and the UK, which all have suitable conditions. Historically, the fish farming industry had several small players, but in recent years there has been a consolidation. In Norway the industry is more fragmented than in other countries, since one company can not own more than 25% of the total licensed biomass and the government must approve license acquisitions resulting in control of more than 15%. The term biomass refers to the current standing fish mass at a farming facility or within a region. In Norway, Marine Harvest controls nearly 25%, while other large producers are Lerøy, Salmar, Cermaq and Grieg Seafood.

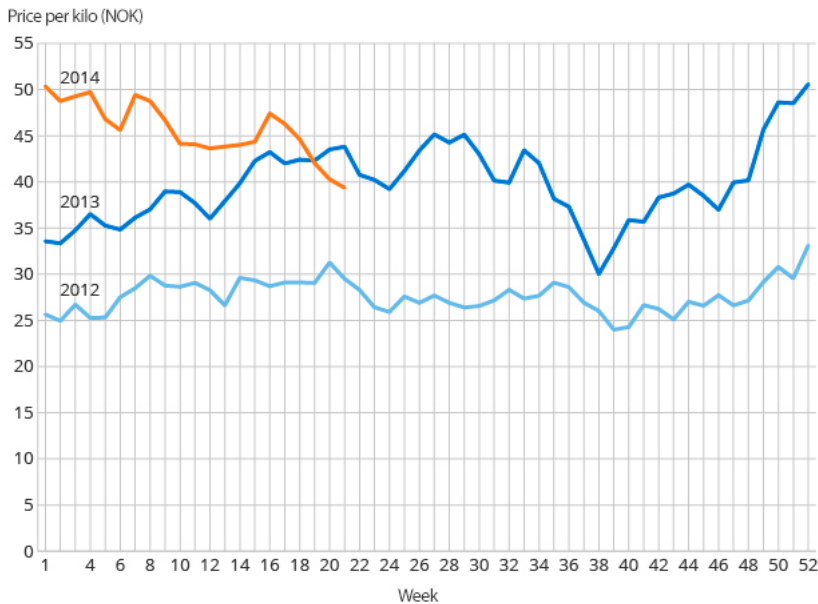


Figure 2.2: Export price of farmed salmon, 2012-2014
(Statistisk sentralbyrå, 2014).

Salmon prices have increased drastically since 2012, as illustrated in Figure 2.2, although there seems to have been a price correction in April and May 2014. Industry experts fear that salmon farming has reached its capacity and that prices are going up because of expected lower supply. Constraints on future production are diseases, environmental challenges and geographical limitations (Olafsen et al., 2012). One important constraint on supply is the reduced availability of marine resources such as raw material for fish feed. Therefore, feed producers are becoming

an increasingly important part of the farmed-salmon value chain. Simultaneously with the consolidation of the salmon farming industry, the salmon feed industry has become an oligopoly of three strong players (Marine Harvest, 2013b). The three feed producers EWOS, Skretting and BioMar control more than 97% of the global salmon feed market, as can be seen in Figure 2.3.

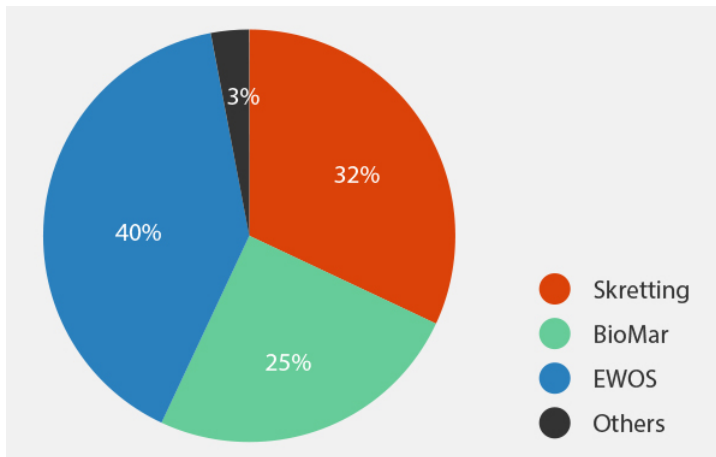


Figure 2.3: Expected market share of salmon feed producers in 2013 (Marine Harvest, 2013a).

2.2 Production of Salmon

As described in Marine Harvest (2013b), the salmon farming production cycle consists of two main phases, one in freshwater and one in seawater. The total production cycle lasts from 24 to 40 months. Figure 2.4 illustrates how the fish evolve from eggs to adult salmon.

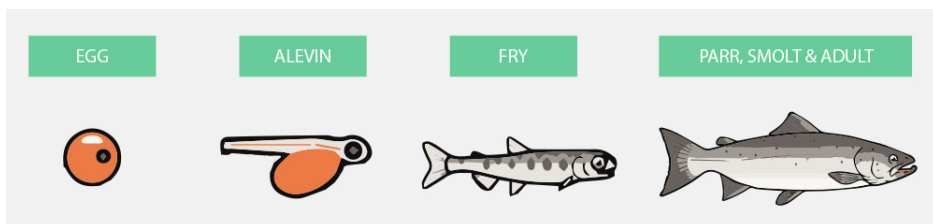


Figure 2.4: Illustration of a salmon's life cycle (Marine Harvest, 2010).

2.2.1 Freshwater production

After egg hatching, the salmon-to-be start off as alevins and feed is provided from an attached yolk sac. When the alevins have grown large enough to consume normal feed, they are referred to as fry. Fry grow into parr and move to larger freshwater tanks where it is possible to control light intensity and to some extent water temperature. Towards the end of the freshwater phase the parr go through a smoltification process, which makes the fish ready for transfer to seawater. Salmon farmers try to maintain a high utilization of the maximum allowable biomass set by government regulations. In order to maintain a steady production of salmon throughout the year, it is important to be able to transfer smolt to seawater more than once a year. This is done by having some fish reach the smoltification stage more quickly, by increasing light intensity and temperature in the tanks. This fish will be released during fall and have a smaller release size than the more slow-growing parr, which will become smolt and transferred to seawater the following spring.

2.2.2 Seawater production

The main growth phase takes place after the smolt is transferred to net pens in seawater and lasts for 14 to 24 months. Each fish farm usually has multiple net pens and can receive smolt in different batches without losing traceability. In seawater the salmon growth is strongly dependent on the sea temperature and in the range 8°C to 14°C, salmon growth increases with temperature. Therefore, the salmon along the Norwegian coastline grows faster during summer than winter. When reaching target weight, the salmon are harvested and transferred to a processing facility to be gutted and put on ice. The salmon are then sold directly to retailers or further processed into value-added products. Fish farms must be fallowed after each production cycle. This means that they are kept empty for two to six months, to prevent diseases from spreading between generations of fish.

2.3 Production of Fish Feed

The need for cost efficiency and economies of scale are the main reasons behind the consolidation in the feed industry, while liberalizations related to biomass and license regulations have contributed in the salmon farming industry. According to calculations made by iLaks.no (2013), profitability in the feed market is strongly correlated with production volumes. While the established players had an average profit margin of 4% to 7% during the period from 2007 to 2012, the entrant Polarfeed has consistently been losing money. In 2012 they posted their first positive result and have plans for expanded production. The main challenge for small producers is high fixed production costs. This gives the large producers an advantage, as they can achieve economies of scale. Naturally, most salmon farmers

in Norway buy feed from one or more of the three major industry players, EWOS, Skretting and BioMar. Cermaq was an exception, with in-house production of feed, until it sold EWOS in July 2013.

Fish feed is produced as pellets containing protein, fat and carbohydrates. Raw materials are mixed before water is added and the mixture is extruded into pellets of desired size. Different sized fish require feed of different size, since small fish can not eat too large pellets. The pellets are then dried and oil is added, before they are cooled down. The feed is distributed in large bags or as bulk from silo to silo. Silo-to-silo is a newer technology being increasingly used today. It is important that the feed sink at the right rate and tolerate handling in silos and feeding machines.

The type of feed consumed by the salmon is an important factor in determining its growth and development. In the course of a salmon's life, different amounts of proteins and fatty acids are needed, depending on fish size, fish health and sea temperature. High competence on feed composition is key to delivering feed that will ultimately yield high growth and high quality fish. Therefore, fixed costs related to product development are substantial for a fish feed producer. The largest component in variable production costs is raw materials, with uncertain prices due to volatile markets (Marine Harvest, 2013b). Feed producers transfer the risk of varying raw material prices to salmon farmers through cost-plus contracts. The farmer pays the realized raw material cost plus a premium including transportation costs and profit margin. Traditionally, fish meal and fish oil were the main components in salmon feed. Constrained availability of these resources and a growing fish farming industry has led to a higher content of agricultural ingredients, such as rape seed oil, soy protein and wheat. These are both cheaper and more environmentally sustainable.

2.4 Salmon Feeding

Two feeding strategies can be distinguished, meal feeding and continuous feeding. They differ in the timing and the duration of meals. Continuous feeding provides feed throughout the day in small doses, while meal feeding outputs more feed over a shorter period of time. Which strategy to apply depends on, among other things, fish size and sea temperature. The available control equipment determines the efficiency of feeding, for example by detecting when the salmon are full to ensure that they are fed enough, while having minimum wastage of feed. Special types of feed can be used to ensure higher immunity, higher growth rates or to help smolt adapt to salt water. With a diversity of different sizes and types of food, the salmon producer can customize feeding to the local environment, seasonal varieties and the size and current needs of the fish.

As can be seen in Figure 2.5, feed costs account for approximately half of a salmon farmer's production costs. Therefore, cost-efficient feeding is important. An important formula is the feed conversion ratio, given as consumed feed divided

by increase in biomass (Skretting, 2009). The ratio measures how much feed is needed to increase the biomass and should be as low as possible. Salmon has a feed conversion ratio close to one, which is superior to other types of meat, such as poultry and pork. Salmon are cold-blooded and therefore spends less energy on staying warm. In addition, movement takes less energy through water compared to on land.

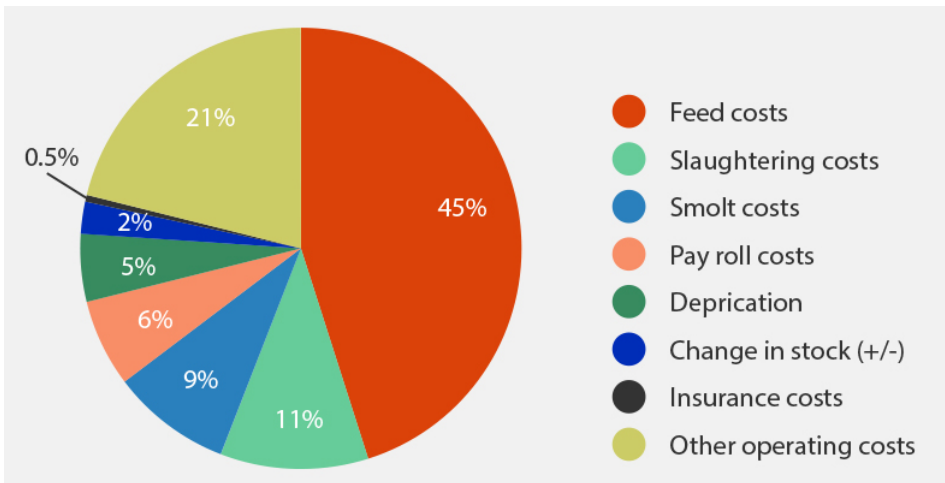


Figure 2.5: Components of production costs for fish farmers in Norway (Fiskeridirektoratet, 2013).

A planner forecasts demand for all fish farms with input from the location manager of each fish farm, and place orders with the feed supplier. The delivery time is around two weeks with an opportunity to adjust the quantity, within negotiable limits, closer to delivery. Changing the ordered feed type is usually not possible. Since salmon growth depends on many uncontrollable factors such as sea temperature, sea conditions and disease breakouts, uncertain demand of feed is a reality. Sea transportation can be uncertain due to weather conditions or ship breakdowns. If a supplier has to prioritize among customers, the fish farmer is not guaranteed to get his feed in time. This could be expensive since it results in a production halt as salmon growth slows down.

2.5 Vertical Integration in the Salmon Industry

There are several reasons for a salmon farmer to buy feed externally, as well as producing in-house. The main arguments are summarized in Table 2.1. The production of fish feed is most profitable in companies producing large volumes. A specialized company is able to maintain higher volumes by selling to several

aquaculture companies. In addition to lower unit production costs, large producers are able to maintain a cost-efficient inventory of many feed types. By aggregating demand forecasts over several fish farms, the same service level can be retained with less inventory (Simchi-Levi et al., 1999, chap.2). Also, the constrained supply of marine ingredients for fish feed leads to the challenge of using alternative resources while maintaining the nutritional value of the fish feed. Fish feed companies have specialized employees with the necessary knowledge to create recipes that are both highly nutritional and cost-efficient. The large quantity produced also enables them to recover more of the potentially high research and development costs.

On the other hand, a large fish farmer could also attain economies of scale with internal production. The internal producer only needs to produce for its own needs and the variety in product types can be limited, allowing for more efficient production. Agreeing on product specifications could be more effective with the command line structure of an internal organization. By handing out less information to external suppliers, the company will be able to better protect trade secrets. Also, ordering and logistics can potentially be done more effectively and there are no sales or marketing costs. An integrated company can maximize across the entire value chain and thereby avoid suboptimizing on quality levels or produced quantity. It is thus less exposed to the market power of suppliers and double marginalization (Kreps, 2004, chap.6). Being an integrated company could also ease the implementation of VMI, since this is dependent on effective and truthful information sharing, which should be easier within the same company. VMI may result in lower risk of empty inventories. Since the risk of changing raw material prices is already transferred from feed producers to buyers, internal production would not worsen the risk situation for the salmon farmer. All this could result in lower costs compared to the transaction costs from buying in the market (Coase, 1937).

External supply	In-house production
Economies of scale	Customized product line
Cost-efficient inventory	Lower ordering and logistics costs
Specialized competence	No sales or marketing costs
	Hide trade secrets
	Avoid suboptimization
	Easier implementation of VMI

Table 2.1: Reasons for external feed supply versus internal feed production for a salmon farmer.

Chapter 3

Problem Description

Marine Harvest is integrating upstream to gain more control of the value chain and capture synergies from in-house feed production. Implementing VMI for feed delivery to fish farms is an important part of capturing these wanted gains. To achieve such goals, it is necessary to create cost-effective and robust transportation plans, taking into account both routing and inventory management. In Section 3.1 we present Marine Harvest's planning situation and the challenges they are facing. In Section 3.2 we give a detailed description of the problem we model in Chapter 5, which is based on Marine Harvest's planning problem. We also describe the assumptions we have made to simplify the problem. Lastly, a summary of the problem characteristics is given in Section 3.3. Information and data related to the planning situation have been received by email and through meetings with involved employees in Marine Harvest (Marine Harvest, 2014).

3.1 Problem Background

Marine Harvest is building a new feed factory at Valsneset in Sør-Trøndelag. This will be ready for operation in June 2014. The capacity of the factory is nearly 400 000 metric tons per year, or 33 000 per month. Due to seasonal demand variations and constrained storage capacity at the factory, the yearly production capacity will not be fully utilized. Therefore, Marine Harvest predicts yearly deliveries from the factory to be around 300 000 metric tons, while the total demand is 340 000. The remaining 40 000 metric tons will be supplied externally by their current feed suppliers, Skretting and BioMar.

Marine Harvest has around 115 fish farms located along the Norwegian coast, as illustrated in Figure 3.1. The coast is divided into four regions, North, Mid, West and South. An important aspect of the planning problem is that demands



Figure 3.1: Marine Harvest's fish farms located along the Norwegian coast. The orange circle represents the feed factory (Marine Harvest, 2012).

vary considerably throughout the year. The monthly production rate is held fixed before and during high season, but is reduced during winter, when the fish grow more slowly. To balance these variations, fish farms are included or excluded in the distribution plan according to season. During low season, Marine Harvest could potentially serve all fish farms internally, while this is not possible during high season, due to constraints on both production and transportation capacity. Then, the factory will most likely supply fish farms in region Mid and parts of region West and North, since it is located in region Mid.

Since the factory will mainly produce four sizes of standard feed, special feed must be provided by external suppliers. As explained in Chapter 2, some fish farms release smolt to seawater during fall instead of spring. These smolt are smaller than normal and need small-sized pellets, which the factory does not produce. If

the two feed boats are strained on capacity, it could be profitable to have external suppliers completely serve fish farms with special feed needs. The decisions of which fish farms to serve from the factory and for which to order externally must be taken several weeks in advance. The current lead time on feed orders is around two weeks, but this is expected to increase when Marine Harvest starts internal production. This is both due to Marine Harvest becoming a less valuable customer with smaller volumes and because of increased volatility in their orders.

Marine Harvest has invested in two LNG-fueled ships to deliver feed in bulk using a technique called silo-to-silo. The feed is stored in silos at the factory, on board ships and at fish farms. Different feed types must be stored separately. Loading and unloading of feed takes time, and does sailing between locations. When a ship loads at the factory, the factory stock is cleared, given that there is room on the ship. The ships are not dependent on visiting the factory or fish farms during working hours. However, at many of the fish farms, maximum silo capacity can only be utilized when there are people at work who can control the unloading. Outside normal working hours, these silos can only be filled up to a certain percentage of capacity, usually around 90 %. The storage capacities, production rate and demand rates constrain the scheduling of deliveries. A feed delivery plan must consist of both the routing and scheduling of ships and the loading and unloading quantities at the factory and fish farms.

In the current scheme, feed is usually delivered for four to five days at a time. The silos should never run empty and to ensure this Marine Harvest maintains a safety stock equal to one day of feed. Timely deliveries are important and schedules should be robust enough to account for unforeseen events. Uncertainties related to biological production and weather conditions lead to both uncertain demands and deliveries. The demand of feed at each fish farm is determined by fish growth, which is again determined by sea temperature and disease breakouts, among other things. Bad weather conditions could lead to delays and ships could break down. The resulting transportation plan should ensure in-time delivery of feed, to avoid expensive emergency deliveries from other suppliers, while minimizing transportation costs.

During our previous work with Marine Harvest it was not sure whether they would implement continuous planning according to inventory levels or create weekly schedules with predetermined quantities. The first option is able to realize the full potential of VMI, while the second provides predictable schedules at the loss of flexibility. Marine Harvest now considers the concept of a weekly schedule to be too rigid and an artificial construction that does not make sense when deciding on optimal feed deliveries. With constantly changing demands, the schedules would have to be changed as often as every month and the argument of predictable delivery dates falls short. Therefore, they are in the process of developing a new decision support system (DSS) for production scheduling and delivery planning.

Currently, deliveries are planned using weekly demand forecasts from regional managers. They also use a 12-month rolling horizon forecast of feed demand,

but they do not account for starving or diseases resulting in lower demand. The situation today is that many fish farms do not keep their numbers up to date and mistakes are made. It is essential that routines for more frequent forecasting and updating of data at each fish farm are in place before the new system is implemented. The DSS will be run daily with updated demand forecasts and current inventory levels. The user can input parameters such as stock capacities and restrictions on visits to a fish farm. The system will most likely be based on a heuristic in order to generate feasible solutions fast. We do not know how good these solutions will be compared to solutions obtained from an exact method. Unfortunately, the DSS will not be ready in time for us to compare its solutions with our results.

3.2 Detailed Problem Description and Assumptions

In order to model a problem based on Marine Harvest's current planning situation, we have made several assumptions. We wanted our model to be more general and decided to include the possibility of having heterogeneous ships and several factories. The fleet of ships is fixed and we assume that it is not possible to hire capacity from elsewhere, since this is highly specialized equipment. If production capacity is insufficient, feed must be bought externally. By considering heterogeneous ships and several factories, the model is usable also if Marine Harvest decides to expand production and transportation capacity. For the implementation stage of our work, we will only look at the current situation with one factory and two homogeneous ships. Our main purpose is to develop a model to aid Marine Harvest's planning of feed deliveries. Therefore, the model also includes problem specific characteristics, such as clearing factory stock when loading and a reduction in fish farm capacity outside working hours.

The objective is to minimize costs, while ensuring in-time deliveries. Costs related to both transportation and external supply must be considered. Transportation costs include fuel consumption and other costs of operation that do not incur when a ship is idle. Fixed costs related to ships are disregarded as they are not affected by the decisions to be made. Since we are dealing with a vertically integrated supply chain, transportation will not affect inventory costs and these costs are also disregarded. External feed costs consist of transportation costs plus the supplier's profit margin. For fish farms requiring external supply of special feed, the transportation cost of external supply should be excluded. As agreed with Marine Harvest, the focus is not solely on cost minimization, but also on securing feed deliveries and creating robust delivery schedules, since lost feed days is a large cost driver. Achieving high utilization of production and ship capacity could be alternative objectives, as these are large investments.

Ships are allowed to wait upon arrival, but we assume that they do not need to stop

for maintenance during operation. In accordance with Marine Harvest's view of operations, ships are assumed to clear the stock when loading at the factory. Due to limited production capacity, there must be a certain number of days between departures if the ships are to leave factories with close to full shiploads. Therefore, we also assume that ships can not be loaded simultaneously, even though this is physically possible. To ensure evenly distributed deliveries to fish farms, there should also be a certain number of days between feed deliveries to a particular fish farm.

Consumption and production rates are assumed constant within the planning horizon, which is reasonable with a planning horizon of around two weeks. Feeding is assumed to be continuous. For fish farms with a high demand rate, Marine Harvest thinks it could be beneficial to deliver feed to a fish farm more than once during a ship's voyage. A voyage is defined as a visit to a factory for loading, followed by consecutive visits to fish farms before returning to a factory.

Several simplifications are made in order to reduce the complexity of the problem. We do not look into aspects of procurement of raw materials or production scheduling. The scheduling of different feed types is assumed to be appropriately carried out for the given forecasts, and we only look at an aggregated production rate. Changeover time between producing different types, seasonal variations in working hours and downtime in production will not be included explicitly in the model, but are considered when calculating an average production rate. Neither do we look into the production of smolt or salmon nor the growth rates of these. This part of the value chain is only included through the demand forecasts made by the local or regional managers.

In the real world, demands are inherently uncertain. We will limit ourselves to a deterministic consumption rate where weekly demand at each fish farm is assumed known beforehand. By running a DSS with daily updates, changes in demand can be discovered and schedules adjusted in time. Marine Harvest does not have any experience with how weather conditions could affect their operations, but they realize that unforeseen events could have severe effects. The problem is complex as it is and we have decided to not include stochastic elements. Even though stochastic programming is not used, uncertainty is accounted for by incorporating a safety stock level of one feed day.

The decisions to be made are which fish farms to visit and how much to deliver of each feed type to each fish farm. Also, how much of each feed type to load at each visit to a factory must be decided. There are multiple feed types and multiple silos both in factories, on board ships and at fish farms. At fish farms, the minimum stock level should be considered per type, while stocks are aggregated when considering the maximum stock level. Most fish farms only have one feed type delivered, but it is still important to consider safety stock levels for each type, to ensure that inventories are not too low. The stock of feed at factories is aggregated across different types. This assumption is reasonable due to the high number of separate storage silos at the factory.

On board the ships there are fewer silos, but allocating products to silos is a complex combinatorial problem. One alternative is to dedicate silos to specific feed types. Another alternative is to reduce the ship capacity if a solution turns out to be infeasible regarding stowage, so that solutions are more likely to be feasible. Both approaches could make the model too strict by cutting off feasible solutions. Therefore, allocation of products is disregarded and we look at an aggregated ship capacity. If a solution turns out to be infeasible in the real world, experience from earlier runs can be used to set product specific capacity if necessary. Different feed types have different densities and silo capacity in tons is dependent on type. Since the differences are small and we aggregate product types when considering silo capacities, we consider a single capacity given in tons for each silo.

Another decision to make is how many fish farms to serve internally, and the volume of feed to order from external suppliers. If the transportation or production capacity is insufficient to maintain non-negative stock levels at all fish farms, some fish farms must be supplied externally. In that case, an external supplier takes responsibility of the total demand throughout the planning period for those fish farms. At times when production capacity is too low to serve all fish farms, the fish farms farthest away from the factory will be excluded from the problem. Fish farms that can not be supplied internally due to incompatible storage types will also be excluded. The excluded fish farms must be supplied externally. Only fish farms that require a delivery during the planning period will be included in the problem.

3.3 Problem Summary

A summary of the problem we model in Chapter 5 is given below. The problem, based on Marine Harvest's current planning situation, is illustrated in Figure 3.2.

- Objective
 - Minimize transportation costs
 - Minimize cost of external supply
 - Ensure in-time deliveries of feed and avoid low stock levels
- Routing and scheduling
 - Fixed fleet of heterogeneous ships
 - Fixed set of factories
 - Fixed set of fish farms
 - Constant speed of sailing
 - Waiting on arrival is allowed
 - Constant rates for loading and unloading

- Feed factories
 - Production rate varies throughout the year, constant during planning horizon
 - Multiple products produced
 - Maximum stock level aggregated over feed types
- Fish farms
 - Feed supplied either internally or externally
 - Demand rate varies throughout the year, constant during planning horizon
 - Demand for one or multiple feed types at each fish farm
 - Maximum stock level aggregated over feed types
 - Minimum stock level per feed type
 - Unloading can only utilize maximum silo capacity within working hours

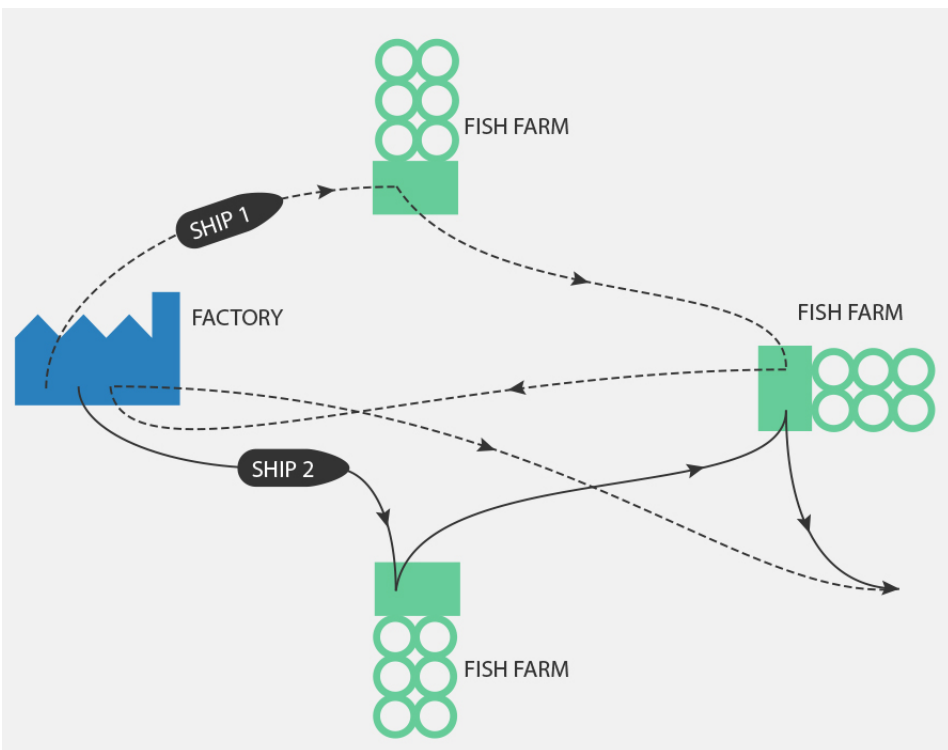


Figure 3.2: Illustration of Marine Harvest's planning situation.

Chapter 4

Related Literature

Applications of transportation problems are often variants of a vehicle routing problem (VRP). The customers to visit and the quantities to deliver are given and the problem is to decide on a route in order to satisfy all customer demand. For a supplier implementing VMI it is necessary to simultaneously decide on routes and manage the customers' inventories. This motivated research on the inventory routing problem (IRP) in the early 1980s. The IRP is also important for internal distribution, where the supplier and customers are part of the same vertically integrated company. Marine Harvest's transportation problem can be categorized as both an IRP and a periodic vehicle routing problem (PVRP). Literature on similar problems use both formulations and we have considered both model types in previous work (Ivarsøy and Solhaug, 2013). In this thesis, we continue our work on the IRP formulation and the focus of this literature review will therefore be on IRPs. We also present research on parallel branch-and-bound frameworks, because we believe this could be combined with more traditional IRP solution methods to solve real-sized instances. We ask the reader to bear in mind that this literature review is not meant as an exhaustive review of all published literature on IRPs, but rather a presentation of papers that we consider to be the most relevant for our version of the IRP.

The earliest research on combining inventory and routing decisions is reviewed in Federgruen and Simchi-Levi (1995), while more recent reviews are Andersson et al. (2010) and Coelho et al. (2014). The classical VRP assumes transportation by truck and this also applies to much research on the IRP. However, problem applications from the past two decades are often set in the maritime transportation industry, which will be the focus of the literature presented here. In Section 4.1 we will discuss differences between land-based transportation problems and problems within maritime transportation. In order to relate our problem to previous research, we will present relevant IRP characteristics in Section 4.2. In Section 4.3 we will look at different solution methods and present articles using both exact optimization methods and heuristics. Section 4.4 introduces the concept of

parallel branch-and-bound and presents developed frameworks for implementing such algorithms.

4.1 Maritime and Land-Based Transportation

Research on transportation problems has traditionally focused on land-based transportation. A basic VRP has one supplier with several customers, and the capacity of the vehicle is much larger than the delivery to one customer. Also, the fleet of vehicles is homogeneous and different products can be stowed together. Maritime transportation differs in several ways. In ship routing problems there are often several loading and unloading ports and the coupling between these are not necessarily fixed. Also, one delivery is often large compared to ship capacity. The fleet is usually heterogeneous and there are multiple products transported that can not be mixed. The nature of maritime transportation is also different. Little slack in travel schedules and capital intensive operations make the effects of uncertainty related to breakdowns and weather conditions severe in the maritime industry.

These differences are noted by Christiansen and Fagerholt (2009) in their description of a basic inventory ship routing problem (ISRP). In Agra et al. (2013b) the same type of problem is referred to as a maritime inventory routing problem (MIRP). Andersson et al. (2010) and Coelho et al. (2014) point out that it is difficult to decide on one standard version of the problem, since for every real application of the problem, a new version is created. In the next section we will use a classification scheme similar to the one used by Andersson et al. (2010) and Coelho et al. (2014) to review articles by presenting characteristics relevant to our problem.

4.2 Inventory Routing Problems

Aspects of the IRP include planning horizon, time considerations, inventory policy, fleet of vehicles, structure of the distribution system, number of products and nature of demand. Our problem has a planning horizon of ten days. During the planning period, demands and production rates can be considered constant. Inventories should not fall below a given safety stock level. The fleet of ships is heterogeneous, and there can be many factories supplying multiple products to many fish farms. In this section we will look into how the characteristics of our problem relate to other applications of the IRP. Table 4.1 provides an overview of problem characteristics related to the articles mentioned in this section.

	Time	Dist. network	Inventory	Multiple products	Fleet	Demand
Our problem	Cont.	Many-to-many	Non-neg., penalty below safety stock level	Yes	Het.	Det.
Agra et al. (2013b)	Disc. and cont.	Many-to-many	Non-neg.	Yes, but dedicated compartments	Het.	Det.
Al-Khayyal and Hwang (2007)	Cont.	Many-to-many	Min. stock level	Yes, but dedicated compartments	Het.	Det.
Bertazzi et al. (2011) Coelho et al. (2012)	Disc.	One-to-many	Can be negative	No	Hom.	Stoch.
Christiansen (1999)	Cont.	Many-to-many	Min. stock level	No	Het.	Det.
Christiansen et al. (2011)	N/A	Many-to-many	Min. stock level	Yes	Het.	Det.
Dauzère-Pérès et al. (2007)	Disc.	One-to-many	Min. stock level	Yes	Het.	Det.
Grønhaug and Christiansen (2009)	Disc.	Many-to-many	Min. stock level	No	Het.	Det.
Popović et al. (2012)	Disc.	One-to-many	Min. stock level	Yes	Hom.	Det.
Ronen (2002)	Disc.	One-to-one	Non-neg., penalty below safety stock level	Yes	Het.	Det.

Table 4.1: Characteristics of problems from related research.

4.2.1 Planning horizon

The length of a planning horizon should depend on the decisions to be made. Tactical decisions need a longer planning period than operational decisions. Since a longer planning horizon often makes the problem more difficult to solve, Rakke et al. (2011) use a rolling horizon approach to determine annual deliveries of natural gas. Their problem is solved for a shorter planning period, and the output is used as input for a new run, until the whole year is planned for. When uncertain forecasts affect the decisions made now, a longer planning horizon could be useful also for short-term decisions. There are many definitions of what is considered operational and tactical decisions. A common view is that operational decisions are made on a daily or weekly basis, while tactical decisions apply to months, up to a year (Anthony, 1965). However, the classification also depends on the industry and the decisions to be made within the planning horizon. Christiansen et al. (2006) consider maritime IRPs in general to be tactical problems. Since our problem has sailing legs that are significantly shorter than sailing legs in many maritime transportation problems, it can be argued that our problem is operational. At the same time, Marine Harvest needs to create preliminary schedules months in

advance to decide on external supply, giving the problem a tactical nature. Due to this lack of a single definition, it is difficult to classify problems as either operational or tactical, and we have left this categorization out of Table 4.1.

4.2.2 Time

Time can be treated continuously or as discrete time periods. For an IRP with fixed demand rates through the planning horizon, a continuous time formulation may be appropriate. This approach is used in Christiansen (1999) for internal distribution of ammonia by ship. A time discretized model is used by Agra et al. (2013b) in a mixed integer formulation for a short sea fuel oil distribution problem. Time is discretized because the consumption rates vary throughout the planning period, but they also have continuous time variables constrained by given time windows for port operation. Agra et al. (2013a) discuss differences between discrete and continuous time formulations for MIRPs. Discrete time formulations result in large problems, but they often enable creation of strong cuts and tend to provide better bounds than continuous time models.

4.2.3 Inventory policy

In some applications, stock levels are not allowed to fall below zero or a certain safety stock level. Other applications penalize negative inventory as lost sales or they allow back-orders. Inventory constraints can be handled at both suppliers and customers, or only at one end. Ronen (2002) addresses a planning problem for shipping liquid bulk-products with inventory constraints at both suppliers and customers. Violation of the safety stock level is allowed at a penalty cost, but the stock levels can not be negative.

4.2.4 Fleet of vehicles

The fleet of vehicles can be homogeneous or heterogeneous. The first is common in land-based transportation, while the latter is more common for maritime applications. The number of vehicles can be fixed or unconstrained, if it is possible to acquire extra capacity. In Dauzère-Pérès et al. (2007) a heterogeneous fleet of vessels deliver calcium carbonate slurry to multiple European ports. The different-sized ships makes it difficult to model the objective function correctly, because of the nonlinear relationship between transportation costs and transported quantity. Popović et al. (2012) model fuel delivery from one depot to several petrol stations where vehicles are homogeneous and have multiple compartments. A homogeneous fleet allows for fewer variables by considering only compartments, instead of both vehicles and compartments.

4.2.5 Structure of distribution network

The structure of the distribution network is important for the modeling of the problem. Different networks are one-to-one, one-to-many or many-to-many. Also, there can be one or multiple customers to visit per route. In maritime transportation, continuous routing is common, where there are no well-defined start or end port. Grønhaug and Christiansen (2009) study a pick-up and delivery problem for LNG tankers, where routing is continuous and the network is many-to-many. In Dauzère-Pérès et al. (2007) there is one supplier and several customers, but ships usually visit only one customer per voyage.

4.2.6 Multiple products

Many real applications deal with transportation of multiple product types. Maritime transportation problems often involve different products that need to be shipped and stored in different compartments. To reduce complexity, simplifications are often made. Both Al-Khayyal and Hwang (2007) and Agra et al. (2013b) consider transportation of multiple products types where compartments on board are dedicated to certain types, thereby avoiding the allocation problem. In Christiansen et al. (2011) a cement producer transports multiple non-mixable products and stowage decisions are taken into consideration. This greatly complicates the problem, but is needed to ensure feasibility of ship schedules.

4.2.7 Nature of demand

Most models assume deterministic demand, either a constant rate throughout the planning period, or a time-varying demand. However, demand is usually uncertain and research has been done on the dynamic and stochastic IRP (DSIRP). Bertazzi et al. (2011) and Coelho et al. (2012) study a problem where a supplier has one vehicle to serve several customers with stochastic demands. Since the problems are dynamic and stochastic, uncertain demand is revealed over time and stockouts may occur. Both approaches use a rolling horizon where the model is rerun using either current inventories or forecasts.

4.2.8 Objective

The research reviewed focuses on minimizing both transportation costs and inventory costs. These are both relevant for a supplier engaged in VMI. For an integrated company performing internal distribution, the inventory costs are most likely not affected by transportation. In Christiansen (1999) distribution is within the same company and inventory costs are disregarded. Another relevant objective is to minimize penalty costs related to stock outs or falling below a given safety stock

level, as in Ronen (2002). This encourages more robust schedules and attempts to avoid expensive stockouts.

4.3 Solution Methods

The basic IRP is a generalized version of the VRP, proven to be NP-hard by Lenstra and Kan (1981). Therefore, much of the available research involves using heuristic solution methods. Literature on exact methods includes decomposition methods and branch-and-cut algorithms. Note that when we discuss exact methods we refer to exact optimization methods that may be terminated before reaching optimality. Due to the complexity of the IRP, developing a tight formulation is essential. In this section we start with a discussion of different formulations, before we give examples of successful exact methods and common heuristics applied to IRPs. Table 4.2 provides an overview of the different solution methods used by the articles mentioned in this section.

	Formulations			Solution methods				
	Arc-flow	Path-flow	Ext.form.	Val. ineq.	B&C	Col.gen.	Heur.	Roll. hor.
Adulyasak et al. (2014)	✓			✓	✓			
Agra et al. (2013b)	✓		✓	✓				
Agra et al. (2014)	✓		✓	✓			✓	✓
Andersson et al. (2011)	✓	✓		✓				
Archetti et al. (2007)	✓			✓	✓			
Bredström and Rönnqvist (2006)		✓						✓
Christiansen (1999)	✓	✓				✓		
Christiansen et al. (2011)							✓	
Coelho and Laporte (2013)	✓			✓	✓			
Dauzère-Pérès et al. (2007)							✓	
Engineer et al. (2012)		✓		✓	✓	✓		
Grønhaug and Christiansen (2009)	✓	✓						
Grønhaug et al. (2010)		✓		✓		✓	✓	
Hwang (2005)	✓	✓				✓	✓	
Popović et al. (2012)	✓						✓	
Rakke et al. (2011)							✓	✓
Solyali and Süral (2011)	✓			✓	✓			
Song and Furman (2013)	✓			✓	✓		✓	
Stålhane et al. (2012)		✓		✓	✓	✓		

Table 4.2: Formulations and solution methods used in related research: arc-flow, path-flow and extended formulations, valid inequalities, branch-and-cut, column generation, heuristics and rolling horizon.

4.3.1 Model formulations

Many articles on IRPs start with an arc-flow formulation, which is further developed into a path-flow formulation (Christiansen, 1999; Grønhaug and Christiansen, 2009). The review by Andersson et al. (2010) reports on successful path-flow formulations for ship routing problems and argues that the same advantages could be obtained for IRPs. However, Rakke et al. (2014) argue that it is not always appropriate to use a path-flow formulation for complex shipping problems. If the extreme points of the resulting subproblems are integer, the gain from a path-flow formulation is lost. An arc-flow formulation can also be reformulated using extended formulations, typically with more variables to capture the combinatorial structure better. Such a reformulation could lead to tighter linear relaxations at the expense of more variables. A common approach to extend an arc-flow formulation is to replace the flow variables with multi-commodity flows (Vanderbeck and Wolsey, 2010).

Andersson et al. (2011) study a shipping problem with given quantities for mandatory and optional cargo loads, where loads can be split and carried on several ships. They first give an arc-flow formulation of the problem and then move on to two alternative path-flow formulations. In the first path-flow formulation, quantities are assigned to each selected route in the master problem. The second formulation has aggregated quantity variables, independent of schedule. The problem is solved using a branch-and-bound method where all non-dominated schedules are generated a priori. The solution method is able to solve small instances to optimality. By heuristically reducing the number of generated schedules, larger instances can be solved.

Agra et al. (2013a) study continuous and discrete time arc-flow formulations for a short sea inventory routing problem. They use valid inequalities and extended formulations to reduce the integrality gap. This tends to increase the solution time of the linear relaxation, but the size of the branch-and-bound tree is reduced. Valid inequalities are created by calculating minimum and maximum number of visits to customers. Also, models are improved by tightening constraints linking binary and continuous variables and reducing the size of time windows. Tighter constraints can be developed for discrete time formulations compared to formulations with continuous time, as more information is included in the variables. Extended formulations include an arc-load flow model and a multi-commodity flow model. The multi-commodity flow model is the tightest formulation, but gives a dramatic increase in variables. By using the arc-load flow model with continuous time, they are able to solve real-sized instances with a planning period of 15 days to optimality.

4.3.2 Exact methods

Branch-and-cut

A popular exact method is branch-and-cut, which was first introduced to solve single vehicle IRPs to optimality by Archetti et al. (2007) and improved by Solyali and Süral (2011). They use a time-discrete arc-flow formulation where subtour elimination constraints are initially excluded. These are added when violated during the branch-and-bound search, in order to create stronger bounds. Adulyasak et al. (2014) and Coelho and Laporte (2013) further develop the branch-and-cut algorithm to solve multi-vehicle instances to optimality. Adulyasak et al. (2014) propose two different formulations, one with and one without a vehicle index. For the formulation with vehicle index, symmetry breaking constraints are added. Coelho and Laporte (2013) focus on solving less basic versions of the problem, with constraints on delivery consistency. They also apply a solution improvement algorithm that experiments with vertex removals and reinsertions whenever the branch-and-cut procedure finds a new best solution.

Decomposition and column generation

The complexity of the arc-flow model studied in Christiansen (1999) calls for a tailor-made solution approach. The model is reformulated as a path-flow model and Dantzig-Wolfe decomposition is used to generate ship schedules in subproblems for each ship and visit sequences in subproblems for each harbor. The master problem includes additional coupling constraints to link auxiliary variables to the original ones. Both types of subproblems are formulated as shortest path problems, and solved by dynamic programming. The Dantzig-Wolfe column generation approach only generates the most promising columns, which are the ship routes and harbor visit sequences with the least reduced costs in the master problem.

Hwang (2005) suggests Lagrangian relaxation to decompose a similar problem by penalizing violations to coupling constraints in the objective function using Lagrange multipliers. Grønhaug and Christiansen (2009) and Grønhaug et al. (2010) consider an inventory routing problem in the liquefied natural gas business. In Grønhaug and Christiansen (2009) the path-flow model is solved by full enumeration of columns and the algorithm suffers from low scalability. In Grønhaug et al. (2010) column generation is used to find the best path to add to the master problem. The subproblems are solved using dynamic programming, while the master problem is solved using branch-and-bound. Also, valid inequalities are added to tighten the formulation.

Engineer et al. (2012) present a path-flow formulation solved by combining column generation and branch-and-cut in a branch-price-and-cut approach. Cuts are added to the master problem during the branch-and-bound procedure to tighten the linear relaxation. Two classes of cuts are added, the first is related to port and ship capacity, while the second class is related to timing of visits. Branch-price-and-cut

is also used in Stålhane et al. (2012) and valid inequalities such as cover inequalities on cargoes, k -path inequalities and branching-dependent inequalities are added to the master problem.

4.3.3 Heuristic methods

Many recent solution methods for IRPs are hybrid heuristics combining a heuristic approach with metaheuristics or an exact optimization method. Dauzère-Pérès et al. (2007) are only able to solve small instances of their calcium carbonate slurry delivery problem to optimality using a standard solver. Instead, they base their solution approach on a metaheuristic consisting of a greedy heuristic that determines transportation plans for each location, a local search to improve the solution and a genetic algorithm to expand the solution space. Christiansen et al. (2011) use a similar approach. Another commonly used heuristic is the variable neighborhood search heuristic, which is randomized by Popović et al. (2012) and combined with branch-and-cut by Song and Furman (2013).

In order to handle long planning horizons, rolling horizon heuristics have been developed. Agra et al. (2014) combine a rolling horizon heuristic with local branching and a feasibility pump to solve their aforementioned arc-load flow formulation. The feasibility pump finds initial solutions and local branching is used to improve the current feasible solution. The rolling horizon heuristic works by dividing the planning horizon into three periods. Binary variables are only used in the central time period. In the first time period they are fixed, while in the last time period they are continuous. This combination of heuristic strategies provides promising results for problems with a long planning horizon. A similar rolling horizon heuristic is used by Rakke et al. (2011) to determine annual deliveries of natural gas and by Bredström and Rönnqvist (2006) to plan distribution of several pulp products to customers. Agra et al. (2014) use a time-continuous formulation, while the two latter articles have time-discretized formulations.

4.4 Parallel Algorithms

Industrial applications of combinatorial problems are often complex and have an immense solution space. Therefore, much research has been done on using multiple processors to search the solution space in parallel. It is important that the solution time using a parallel algorithm scales linearly with the number of processors used, to justify the use of extra resources. We will first present some general aspects of parallel branch-and-bound algorithms, where it is the work of evaluating different subproblems that is parallelized. A lower level of parallelism would be to parallelize the work done on a single subproblem, not changing the overall structure of the branch-and-bound algorithm. A higher level of parallelism would be to build entire search trees in parallel. Lastly, we present some developed frameworks for implementing parallel branch-and-bound. We have not found any articles related

to IRPs, but we end the chapter with an application of parallel branch-and-bound on a capacitated VRP.

4.4.1 Aspects of parallel branch-and-bound

A parallel system is the combination of a parallel architecture and a parallel algorithm (Kumar and Gupta, 1994). The design of a parallel algorithm is strongly influenced by the architecture it is implemented on. Corrêa and Ferreira (1996) provide a classification of parallel branch-and-bound implementations with a shared data model versus a distributed data model. Today's high performance computing (HPC) clusters are usually hybrid versions of these, implemented as a network of shared memory computers. The three models are illustrated in Figure 4.1. Gendron and Crainic (1994) discuss several aspects of parallelism at the hardware level such as control, synchronization, granularity of work, communication and number of processors. Algorithms are then categorized according to whether they are synchronous or asynchronous and whether they have one or multiple work pools to locate and store data. In synchronous algorithms, the processors synchronize their work at certain points in time, while in asynchronous implementations, they work independently of each other.

A branch-and-bound algorithm has four important rules. A selection rule says which subproblem to branch from next and a branching rule governs which variable to branch on. A bounding rule says how to compute a lower bound for a subproblem and an elimination rule says when subproblems can be pruned. In sequential branch-and-bound these rules are applied in sequence to the active subproblem at each iteration of the algorithm. An efficient search is ensured by considering the current best solution and the order of active subproblems. In parallel branch-and-bound, each processor works on different subproblems in parallel and one must consider the concurrent knowledge handling between them. To enable an efficient search and high utilization of processors, knowledge must be shared and work must be evenly distributed among processors through load balancing strategies.

For a synchronous algorithm with shared memory, quantitative and qualitative workload sharing is guaranteed. With asynchronous algorithms, local datasets are not immediately updated and some processors risk working on subproblems that should have been eliminated. The branch-and-bound tree is not known beforehand, and subproblems are generated and selected in an unpredictable way for asynchronous, distributed implementations, causing irregularities in the algorithm. Trienekens and Bruin (1992) present a taxonomy of parallel branch-and-bound algorithms with four dimensions considering how knowledge is shared and used, how work is divided among processors and the synchronicity of the algorithm. There is a trade-off between communication and search efficiency. Synchronous algorithms tend to yield high idle times and the shared database becomes a bottleneck with many processors requiring access. An asynchronous

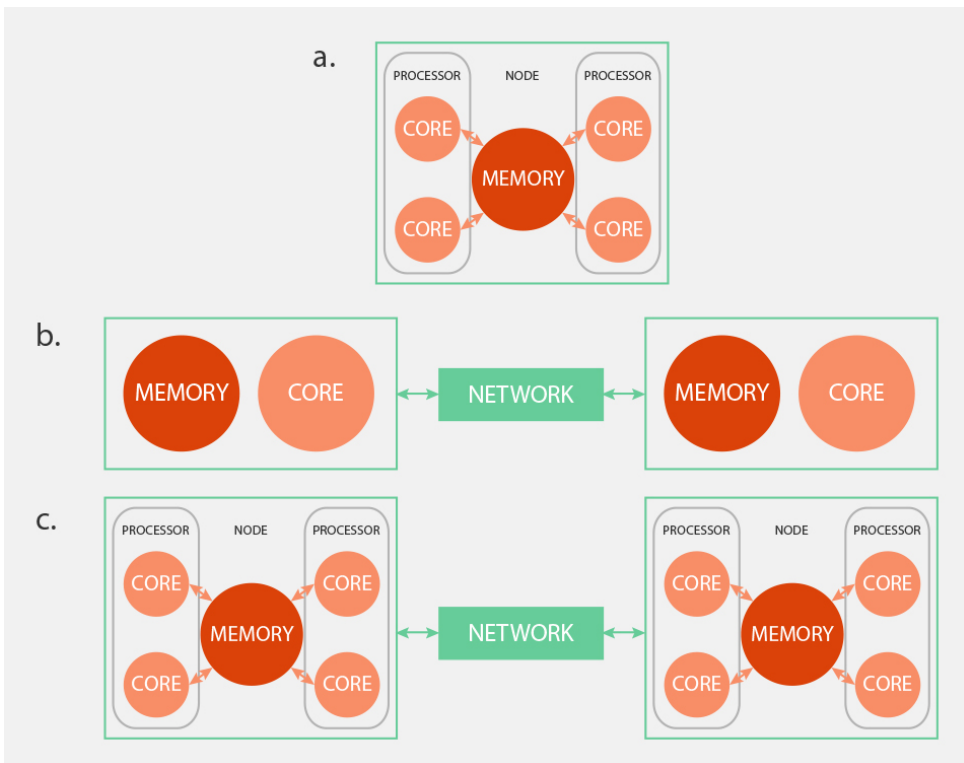


Figure 4.1: Illustration of a shared (a), distributed (b) and hybrid (c) data model.

algorithm avoids high latency times, but search efficiency could be lower because of redundant work.

Work breakdown into smaller tasks requires some pre- and post-processing and the granularity of tasks affects the amount of communication needed. This initialization, termination and communication, along with possible redundant work results in overhead time, which could make a parallel algorithm less efficient than the sequential. There are many measures on the efficiency of parallel algorithms. The speedup relates the solution time of the parallel algorithm to the solution time of the best sequential algorithm. If T_S is the sequential solution time and T_p is the parallel solution time using p processors, the speedup S is defined as T_S/T_p . The efficiency E is then given as the speedup divided by the number of processors, S/p . In theory, the maximum possible efficiency is one, but with asynchronous parallel algorithms, acceleration anomalies can be observed causing super linear speedup.

The speedup does not continue to increase as the number of processors increases and it peaks at a certain value. The isoefficiency function is a well-established measure for the scalability of a parallel algorithm. It is the rate at which the problem size must increase with respect to the number of processors to keep a constant efficiency

(Grama et al., 1993). The algorithm is scalable if this rate is small, meaning that it can utilize more processors efficiently even with a small increment in problem size. This measure helps capture the effect of algorithms that seem attractive due to low overhead, but have limited concurrency. The efficiency is dependent on the time spent on each subproblem, speed of communication, the network topology, the load balancing strategy and whether the problem experiences anomalies. Typical characteristics of parallel designs related to low and high scalability are shown in Table 4.3.

Low scalability	High scalability
Centralized control	Decentralized control
Synchronous operation	Asynchronous operation
Shared memory	Distributed memory
Fine-grained tasks	Coarse-grained tasks

Table 4.3: Typical characteristics of algorithms with low and high scalability.

4.4.2 Applications

The simplest parallel implementation is a master-slave scheme, where one processor controls all other processors. This central control results in a bottleneck as the number of processors increases. Therefore, more scalable systems with several hubs controlling subsets of workers have been developed. Eckstein (1994b) describes a distributed implementation of parallel branch-and-bound which allows for a varying degree of centralization. Design decisions on whether control should be centralized or distributed and how to store and distribute data are discussed. Experiments are done with one hub and many workers and with a fully decentralized implementation where each worker is its own hub. The decentralized version is slower, mainly because processors experience high idle times. In distributed schemes it is essential to have a good work sharing strategy, both in terms of quantity and quality of subproblems. As an improvement, Eckstein (1994a) combines a randomized work sharing scheme with a global view of work balancing. This combination yields a scalable method, competitive with more centralized schemes.

Eckstein et al. (2001) describe an object-oriented framework for parallel branch-and-bound called Parallel Integer and Combinatorial Optimizer (PICO), based on the aforementioned work in Eckstein (1994b). The goal is a general framework, separated from the application and computing platform, to allow a wide variety of branch-and-bound algorithms and parallel systems. This is realized through polymorphism and inheritance in C++ and by using base classes, run-time parameters and required and optional methods. PICO has a serial layer and a parallel layer, where the parallel layer is designed using a distributed-memory model and the standard message-passing interface (MPI) to pass messages between

processors. Processors are partitioned into clusters controlled by hubs and work is distributed both within and between clusters, as in Eckstein (1994b). Computational testing reveals larger search trees for the parallel implementation, due to the lack of a sophisticated incumbent heuristic, but the speedup is near linear for 32 to 48 processors.

Ralphs et al. (2004) describe a library hierarchy for implementing scalable parallel search algorithms for data-intensive applications. The base layer is the search handling layer called Abstract Library for Parallel Search, (ALPS). This includes search tree management and load balancing. On top of this is a data handling layer called Branch, Constrain and Price Software (BiCePS), used to represent variables and constraints and process subproblems. The framework is based on previous work by the authors, the Single or Multi-Process Optimization over Networks (SYMPHONY) written in C and COIN/BCP written in C++. SYMPHONY and COIN/BCP are based on master-slave paradigms, making them difficult to use for data-intensive applications. Therefore, ALPS has a layer of “middle management” with hubs between the master and workers, as in Eckstein et al. (2001). To further improve scalability, the processors work asynchronously and task granularity is increased from subproblems to subtrees.

Ralphs (2003) implements a solver for a capacitated vehicle routing problem using SYMPHONY. It is a modular implementation consisting of a master module, node processing modules and two modules for managing global data, one for search tree nodes and one for cuts. Storing search-tree nodes and cuts is memory-consuming and node processing and information sharing between processors is time-consuming. The parallel overhead is mainly due to communication and the time it takes to employ all processors during initialization. Since the number of nodes in the tree remains relatively constant as the number of processors increase, the conclusion is that little redundant work is done. However, this is mainly attributed to the single pool of data, which will be a scalability issue with larger problems and more processors.

Chapter 5

Model Formulations

In Chapter 4 we saw that the inventory routing problem (IRP) is a complex problem and that every application provides a new version of it. This is also the case with Marine Harvest's transportation problem. In previous work we formulated a time-continuous arc-load model based on an IRP and a frequency-based model similar to a periodic vehicle routing problem (PVRP) (Ivarsøy and Solhaug, 2013). We identified problem specific characteristics to model the problem as realistically as possible. Here we have further developed the IRP model, as this is the most accurate representation of the planning problem described in Chapter 3. In this chapter, we provide an improved version of our original arc-load formulation and two extended formulations with arc-flows and multi-commodity flows. All formulations are more general than the specific problem currently faced by Marine Harvest. We include the possibility to have several factories and a fleet of heterogeneous ships.

In Section 5.1, we start with an introduction which applies to all three formulations. In Sections 5.2, 5.3 and 5.4 we present the mathematical formulations of the improved arc-load model and the two extended formulations. All sets, indices, parameters, variables and constraints will be stated along with explanations of why they have been included. When we discuss aspects that apply to both fish farms and factories, we use the term locations. The complete mathematical formulations can be found in Appendices A, B and C.

5.1 Model Introduction

5.1.1 Common model characteristics

When building an IRP model, one has to consider whether a time-discretized or time-continuous model is most appropriate. A time-discretized model can handle

time-varying production and consumption rates, while a pure time-continuous model cannot. For time-discretized models there is a trade-off between the length of the planning horizon and the granularity of time periods, to avoid an intractable problem with too many time periods. A time-continuous model is less detailed as it is restricted to use fixed rates, but the size of the model is smaller. With a planning horizon of around two weeks, changes in demand and supply rates are not significant and we have chosen to use a time-continuous model. In order to separate different visits to a location, we enumerate each visit using visit numbers.

Ships have a given initial location and the model ensures that the ships start here and then enter and leave all subsequent fish farms and factories. After the last visit, the ship will travel to an artificial end node. Ships can unload outside opening hours at fish farms, but to account for the loss in capacity when nobody is there to control the unloading, soft time windows are added. This means that ships are allowed to unload at any time, but more feed can be unloaded if service starts within the given service time windows. Loading at factories can be done at any time. Visits to a fish farm should be separated by a certain number of days and this yields a lower bound on the quantity unloaded. The quantity loaded at the factory is given by the current amount of stock, which should be cleared.

To encourage robust schedules, low stock levels are penalized. Penalty costs are given per hour of having stock levels below the safety stock level. This means that for an equal amount of underage, a fish farm with a low consumption rate and thereby a low safety stock level is penalized more than a fish farm with a high consumption rate. Constraints are included to make sure that stock levels at the end of the planning period are within their limits. To account for end-of-horizon (EOH) effects, where stock levels at fish farms are at the minimum and the factories have reached maximum capacity, the constraints could be adjusted with upscaled safety stock levels and downscaled capacities. A more flexible alternative is to use aggregated constraints, where the total stock at fish farms must be above a certain level and the total stock at factories must be below a certain level. This will be further explored in the computational study in Chapter 7, but is not included in the following mathematical formulations.

5.1.2 Model applications

The model should be used as an operational planning tool to support decisions on delivery schedules. It should ideally be rerun once new information is available, and could thereby support weekly or even daily decisions on delivery planning. The operational decisions include how much to load on a boat that is due to leave the factory, which fish farms to visit when and how much to unload. The model could be used in a rolling horizon planning scheme, where operational decisions can be made for the coming week, while also taking into account forecasts for following weeks.

With a longer planning horizon, the model can be used as support for tactical

decisions, by giving an indication of how many fish farms the factory can serve throughout the year. Then, Marine Harvest can ahead of each season determine approximately how much feed to order from external suppliers. As demand is realized and forecasts are updated, adjustments on external orders can be made in time. The model can also explore whether each ship should be fixed to serve a particular set of fish farms, since there are fish farms both north and south of the factory and several nearby. It could make sense to have the ships serve two disjunct sets, or serve fish farms either in the north or south and share the supply of fish farms near the factory.

As a strategic tool, the model can help plan transportation in an efficient way, in order to avoid expensive and unnecessary investments in more ships, storage or production capacity. It can also aid in decision making on further investments. The model can be run with a variable number of ships or factories and sensitivity analysis can be used to explore the gain of increased storage capacities for factories, fish farms and ships. In this thesis, we have focused on the operational and tactical applications of the model.

5.2 Arc-Load Formulation

Our original arc-load model, as presented in Ivarsøy and Solhaug (2013), used binary variables to indicate where a ship ends its voyage. In our new arc-load model we have removed these variables in favor of adding dummy nodes. After a ship's last visit, it is forced to travel to its end node, $d(v)$. This simple change turned out to give a significant reduction in the time required to achieve good solutions, compared to the previous formulation. Ships have an initial position $o(v)$, which is either a visit to a factory or a fish farm. Other adjustments include improvements of loading constraints and changes in the objective function. The formulation is illustrated in Figure 5.1. The complete mathematical formulation can also be found in Appendix A.

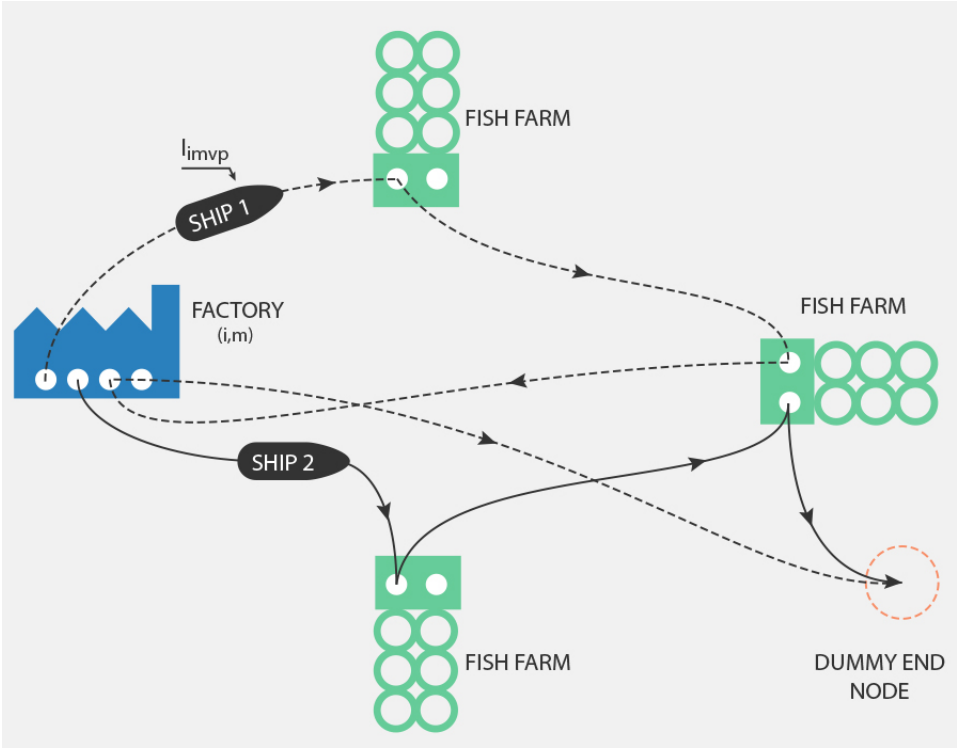


Figure 5.1: Illustration of the arc-load formulation.

5.2.1 Definitions

Sets

\mathcal{N}^P	Set of factories
\mathcal{N}^C	Set of fish farms
\mathcal{M}_i	Set of visit numbers for location i
\mathcal{V}	Set of ships
\mathcal{S}^P	Set of factory visits (i, m) where $i \in \mathcal{N}^P$ and $m \in \mathcal{M}_i$
\mathcal{S}^C	Set of fish farm visits (i, m) where $i \in \mathcal{N}^C$ and $m \in \mathcal{M}_i$
\mathcal{S}	Set of visits, $\mathcal{S} = \mathcal{S}^P \cup \mathcal{S}^C$
\mathcal{S}_v	Set of feasible visits for ship v , $\mathcal{S}_v = \mathcal{S} \cup \{d(v)\}$
\mathcal{P}	Set of products
\mathcal{D}	Set of days within the planning horizon

Indices

i, j	Locations
$o(v)$	Start node of ship v
$d(v)$	Dummy end node of ship v
m, n	Visit numbers
v	Ships
p	Products
d	Days

Parameters

B	Penalty cost for stock levels below the safety stock level	[NOK/hour]
C_{ij}	Transportation cost for sailing from location i to location j	[NOK]
E_p	Unit cost of buying product p externally	[NOK/ton]
E_i^T	Transportation cost for external feed delivery to fish farm i	[NOK]
L_{vp}^0	Initial load of product p on ship v	[tons]
K_v^{MAX}	Maximum capacity for ship v	[tons]
Q_{ip}^{MIN}	Minimum unloading quantity of product p for fish farm i	[tons]
S_i^0	Initial inventory level at factory i	[tons]
S_{ip}^0	Initial inventory level of product p at fish farm i	[tons]
S_{ip}^{MIN}	Minimum inventory level of product p at fish farm i	[tons]
S_i^{MAX}	Maximum inventory level at location i	[tons]
A_i	Reduction in storage capacity at fish farm i outside working hours	[%]
T^{MAX}	Length of planning period	[hours]
T_{ij}^S	Sailing time from location i to location j	[hours]
T_i^L	Loading or unloading time per ton of feed for location i	[hours/ton]
T_d^{WS}	Start of service hours for day d	[hours]
T_d^{WE}	End of service hours for day d	[hours]
J_i	Location type for location i , 1 for factories and -1 for fish farms	
R_{ip}	Consumption rate of product p at fish farm i	[tons/hour]
P_i	Production rate at factory i	[tons/hour]
H_i	Time between visits to location i	[hours]

Decision variables

x_{imjnv}	1 if ship v sails from visit (i, m) to visit (j, n) , else 0	
y_{im}	1 if visit (i, m) is not made by any ship, else 0	
u_i	1 if fish farm i is supplied internally, 0 if supplied externally	
q_{imvp}	Amount of product p loaded/unloaded by ship v during visit (i, m)	[tons]
l_{imvp}	Amount of product p on board ship v when leaving visit (i, m)	[tons]
s_{im}	Amount of feed in stock at the start of visit $(i, m) \in \mathcal{S}^P$	[tons]
s_{im}^E	Amount of feed in stock at the end of visit $(i, m) \in \mathcal{S}^P$	[tons]
s_{imp}	Amount of product p in stock at the start of visit $(i, m) \in \mathcal{S}^C$	[tons]
s_{imp}^E	Amount of product p in stock at the end of visit $(i, m) \in \mathcal{S}^C$	[tons]
d_{imp}	Amount of product p below S_{ip}^{MIN} at the start of visit $(i, m) \in \mathcal{S}^C$	[tons]
t_{im}	Time for start of service for visit (i, m)	[hours]
t_{im}^E	Time for end of service for visit (i, m) ,	[hours]
σ_{imd}	1 if visit $(i, m) \in \mathcal{S}^C$ is within service hours on day d , else 0	

5.2.2 Model formulation

Objective function

$$\begin{aligned}
 \min z = & \sum_{(i,m) \in \mathcal{S}} \sum_{(j,n) \in \mathcal{S}} \sum_{v \in \mathcal{V}} C_{ij} x_{imjnv} + T^{MAX} \sum_{i \in \mathcal{N}^C} \sum_{p \in \mathcal{P}} E_p R_{ip} (1 - u_i) \\
 & + \sum_{i \in \mathcal{N}^C} E_i^T (1 - u_i) + \sum_{(i,m) \in \mathcal{S}^C} \sum_{p \in \mathcal{P}} B \frac{d_{imp}}{R_{ip}}
 \end{aligned} \tag{5.1}$$

The objective function (5.1) minimizes transportation costs, costs of buying feed externally and penalty costs related to low stock levels. A transportation cost, C_{ij} is included for each arc used. External feed costs are added as two terms, a margin per unit of feed, E_p , and a transportation cost, E_i^T . For fish farms with special feed needs, the transportation cost of external delivery is ignored, since they need external supply regardless of the decisions made by the model. The penalty cost, B , is added for each unit of time that a fish farm is below the safety stock level. The time is given by the fraction of underage over the rate of feed depletion from stock.

Routing constraints

$$\sum_{(j,n) \in \mathcal{S}_v} x_{o(v)jnv} = 1 \quad v \in \mathcal{V} \tag{5.2}$$

Constraints (5.2) ensure that each ship leaves its initial visit. A ship can travel directly from its initial visit to its dummy node, meaning the ship is not used.

$$\sum_{(j,n) \in \mathcal{S}} x_{jnimv} - \sum_{(j,n) \in \mathcal{S}_v} x_{imjnv} = 0 \quad (i, m) \in \mathcal{S} \setminus \{o(v)\}, v \in \mathcal{V} \quad (5.3)$$

Constraints (5.3) ensure that all subsequent visits to locations have equal ingoing and outgoing flow.

$$\sum_{(j,n) \in \mathcal{S}} x_{jnd(v)v} = 1 \quad v \in \mathcal{V} \quad (5.4)$$

Constraints (5.4) ensure that each ship ends in its designated end node.

$$\sum_{(j,n) \in \mathcal{S}_v} \sum_{v \in \mathcal{V}} x_{imjnv} = 1 - y_{im} \quad (i, m) \in \mathcal{S} \quad (5.5)$$

Constraints (5.5) set the value of y_{im} , indicating that a visit is not made.

$$y_{im} - y_{im-1} \geq 0 \quad (i, m) \in \mathcal{S} | m > 1 \quad (5.6)$$

By adding constraints (5.6) to the model formulation, we ensure that only the smallest subsequent visit numbers are used. These constraints are needed for the solutions to make sense, meaning that higher visit numbers are not used unless the preceding number is also used. In addition, these constraints are important because they reduce the number of symmetric solutions. They eliminate combinations of visit numbers that appear different to the computer, but in reality give the same solution.

$$u_i = 1 - y_{i1} \quad i \in \mathcal{N}^C \quad (5.7)$$

Constraints (5.7) set the variable u_i to indicate if a fish farm is served by the internal factories during the planning horizon. It is simply the opposite of the variable y_{i1} , which indicates whether the first visit to a location is made or not. All fish farms are assumed to need at least one delivery during the planning horizon and if the first visit to a fish farm is not made, external feed costs apply. To reduce the number of variables, u_i is not included in the implementation of the model, but is included here for increased readability.

Loading and unloading constraints

$$L_{vp}^0 + J_i q_{o(v)vp} = l_{o(v)vp} \quad v \in \mathcal{V}, p \in \mathcal{P} \quad (5.8)$$

Constraints (5.8) set the load for each ship after their initial visit equal to the sum of its initial load and the quantity loaded or unloaded.

$$x_{imjnv}(l_{imvp} + J_j q_{jnv} - l_{jnv}) = 0 \quad (i, m) \in \mathcal{S}, (j, n) \in \mathcal{S}_v \setminus \{o(v)\}, v \in \mathcal{V}, p \in \mathcal{P} \quad (5.9)$$

For all subsequent visits, ship loads are updated according to constraints (5.9). This equation is obviously nonlinear and is not suited for direct implementation in our solver of choice. Constraints (5.9) are therefore linearized and replaced with constraints (6.1) and (6.2), as presented in Chapter 6.

$$\sum_{p \in \mathcal{P}} l_{imvp} \leq \sum_{(j,n) \in \mathcal{S}_v} K_v^{MAX} x_{imjnv} \quad (i, m) \in \mathcal{S}^P, v \in \mathcal{V} \quad (5.10)$$

Constraints (5.10) limit the outgoing load from a factory to not exceed a ship's capacity. If a visit is not made by ship v , the corresponding load variable is forced to zero.

$$\sum_{p \in \mathcal{P}} l_{imvp} \leq \sum_{(j,n) \in \mathcal{S}_v} K_v^{MAX} x_{imjnv} - \sum_{p \in \mathcal{P}} q_{imvp} \quad (i, m) \in \mathcal{S}^C, v \in \mathcal{V} \quad (5.11)$$

Constraints (5.11) limit the outgoing load from a fish farm to be no larger than a ship's capacity less the quantity unloaded. In our original model in Ivarøy and Solhaug (2013), loads were only constrained by the ship capacity. By also subtracting the quantity, we achieve a tighter constraint. If a visit is not made by ship v , the corresponding load variable is forced to zero.

$$\sum_{p \in \mathcal{P}} q_{imvp} \leq \sum_{(j,n) \in \mathcal{S}_v} S_i^{MAX} x_{imjnv} \quad (i, m) \in \mathcal{S}^C, v \in \mathcal{V} \quad (5.12)$$

Constraints (5.12) limit the quantity unloaded to be less than or equal to a fish farm's maximum storage capacity. This is tighter than simply using ship capacity as a bound, since this is much larger than the storage capacity at any fish farm. The constraints are aggregated over product types. With product specific storage capacities, the formulation would have become tighter. If a visit is not made by ship v , the corresponding quantity-variable is forced to zero.

$$\sum_{(j,n) \in \mathcal{S}_v} Q_{ip}^{MIN} x_{imjnv} \leq q_{imvp} \quad (i, m) \in \mathcal{S}^C, v \in \mathcal{V}, p \in \mathcal{P} \quad (5.13)$$

Visits to fish farms are required to be a certain amount of time apart, ensured by constraints (5.35). We can calculate the amount of feed consumed during this time and use it as a lower bound on the unloading quantity, as shown in constraints (5.13).

$$\sum_{v \in \mathcal{V}} \sum_{p \in \mathcal{P}} q_{imvp} = s_{im}(1 - y_{im}) \quad (i, m) \in \mathcal{S}^P \quad (5.14)$$

Marine Harvest has specified that a ship visiting a factory should always clear the stock. Constraints (5.14) ensure that the loaded quantity equals the current stock level, if the visit is made. Since their factory and ships have equal storage capacities, the entire factory stock can be loaded, assuming that ships arrive empty at the factory. For a problem with smaller ship capacities, these constraints should be eliminated and replaced with constraints equal to (5.12) and (5.13) for factory visits. The constraints are linearized and replaced with constraints (6.3) and (6.4) in Chapter 6.

Inventory constraints

All stock variables are required to be non-negative. If a stock variable is below the safety stock level, an underage variable is set. If a fish farm is supplied externally, its stock variables will remain at the fish farm's initial stock level. This is because constraints (5.30) ensure that time variables related to visits to this fish farm are set to zero, and the stock variables will not be updated.

$$S_i^0 + P_i t_{im} = s_{im} \quad (i, m) \in \mathcal{S}^P | m = 1 \quad (5.15)$$

$$S_{ip}^0 - R_{ip} t_{im} = s_{imp} \quad (i, m) \in \mathcal{S}^C | m = 1, p \in \mathcal{P} \quad (5.16)$$

Equations (5.15) and (5.16) set the stock level at the start of the first visit to factories and fish farms as the initial stock plus the amount produced or consumed before the first visit. The inventory constraints should ideally be written in a more compressed form, using the location type parameter J_i . Since product types are considered only at the fish farms, two sets of inventory constraints are needed, one without and one with the p index, for factories and fish farms respectively.

$$s_{im} + P_i(t_{im}^E - t_{im}) - \sum_{v \in \mathcal{V}} \sum_{p \in \mathcal{P}} q_{imvp} = s_{im}^E \quad (i, m) \in \mathcal{S}^P \quad (5.17)$$

$$s_{imp} - R_{ip}(t_{im}^E - t_{im}) + \sum_{v \in \mathcal{V}} q_{imvp} = s_{imp}^E \quad (i, m) \in \mathcal{S}^C, p \in \mathcal{P} \quad (5.18)$$

Constraints (5.17) ensure that the stock at the end of a factory visit equals the stock at the start of the visit plus the amount of feed produced and less the amount loaded during the visit. Constraints (5.18) ensure that the stock at the end of a fish farm visit equals the stock at the start of the visit less the amount of feed consumed and plus the amount unloaded during the visit. The end stock variables s_{im}^E are included for readability and are not needed in the implementation as they can be expressed using the start stock variables s_{im} .

$$s_{i(m-1)}^E + P_i(t_{im} - t_{i(m-1)}^E) = s_{im} \quad (i, m) \in \mathcal{S}^P | m > 1 \quad (5.19)$$

$$s_{i(m-1)p}^E - R_{ip}(t_{im} - t_{i(m-1)}^E) = s_{imp} \quad (i, m) \in \mathcal{S}^C | m > 1, p \in \mathcal{P} \quad (5.20)$$

Constraints (5.19) and (5.20) relate the stock at the end of a visit to the stock at the start of the next visit by considering the production or consumption that takes place between the visits. Looking at (5.17) and (5.18) together with (5.19) and (5.20), it may seem like too many constraints are created and that it would have been sufficient with one set of inventory update constraints. However, this would enable a ship to load more than available at a factory or unload more than can be stored at a fish farm, and then restore the variables to feasible levels before the next visit. Therefore, we have one set of constraints for inventory update during visits and one for inventory update between visits.

$$s_{im} \leq S_i^{MAX} \quad (i, m) \in \mathcal{S}^P \quad (5.21)$$

$$\sum_{p \in \mathcal{P}} s_{imp}^E \leq (1 - A_i)S_i^{MAX} + A_i S_i^{MAX} \sum_{d \in \mathcal{D}} \sigma_{imd} \quad (i, m) \in \mathcal{S}^C \quad (5.22)$$

Constraints (5.21) and (5.22) ensure that the stock level at the start of service in a factory and at the end of service at a fish farm does not exceed the storage capacities. Since loading and unloading rates are higher than production and consumption rates, only these variables need to be constrained by upper limits. Then, the stock variable at the end of service in a factory and the stock variable at the start of service in a fish farm will never be above the maximum stock level. Constraints (5.22) take into account that the storage capacity is reduced by A_i if a fish farm visit starts outside service hours.

$$S_{ip}^{MIN} u_i \leq s_{imp} + d_{imp} \quad (i, m) \in \mathcal{S}^C, p \in \mathcal{P} \quad (5.23)$$

Constraints (5.23) ensure that the sum of the inventory level and the underage stays above the safety stock level. If a fish farm is supplied externally, the constraint is not binding. Note that constraints (5.23) only apply to fish farms. The lower inventory limit at a factory is zero, ensured by non-negativity constraints on the factory stock variables.

$$d_{imp} \leq S_{ip}^{MIN} u_i \quad (i, m) \in \mathcal{S}^C, p \in \mathcal{P} \quad (5.24)$$

For fish farms served internally, the underage can not be larger than the minimum stock level, as stated by constraints (5.24). If a fish farm is supplied externally, the underage is forced to zero.

$$s_{im}^E + P_i(T^{MAX} - t_{im}^E) \leq S_i^{MAX} \quad (i, m) \in \mathcal{S}^P | m = |\mathcal{M}_i| \quad (5.25)$$

Constraints (5.25) are added to ensure a feasible inventory level for factories at the end of the planning period. This is done by ensuring that the sum of the stock after the last visit and the production during the remaining time is below the storage capacity.

$$S_{ip}^{MIN} u_i + R_{ip} T^{MAX} u_i - R_{ip} t_{im}^E \leq s_{imp}^E \quad (i, m) \in \mathcal{S}^C | m = |\mathcal{M}_i|, p \in \mathcal{P} \quad (5.26)$$

Constraints (5.26) are added to ensure feasible inventory levels for fish farms at the end of the planning period. For fish farms supplied by the internal factories, the inventory level at the end of the last visit must be enough to stay above the safety stock level until the end of the planning period. If an external supplier is given responsibility for a fish farm's feed supply, the constraint is not binding, since all the time variables for externally supplied fish farms are forced to zero by constraints (5.30).

Timing constraints

$$t_{im} + \sum_{v \in \mathcal{V}} \sum_{p \in \mathcal{P}} T_i^L q_{imvp} = t_{im}^E \quad (i, m) \in \mathcal{S} \quad (5.27)$$

In order to relate the start and end time of a visit, constraints (5.27) are added. They ensure that a visit ends when the loading or unloading operation has finished. If no feed is loaded or unloaded, as when a visit is not made, the end time is set equal to the start time. The end time variables t_{im}^E are included for readability and are not needed for implementation as they can be expressed using the start time variables t_{im} .

$$\sum_{v \in \mathcal{V}} x_{imjnv} (t_{im}^E + T_{ij}^S - t_{jn}) \leq 0 \quad (i, m), (j, n) \in \mathcal{S} \quad (5.28)$$

Constraints (5.28) ensure consistency in timing of visits. If a ship sails directly between two visits, the start time of the next visit can not be earlier than the end time of the previous visit plus the sailing time between the two locations. By modeling the constraint as an inequality, waiting on arrival is allowed. The constraints are linearized and replaced by constraints (6.5), as shown in Chapter 6.

$$t_{im}^E \leq T^{MAX} \quad (i, m) \in \mathcal{S} \quad (5.29)$$

Constraints (5.29) ensure that visits cannot end later than the end of the planning period. We assume that we start at time zero and so the end of the planning period is equal to the length of the planning period given by T^{MAX} . It is easy to include a different start time, T^{START} , where the end time of the last visit cannot be later than $T^{START} + T^{MAX}$.

$$t_{im} \leq T^{MAX} u_i \quad (i, m) \in \mathcal{S}^C \quad (5.30)$$

Constraints (5.30) ensure that if a fish farm is supplied externally, the start time of visits to this fish farm are set to zero.

$$y_{im}(t_{im} - t_{i(m-1)}^E) = 0 \quad (i, m) \in \mathcal{S} | m > 1 \quad (5.31)$$

Constraints (5.31) ensure that if a visit is not made, the start time should be set equal to the end time of the preceding visit number. This creates artificial start times so that the model can not bypass constraints (5.25) and (5.26) by setting a larger visit end time on the last possible visit. The constraints are linearized in Chapter 6 and replaced by constraints (6.6) and (6.7).

$$T_d^{WS} - T^{MAX}(1 - \sigma_{imd} + y_{im}) \leq t_{im} \leq T_d^{WE} + T^{MAX}(1 - \sigma_{imd} + y_{im}) \quad (5.32)$$

$$(i, m) \in \mathcal{S}^C, d \in \mathcal{D}$$

$$\sum_{d \in \mathcal{D}} \sigma_{imd} \leq 1 \quad (i, m) \in \mathcal{S}^C \quad (5.33)$$

Constraints (5.32) ensure that if a fish farm visit starts outside the given service time windows, σ_{imd} is set to 0. Then, the storage capacity is reduced by A_i in constraints (5.22). If a visit is not made, the constraint is not binding. For each visit, at most one of the time window variables can be set to 1, ensured by constraints (5.33). If the extra capacity is not needed, the variables are not necessarily set to 1, even if the visit is within service hours.

$$\sum_{d \in \mathcal{D}} \sigma_{imd} \geq y_{im} \quad (i, m) \in \mathcal{S}^C \quad (5.34)$$

Visits that are not made should all have σ_{imd} equal to 1, ensured by constraints (5.34). This is to ensure that the upper inventory constraints (5.22) for these visits are not stricter than the upper inventory constraints for the preceding visit numbers that are used.

$$t_{im} + H_i(1 - y_{i(m+1)}) \leq t_{im+1} \quad (i, m) \in \mathcal{S} \quad (5.35)$$

Marine Harvest wants deliveries to their fish farms to be evenly spread throughout the planning horizon. Due to production constraints, the factories can not provide full ship loads every day and visits should be separated for factories as well. By adding constraints (5.35), visits are separated by at least H_i hours for both factories and fish farms.

Variable constraints

$$x_{imjnv} \in \{0, 1\} \quad (i, m) \in \mathcal{S}, (j, n) \in \mathcal{S}_v, v \in \mathcal{V} \quad (5.36)$$

$$y_{im} \in \{0, 1\} \quad (i, m) \in \mathcal{S} \quad (5.37)$$

$$u_i \in \{0, 1\} \quad i \in \mathcal{N}^C \quad (5.38)$$

$$\sigma_{imd} \in \{0, 1\} \quad (i, m) \in \mathcal{S}^C, d \in \mathcal{D} \quad (5.39)$$

$$q_{imvp} \geq 0 \quad (i, m) \in \mathcal{S}, v \in \mathcal{V}, p \in \mathcal{P} \quad (5.40)$$

$$l_{imvp} \geq 0 \quad (i, m) \in \mathcal{S}, v \in \mathcal{V}, p \in \mathcal{P} \quad (5.41)$$

$$s_{im} \geq 0 \quad (i, m) \in \mathcal{S}^P \quad (5.42)$$

$$s_{im}^E \geq 0 \quad (i, m) \in \mathcal{S}^P \quad (5.43)$$

$$s_{imp} \geq 0 \quad (i, m) \in \mathcal{S}^C, p \in \mathcal{P} \quad (5.44)$$

$$s_{imp}^E \geq 0 \quad (i, m) \in \mathcal{S}^C, p \in \mathcal{P} \quad (5.45)$$

$$d_{imp} \geq 0 \quad (i, m) \in \mathcal{S}^C, p \in \mathcal{P} \quad (5.46)$$

$$t_{im} \geq 0 \quad (i, m) \in \mathcal{S} \quad (5.47)$$

$$t_{im}^E \geq 0 \quad (i, m) \in \mathcal{S} \quad (5.48)$$

5.3 Arc-Flow Formulation

To further develop our basic arc-load model, we tried two reformulations, as in Agra et al. (2013a). In our first reformulation we replaced the load variables l_{imvp} with l_{imjnpv} , indicating the flow on each arc instead of the load when leaving a visit. This enables us to avoid the nonlinear load update constraints (5.9) and achieve tighter constraints relating routing variables and load variables. The arc-flow formulation is similar to the arc-load formulation except for constraints including the new variables l_{imjnpv} . The loading and unloading constraints (5.8) - (5.11) are replaced by the new constraints presented below. The formulation is illustrated in Figure 5.2 and the complete mathematical formulation can be found in Appendix B.

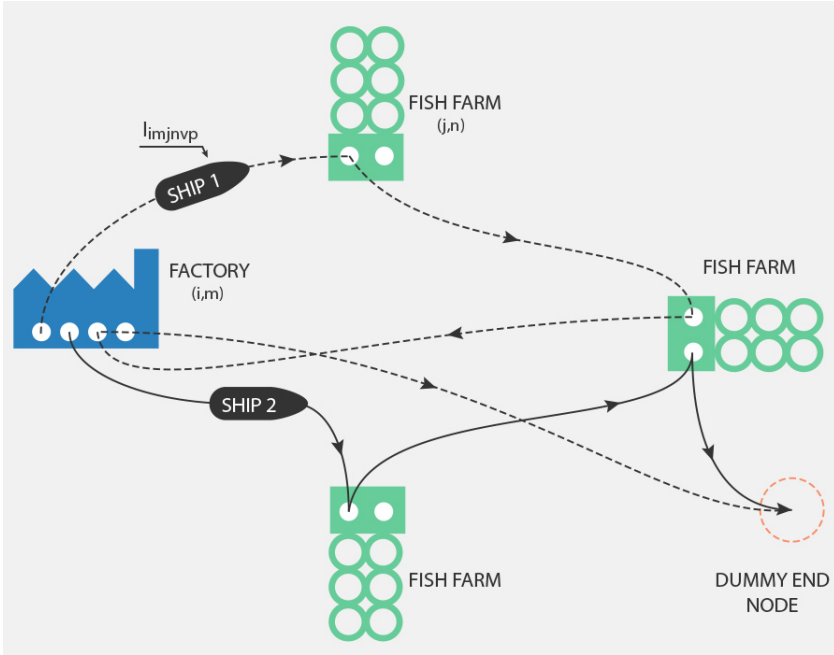


Figure 5.2: Illustration of the arc-flow formulation.

5.3.1 Definitions

Decision variables

l_{imjnv} Amount of product p on board ship v when traveling on arc (i, m, j, n) [tons]

5.3.2 Model formulation

Loading and unloading constraints

$$L_{vp}^0 + J_i q_{o(v)vp} = \sum_{(j,n) \in \mathcal{S}_v} l_{o(v)jnv} \quad v \in \mathcal{V}, p \in \mathcal{P} \quad (5.49)$$

Constraints (5.49) set the load on board a ship after its initial visit equal to the sum of its initial load and the quantity loaded or unloaded. They replace constraints (5.8).

$$\sum_{(j,n) \in \mathcal{S}} l_{jnimvp} + J_i q_{imvp} - \sum_{(j,n) \in \mathcal{S}_v} l_{imjnv} = 0 \quad (i, m) \in \mathcal{S} \setminus \{o(v)\}, v \in \mathcal{V}, p \in \mathcal{P} \quad (5.50)$$

For all subsequent visits, ship loads are updated according to constraints (5.50). These are linear, as opposed to the previous load update constraints (5.9).

$$\sum_{p \in \mathcal{P}} l_{imjnv} \leq K_v^{MAX} x_{imjnv} \quad (i, m) \in \mathcal{S}, (j, n) \in \mathcal{S}_v, v \in \mathcal{V} \quad (5.51)$$

Constraints (5.51) limit a load to be equal to or less than a ship's capacity and replace constraints (5.10) and (5.11). If the arc (i, m, j, n) is not used by ship v , the corresponding load variable is forced to zero.

Variable constraints

$$l_{imjnv} \geq 0 \quad (i, m) \in \mathcal{S}, (j, n) \in \mathcal{S}_v, v \in \mathcal{V}, p \in \mathcal{P} \quad (5.52)$$

5.4 Multi-Commodity Flow Formulation

In order to further tighten the formulation, we use the even more detailed load variables $l_{imjnkoup}$. These include information about which visit (k, o) a load is destined to, where (k, o) is a visit to a flow receiver. Flow receivers include all fish

farms and the dummy end nodes $d(v)$. If a ship has load destined to its dummy end node, this load remains on board the ship at the end of the planning horizon. As in the arc-flow model, we avoid the nonlinear load update constraints (5.9). The constraints linking routing and load variables are further tightened, using the extra information provided by the new variables. However, we also get a dramatic increase in the number of variables and constraints. To avoid summing over a large number of x_{imjnv} variables in the many loading constraints, we have created a new variable, w_{imv} , indicating whether a visit (i, m) is made by ship v .

The multi-commodity flow model is similar to the arc-load and arc-flow models except for constraints including the new variables $l_{imjnkovp}$ and w_{imv} . In this section we present the variables and constraints in the multi-commodity flow model that differs from the arc-load and arc-flow models. A routing constraint to set w_{imv} according to x_{imjnv} is added, and the loading and unloading constraints (5.8) - (5.14) are replaced by the new constraints presented below. The formulation is illustrated in Figure 5.3 and the complete mathematical formulation can be found in Appendix C.

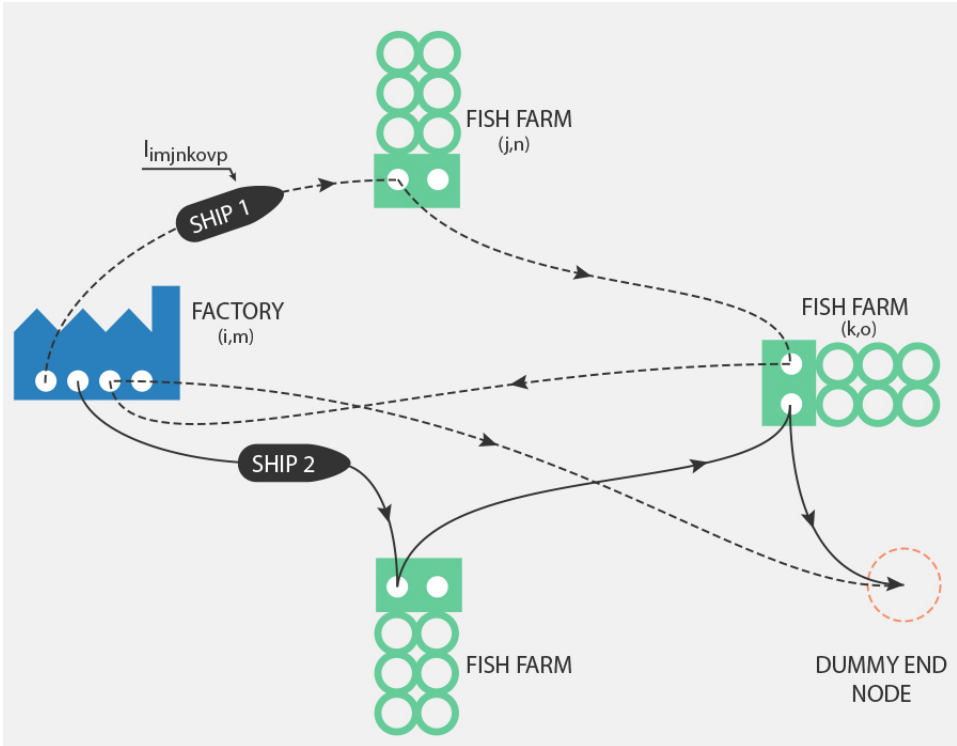


Figure 5.3: Illustration of the multi-commodity flow formulation.

5.4.1 Definitions

Sets

\mathcal{S}_v^F Set of visits to flow receivers for ship v , $\mathcal{S}_v^F = \mathcal{S}^C \cup \{d(v)\}$

Decision variables

w_{imv} 1 if ship v makes visit (i, m) , else 0
 $l_{imjnkovp}$ Amount of product p on board ship v on arc (i, m, j, n) , [tons] destined for visit (k, o)

Indices

k Flow receivers
 o Visit numbers

5.4.2 Model formulation

Routing constraints

$$\sum_{(j,n) \in \mathcal{S}_v} x_{imjnv} = w_{imv} \quad (i, m) \in \mathcal{S}, v \in \mathcal{V} \quad (5.53)$$

Constraints (5.53) set the variable w_{imv} according to whether ship v makes the visit (i, m) .

Loading and unloading constraints

$$L_{vp}^0 + J_i q_{o(v)vp} = \sum_{(j,n) \in \mathcal{S}_v} \sum_{(k,o) \in \mathcal{S}_v^F} l_{o(v)jnkovp} \quad v \in \mathcal{V}, p \in \mathcal{P} \quad (5.54)$$

Constraints (5.54) set the total load on board a ship after its initial visit equal to the sum of its initial load and the quantity loaded or unloaded, and replace constraints (5.8).

$$\sum_{(j,n) \in \mathcal{S}} \sum_{(k,o) \in \mathcal{S}_v^F} l_{jnimkovp} + J_i q_{imvp} - \sum_{(j,n) \in \mathcal{S}_v} \sum_{(k,o) \in \mathcal{S}_v^F} l_{imjnkovp} = 0 \quad (5.55)$$

$$(i, m) \in \mathcal{S} \setminus \{o(v)\}, v \in \mathcal{V}, p \in \mathcal{P}$$

For all subsequent visits, ship loads are updated according to constraints (5.55). These are linear, as opposed to the original load update constraints (5.9).

Constraints (5.56)- (5.60) replace constraints (5.10) -(5.12).

$$\sum_{(j,n) \in \mathcal{S}} l_{jnimivp} - q_{imvp} = 0 \quad (i, m) \in \mathcal{S}^C, v \in \mathcal{V}, p \in \mathcal{P} \quad (5.56)$$

$$\begin{aligned} \sum_{(j,n) \in \mathcal{S}_v} l_{imjnkovp} - \sum_{(j,n) \in \mathcal{S}} l_{jnimkovp} &= 0 \\ (i, m) \in \mathcal{S}^C, (k, o) \in \mathcal{S}_v^F, v \in \mathcal{V}, p \in \mathcal{P} \end{aligned} \quad (5.57)$$

When a load reaches a fish farm that is also its destination, constraints (5.56) ensure that this load is equal to the quantity unloaded. When a load reaches a fish farm that is not its destination, constraints (5.57) set the incoming load equal to the outgoing load.

$$\begin{aligned} \sum_{(j,n) \in \mathcal{S}_v} l_{imjnkovp} &\leq \sum_{(j,n) \in \mathcal{S}} l_{jnimkovp} + q_{imvp} \\ (j, n) \in \mathcal{S}^P, (k, o) \in \mathcal{S}_v^F, v \in \mathcal{V}, p \in \mathcal{P} \end{aligned} \quad (5.58)$$

$$\begin{aligned} \sum_{(j,n) \in \mathcal{S}} l_{jnimkovp} &\leq \sum_{(j,n) \in \mathcal{S}_v} l_{imjnkovp} \\ (i, m) \in \mathcal{S}^P, (k, o) \in \mathcal{S}_v^F, v \in \mathcal{V}, p \in \mathcal{P} \end{aligned} \quad (5.59)$$

Constraint (5.58) ensure that flows leaving the factory are not larger than the flows entering the factory plus the quantity loaded. Constraints (5.59) ensure that flows entering the factory must be smaller than or equal to flows leaving the factory.

$$\sum_{(k,o) \in \mathcal{S}_v^F} \sum_{p \in \mathcal{P}} l_{imjnkovp} \leq K_v^{MAX} x_{imjnv} \quad (i, m) \in \mathcal{S}, (j, n) \in \mathcal{S}_v, v \in \mathcal{V} \quad (5.60)$$

Constraints (5.60) limit the total load on an arc to be below a ship's capacity. If an arc is not used by ship v , the corresponding load-variable is forced to zero.

$$\begin{aligned} \sum_{p \in \mathcal{P}} l_{imjnkovp} &\leq S_k^{MAX} x_{imjnv} \\ (i, m) \in \mathcal{S}, (j, n) \in \mathcal{S}_v, (k, o) \in \mathcal{S}_v^F, v \in \mathcal{V} \end{aligned} \quad (5.61)$$

$$\sum_{p \in \mathcal{P}} l_{imjnkovp} \leq S_k^{MAX} w_{kov} \quad (i, m) \in \mathcal{S}, (j, n) \in \mathcal{S}_v, (k, o) \in \mathcal{S}^C, v \in \mathcal{V} \quad (5.62)$$

Constraints (5.61) and (5.62) limit a load to be equal to or less than the destination's storage capacity. If an arc is not used by ship v or if the destination of the load is not visited by ship v , the corresponding load variable is forced to zero. Both constraints (5.61) and (5.62) are aggregated over product types. As mentioned for constraints (5.12), the formulation would have been tighter if storage capacities were product specific.

$$Q_{ip}^{MIN} w_{imv} \leq q_{imvp} \quad (i, m) \in \mathcal{S}^C, v \in \mathcal{V}, p \in \mathcal{P} \quad (5.63)$$

$$\sum_{p \in \mathcal{P}} q_{imvp} = s_{im} w_{imv} \quad (i, m) \in \mathcal{S}^P, v \in \mathcal{V} \quad (5.64)$$

Constraints (5.63) and (5.64) are similar to constraints (5.13) and (5.14), except that the new variable w_{imv} is used instead of the x_{imjnv} and y_{im} variables. Constraints (5.64) are nonlinear and replaced with linearizations similar to (6.3) and (6.4) from Chapter 6.

Variable constraints

$$w_{imv} \in \{0, 1\} \quad (i, m) \in \mathcal{S}, v \in \mathcal{V} \quad (5.65)$$

$$l_{imjnkovp} \geq 0 \quad (i, m) \in \mathcal{S}, (j, n) \in \mathcal{S}_v, (k, o) \in \mathcal{S}_v^F, v \in \mathcal{V}, p \in \mathcal{P} \quad (5.66)$$

Chapter 6

Implementation

The three formulations described in Chapter 5 are written in Mosel and implemented in Xpress-IVE Version 1.24.00, 64 bit. We have run all tests on computers in NTNU's HPC-cluster, Solstorm. The Xpress Optimizer Version of these computers is 25.01.05. In the cluster, computers of the same type are located in the same rack. Computers in rack 3 are faster than computers in rack 5, which have more memory. Since many of our model runs need a lot of memory, we have mostly used computers in rack 5. For one of our parallelization frameworks, speed is more important, and for this we have used rack 3. The specifications for computers in rack 3 and 5 are listed in Table 6.1.

	Rack 3	Rack 5
Operating system	CentOS Linux	CentOS Linux
Processor	2.4GHz AMD Opteron 2431	2.2GHz AMD Opteron 6274
Number of processors	2	4
Cores per processor	6	16
Memory (RAM)	24 GB	128GB

Table 6.1: Specifications of computers in Solstorm.

In Section 6.1, we describe the test cases we have used for testing the three formulations. In Sections 6.2 and 6.3 we present adjustments and simplifications we have made to be able to implement linear and solvable models. Section 6.4 describes attempts to improve the models, such as tightening of constraints and valid inequalities. Lastly, Section 6.5 presents our work on parallelizing the branch-and-bound search by creating two frameworks.

6.1 Test Cases

As mentioned in Chapter 3, we have implemented less general models than the formulations given in Chapter 5. We look at the current situation with two homogeneous ships and one factory and we will test our models using three test cases with 20, 40 and 60 fish farms. We use realistic data provided by Marine Harvest for Region North, Mid and West during high season. For the test cases with 20 and 40 fish farms, the production rate and the storage capacity of the factory and ships are downscaled to one-third and two-thirds, respectively. The test case with 60 fish farms use full-scaled values and includes all fish farms from Region North, Mid and West that have biomass during the period from which we have calculated our demand rates. Since our demand data is from the high season, we have excluded fish farms that have smolt release at this time, because smolts released at fall are smaller and require special feed not supplied by the factory.

The production rate is an estimation given by Marine Harvest, since the factory is not yet operating. It will vary throughout the year, but kept constant at an estimated maximum level of 45 tons per hour during high season. Maximum storage capacity of the ships, factory and fish farms are known. The ships and the factory have an equal capacity of 3000 tons. The hourly consumption rates for fish farms are calculated from historical monthly data. An overview of data related to the factory, ships and fish farms for each test case can be found in Appendix D.

The safety stock level is set to one feed day for each fish farm. The service hours at fish farms are from 8:00 a.m. to 4:00 p.m. every day. Outside working hours, storage capacity is reduced with 10% for all fish farms. Visits to the factory and fish farms should be separated by 24 hours. This enables us to calculate minimum unload quantities for fish farms by finding the amount of feed consumed during this time. The loading rate is 270 tons per hour, while the unloading rate is 180 tons per hour.

Travel distances between all locations are estimations provided by Marine Harvest and travel times are calculated using a constant speed of 13 knots. The transportation cost is 1600 NOK per hour, calculated with help from Ivar Christian Ulvan (Egil Ulvan Rederi, 2014) and Océane Balland (Balland, 2014). The cost is based on LNG consumption per hour and is dependent on LNG prices. The cost of external supply is divided into two parts, a fixed transportation cost of 4400 NOK for each fish farm and a profit margin of 250 NOK per ton of feed. The external transportation cost is parameterized relative to the internal transportation cost, while the profit margin is an estimation we have made using Marine Harvest's current cost of standard feed. The penalty cost per hour of being below the safety stock level is parameterized to make cost-effective plans, while avoiding low stock levels. It has been set to 400 NOK per hour and is equal for all fish farms. Since all costs are rough estimations made by us, they should not be interpreted as real cost values. More detailed calculations can be found in Appendix E

A planning horizon of two weeks resulted in a too large problem size. Therefore,

we have decided to use a planning period of 10 days. This allows us to plan for the coming week, while also accounting for three more days of production and consumption. To be able to solve the problem, initial stock levels must be set in accordance with the planning period. Fish farms have initial stock levels of six feed days, since this was the lowest level that could be used with a planning period of 10 days. If initial stock levels are lower, the solver is not able to find any feasible solutions after running for 10 hours. The initial stock level at the factory is set to half capacity. Both ships are assumed to start at the factory and are initially empty.

6.2 Model Adjustments

6.2.1 Linearizing constraints

In order to solve the models with Xpress-Optimizer, we have linearized the nonlinear constraints from Chapter 5 by using the Big M method (Griva et al., 2009).

$$l_{imvp} + J_j q_{jnv} - l_{jnv} + K_v^{MAX} x_{imjnv} \leq K_v^{MAX} \quad (i, m) \in \mathcal{S}, (j, n) \in \mathcal{S}_v \setminus \{o(v)\}, v \in \mathcal{V}, p \in \mathcal{P} \quad (6.1)$$

$$l_{imvp} + J_j q_{jnv} - l_{jnv} - K_v^{MAX} x_{imjnv} \geq -K_v^{MAX} \quad (i, m) \in \mathcal{S}, (j, n) \in \mathcal{S}_v \setminus \{o(v)\}, v \in \mathcal{V}, p \in \mathcal{P} \quad (6.2)$$

Constraints (6.1) and (6.2) linearize constraints (5.9). Since K_v^{MAX} is the largest value that $l_{imvp} + J_j q_{jnv} - l_{jnv}$ can take and $-K_v^{MAX}$ is the smallest, the constraints are only binding if an arc is used.

$$\sum_{v \in \mathcal{V}} \sum_{p \in \mathcal{P}} q_{imvp} \leq s_{im} + S_i^{MAX} y_{im} \quad (i, m) \in \mathcal{S}^P \quad (6.3)$$

$$\sum_{v \in \mathcal{V}} \sum_{p \in \mathcal{P}} q_{imvp} \geq s_{im} - S_i^{MAX} y_{im} \quad (i, m) \in \mathcal{S}^P \quad (6.4)$$

Constraints (6.3) and (6.4) linearize constraints (5.14). Since S_i^{MAX} is the largest value that $q_{imvp} + s_{im}$ can take and $-S_i^{MAX}$ is the smallest, the constraints are only binding if a visit made. By replacing y_{im} with w_{imv} , we get the linearizations of constraints (5.64).

$$t_{im}^E + T_{ij}^S \sum_{v \in \mathcal{V}} x_{imjnv} - t_{jn} + T^{MAX} \sum_{v \in \mathcal{V}} x_{imjnv} \leq T^{MAX} \quad (i, m) \in \mathcal{S}, (j, n) \in \mathcal{S} \setminus \{o(v)\} \quad (6.5)$$

Constraints (6.5) linearize constraints (5.28). Since (5.28) is an inequality, only one relation is needed for the linearization. The largest value that $t_{im}^E - t_{jn}$ can take is T^{MAX} and the constraint is only binding if a visit is made.

$$t_{im} - t_{i(m-1)}^E + T^{MAX} y_{im} \leq T^{MAX} \quad (i, m) \in \mathcal{S} | m > 1 \quad (6.6)$$

$$t_{im} - t_{i(m-1)}^E - T^{MAX} y_{im} \geq -T^{MAX} \quad (i, m) \in \mathcal{S} | m > 1 \quad (6.7)$$

Constraints (6.6) and (6.7) linearize constraints (5.31), by using two inequalities to ensure equality when a visit is not made. The largest value that $t_{im} - t_{i(m-1)}^E$ can take is T^{MAX} and the constraint is only binding if a visit is made.

6.2.2 Slack in equality constraints

$$\sum_{v \in \mathcal{V}} \sum_{p \in \mathcal{P}} q_{imvp} - k_{im} \leq s_{im} + S_i^{MAX} y_{im} \quad (i, m) \in \mathcal{S}^P \quad (6.8)$$

$$\sum_{v \in \mathcal{V}} \sum_{p \in \mathcal{P}} q_{imvp} + k_{im} \geq s_{im} - S_i^{MAX} y_{im} \quad (i, m) \in \mathcal{S}^P \quad (6.9)$$

Constraints (6.3) and (6.4) ensure that the quantity loaded equals the current stock level at the factory. Equalities may result in numerical problems for commercial solvers and lead to infeasibility. Therefore, we include a slack variable k_{im} in both constraints and replace them with (6.8) and (6.9). To ensure that the stock is cleared, the slack variable has a large cost term in the objective function.

6.3 Problem Reduction

6.3.1 Simplifications

The implementation is simplified due to lack of data regarding different product types, which forces us to only use aggregated values. Therefore, we are not able to explore the feasibility of solutions regarding stowage of product types on board ships. Also, we have not received information about which fish farms that need external supply of special feed and we will not be able to ignore the transportation cost of external supply for any fish farms.

6.3.2 Sets of fish farms

In order to reduce the number of routing variables, we have split the fish farms into two sets and each ship is fixed to one set. Splitting results in a less flexible model, but it helps reduce the number of symmetric solutions arising from the fact that we have two homogeneous ships. This is also in accordance with Marine Harvest's view of operation, where each ship most likely will serve partly separate areas. In Ivarsøy and Solhaug (2013) we considered two alternative splits, a full split and a partial split. In the first splitting approach, the fish farms farthest north are served by one ship, while the ones farthest south are served by the other and the two sets are disjoint. The partial split includes the fish farms closest to the factory in both sets, since it will not be a significant detour for a ship sailing in either direction to visit them. We consider both the partial and full split to be reasonable simplifications, due to the geographical situation of the fish farms along the Norwegian coast. Results from Ivarsøy and Solhaug (2013) showed that the partial split was the best alternative and we have continued using this. The full split has been considered for the largest instance, as an attempt to reduce the problem size further.

6.3.3 Reduced number of visits

Results in Ivarsøy and Solhaug (2013) showed that most fish farms are only visited once with a planning horizon of 10 days. With initial stock levels of six days, no fish farm requires more than one visit. Since each additional visit number leads to one additional arc variable to all other locations' visit numbers, the variable reduction by setting this limit tight is significant. Reducing the number of visits limits the model flexibility and therefore we allow two visits per fish farm for the smallest test case. For the larger test cases, the number of visits must be reduced to one, in order to achieve solutions within reasonable time limits. If the models were to be run with a longer planning horizon, multiple visits would be necessary to most fish farms.

6.3.4 Aspect of service hours removed

In Ivarsøy and Solhaug (2013) we discovered that the solutions obtained from the model did not seem to utilize the extra capacity obtained when arriving within service hours, since large deliveries are rarely made. If the extra capacity is not needed, the σ_{imd} variables may not be set to 1, even though the visit is within service hours. For the largest test case, we have also tested our formulations without the aspect of working hours, as an effort to reduce the number of binary variables and thereby problem complexity.

6.3.5 Elimination of variables

To be able to solve the planning problem within reasonable time, elimination of variables is essential. By disregarding multiple products, fixing ships to sets of fish farms and only allowing one visit to each fish farm, the number of variables is dramatically reduced. The decision variables x, y, u and w should only take the values 0 or 1 and this results in a problem with many binary variables. Therefore, we have made further effort to reduce the number of binary variables. As mentioned in Chapter 5, u is not included in the implementation. Constraints (5.2), (5.5) and (5.53) ensure that variables y and w are always set to 0 or 1, even when declared as continuous. Therefore, it is enough to declare x as binary, but with five indexes there are still many binary variables.

We have reduced the number of x variables by only creating variables where $i \neq j$. Also, x variables are only created for allowed combinations of locations, visit numbers and ships, with regards to sets of fish farms and allowed number of visits. The restrictions on allowed combinations are also used when creating the other variables, such as the load variables l . It is especially important to reduce the number of load variables in the multi-commodity flow formulation, since these variables have seven indexes. The elimination of variables also allows us to reduce the number of constraints created, by only creating constraints where a related variable exists for the corresponding index combination.

6.4 Attempts of Model Improvement

6.4.1 Special ordered sets

A time window for working hours is given for each day of the planning horizon. The implementation of soft time windows is done using the binary variables, σ_{imd} . For each fish farm visit (i, m) , at most one of the $|\mathcal{D}|$ time window variables σ_{imd} can be chosen. Therefore, we have also tried modeling the variables as part of a special ordered set of type 1 (SOS1). A SOS1 is an ordered set of variables where at most one variable can have a non-zero value. This means that there is one SOS1 for each fish farm visit (i, m) and the set variables are ordered according to days. Branching on SOS1 variables leads to more balanced search trees and is more efficient than branching directly on binary variables, as discussed in Beale and Tomlin (1970).

6.4.2 Tightening constraints

We have attempted to tighten the models by creating time windows for the time variables t_{im} and t_{im}^E .

A_{im}	Earliest start time for visit (i, m)
B_{im}	Latest start time for visit (i, m)
A_{im}^E	Earliest end time for visit (i, m)
B_{im}^E	Latest end time for visit (i, m)

$$A_{im}(1 - y_{im}) \leq t_{im} \leq B_{im} \quad (i, m) \in \mathcal{S} \quad (6.10)$$

$$A_{im}^E(1 - y_{im}) \leq t_{im}^E \leq B_{im}^E \quad (i, m) \in \mathcal{S} \quad (6.11)$$

Constraints (6.10) and (6.11) limit the value of a visit's start and end time. If a visit is not made, the time variables are set equal to the end of the previous visit and the lower bound should not apply.

The earliest start time for each first visit to a location is calculated as the travel time from the nearest possible initial position. The earliest start time for subsequent visits is set to the earliest start time of the previous plus the minimum number of hours between each visit. The latest start time for the first visit to a fish farm is when its stock level reaches zero, while for the factory it is when the maximum capacity is reached. The latest start time for subsequent visits to fish farms is the latest end time of the previous visit plus the time it takes to reach a stock level of zero, given that it was filled to maximum. For the factory, the latest start time for subsequent visits is given by the latest end time of the previous visit plus the time it takes to again reach maximum capacity, given that the stock was cleared.

The earliest end time of a visit to a location is the earliest start time of the visit plus the loading or unloading time of the minimum loading or unloading quantity. The minimum loading or unloading quantity is given by the minimum number of hours between visits to a location. The latest end time is set to the latest start time plus the time it takes to load or unload the maximum quantity, given by the location's storage capacity. If any of the calculations give a value higher than the end time, T^{MAX} , the value is set to T^{MAX} instead.

Creating time windows for each visit, allows us to tighten the linearizations of the timing constraints, (6.5), (6.6) and (6.7). Instead of using T^{MAX} as the Big M, we use the time windows for the time variables.

$$t_{im}^E + (T_{ij}^S + B_{im}^E - A_{jn}) \sum_{v \in \mathcal{V}} x_{imjnv} - t_{jn} \leq B_{im}^E - A_{jn}(1 - y_{j1}) \quad (6.12)$$

$$(i, m) \in \mathcal{S}, (j, n) \in \mathcal{S}$$

In constraints (6.12) we use the difference between the latest end time of visit (i, m) and the earliest start time of visit (j, n) as Big M. The tightening effect from using time windows is strongest for the first visit numbers, since the difference between start and end times is larger for higher visit numbers. The earliest start time of visit (j, n) should only be subtracted from the right hand side if location j is visited.

If the location is not visited, t_{jn} is zero, and we risk infeasibility if t_{im}^E is larger than $B_{im}^E - A_{jn}$.

$$t_{im} - t_{i(m-1)}^E + (B_{im} - A_{i(m-1)}^E)y_{im} \leq B_{im} - A_{i(m-1)}^E \quad (i, m) \in \mathcal{S} | m > 1 \quad (6.13)$$

$$t_{im} - t_{i(m-1)}^E - (B_{im} - A_{i(m-1)}^E)y_{im} \geq -(B_{im} - A_{i(m-1)}^E) \quad (i, m) \in \mathcal{S} | m > 1 \quad (6.14)$$

In constraints (6.13) and (6.14) we use the difference between the latest start time of visit number m and the earliest end time of visit number $(m - 1)$ as the Big M for each location i .

6.4.3 Valid inequalities

In order to tighten the linear relaxation of the problem, we have attempted to create valid inequalities to cut off solutions that are not integer feasible. This could make the problem easier to solve. We have created cuts that are added a priori to the formulations and also tried generating cuts dynamically as they are violated.

Minimum visit number

We have calculated the minimum number of visits required for each location i , given by μ_i . μ_i is calculated by dividing location i 's net demand by its maximum storage capacity. The net demand of the factory is given by the total production during the planning horizon plus the initial stock, less the storage capacity. The net demand of a fish farm is given by its total demand during the planning horizon less its initial stock level.

$$\sum_{(j,n) \in \mathcal{S}_v} \sum_{v \in \mathcal{V}} x_{i\mu_i jnv} = 1 - y_{i1} \quad i \in \mathcal{N}^C \cup \mathcal{N}^P \quad (6.15)$$

The equalities (6.15) ensure that the visit number equal to a location's minimum number of visits is used, given that the first visit to the location is made. The symmetry breaking inequalities (5.6) ensure that all visit numbers up to this are used as well.

Minimum load destined to a fish farm

For the multi-commodity flow formulation, the net demand ND_i can be used as a lower bound on loads destined to a fish farm i .

$$\sum_{(j,n) \in \mathcal{S}} \sum_{m \in \mathcal{M}_i} \sum_{v \in \mathcal{V}} l_{jnimv} \geq ND_i u_i \quad i \in \mathcal{N}^C \quad (6.16)$$

The inequalities (6.16) ensure that the total load destined to a fish farm is at least enough to cover its net demand. If the fish farm is supplied externally, the constraints are not binding.

Subtour elimination constraints

$$\sum_{v \in \mathcal{V}} x_{imjnv} + \sum_{v \in \mathcal{V}} x_{jnimv} \leq 1 \quad (i, m), (j, n) \in \mathcal{S} \quad (6.17)$$

By running the models without binary restrictions on the routing variables x_{imjnv} , the problem solves immediately. We discovered that between pairs of fish farms with short travel times, the routing variables are often set to a fractional value close to one for both arcs connecting two nodes, (i, m, j, n) and (j, n, i, m) . Constraints (6.17) eliminate these subtours.

Dynamic cut generation

We have also tested a set of cuts called clique inequalities, as described in Agra et al. (2014). These inequalities are created by finding conflicting routing variables. In our implementation, this means finding a set of x_{imjnv} variables where at most one of the variables can be used. We have considered the following conflicts:

$$\begin{aligned} x_{imjnv} \text{ and } x_{jnimw} \text{ where } & (i, m), (j, n) \in \mathcal{S}, v, w \in \mathcal{V} \\ x_{imjnv} \text{ and } x_{imkow} \text{ where } & (i, m), (j, n), (k, o) \in \mathcal{S}, v, w \in \mathcal{V} \\ x_{jnimv} \text{ and } x_{koimw} \text{ where } & (i, m), (j, n), (k, o) \in \mathcal{S}, v, w \in \mathcal{V} \\ x_{jnimv} \text{ and } x_{imjow} \text{ where } & (i, m), (j, n), (j, o) \in \mathcal{S} | n > o, v, w \in \mathcal{V} \\ x_{jnimv} \text{ and } x_{imkow} \text{ where } & (i, m), (j, n), (k, o) \in \mathcal{S}, v, w \in \mathcal{V} | v \neq w \end{aligned}$$

The first conflict is equal to the subtour elimination constraints (6.17). The second says that you can not go from a location to more than one other location and the third says that you can not go to a location from more than one other location. The fourth conflict says that you can not come from a location with a visit number n and then leave to the same location with a visit number lower than n . The last conflict says that if you enter a location using ship v , you can not leave using ship w .

Adding all such inequalities a priori would create a very large set of redundant constraints. Instead, we have used the cut manager in Xpress to dynamically

create clique inequalities as they are violated during the branch-and-bound search. A current fractional solution is searched for conflicting variables and a conflict matrix is created. Finding the maximum clique is an NP-hard problem, and instead we use a heuristic approach to find some clique. First, we find the pair of conflicting variables with the largest total fractional value and then other conflicting variables are added in a greedy fashion. If the resulting clique of conflicting variables sums to more than one, a cut is added. Due to this simple heuristic approach, cuts are not necessarily found, even though there could be many.

6.4.4 Aggregated variables for branching

As explained earlier, an effort has been made to reduce the number of variables. This includes only implementing variables that are strictly needed to solve the problem. However, a variable that is aggregated from the x variables could be preferable to branch on. In order to test this, we have added the variables w_{imv} used in the multi-commodity flow formulation to the arc-load and arc-flow formulations as well. We have also experimented with prioritizing which binary variables to branch on by using the function *setmipdir* in Xpress (Dash Optimization, 2007).

6.4.5 Stricter constraints

Stricter constraints could make the model more realistic and easier to solve. Therefore, we have experimented with constraints to remove end-of-horizon (EOH) effects and increased minimum unloading quantities.

EOH

To avoid unwanted EOH effects, constraints should be added to ensure that not all stock levels reach their extreme values at the end of the planning horizon. We have experimented with limits on aggregated stock levels for fish farms and a downscaled capacity at the factory.

$$s_{im}^E + P_i(T^{MAX} - t_{im}^E) \leq 0.8 \cdot S_i^{MAX} \quad (i, m) \in \mathcal{S}^P | m = |\mathcal{M}_i| \quad (6.18)$$

The EOH constraint for the factory is given as (6.18), where the end stock level is limited to 80% of maximum storage capacity.

$$D \sum_{i \in \mathcal{N}^C} S_{ip}^{MIN} u_i \leq \sum_{i \in \mathcal{N}^C} [s_{imp}^E - R_{ip}(T^{MAX} - t_{im}^E) - S_{ip}^0(1 - u_i) + R_{ip}T^{MAX}(1 - u_i)]$$

$$m = |\mathcal{M}_i| \quad (6.19)$$

The EOH constraint (6.19) for fish farms require the total fish farm stock to be at least D times the total safety stock of all fish farms. We have experimented with two and three times the total safety stock level. Externally supplied fish farms should not contribute to the total amount. These fish farms have all stock variables set to initial stock levels, and therefore this term is subtracted, while the term for remaining demand is added to ensure that the right hands side equals zero.

Minimum unloading quantities

In the original model, minimum unloading quantities were set to one feed day for each fish farm. Marine Harvest believes that at least three feed days will be delivered at each visit. Therefore, we have experimented with values of two, three and four feed days.

6.5 Parallel Branch-and-Bound Search

The problem formulated in Chapter 5 is complex and has an enormous solution space. In an effort to decrease the solution time, we have attempted to parallelize the work of searching the solution space by embedding the formulation that performs best, within a parallel branch-and-bound framework. We have explored two levels of branch-and-bound parallelization, resulting in the development of two frameworks. The frameworks are written in C++, while underlying models are still solved using Xpress Optimizer. The C++ code for both frameworks can be found in Appendix F.

As illustrated in Figure 6.1, we have a master process which governs n worker processes. Communication is done using the standard Message Passing Interface (MPI). The master sends subproblems to workers, while workers send solutions to the master. There is no intermediate information sharing on current incumbent solution or lower bounds and workers do not communicate with each other. To account for the increased use of resources, the solution time should decrease accordingly. As explained in Chapter 4, this is not necessarily the case, since both detrimental and acceleration anomalies can occur.

The first framework utilize the fact that each branch-and-bound node can be solved separately, and the branch-and-bound search is governed by the framework. The second framework creates n subtrees and lets the workers completely solve these. Here, the branch-and-bound search is only initiated by the framework and Xpress Optimizer is used to search the subtrees created.

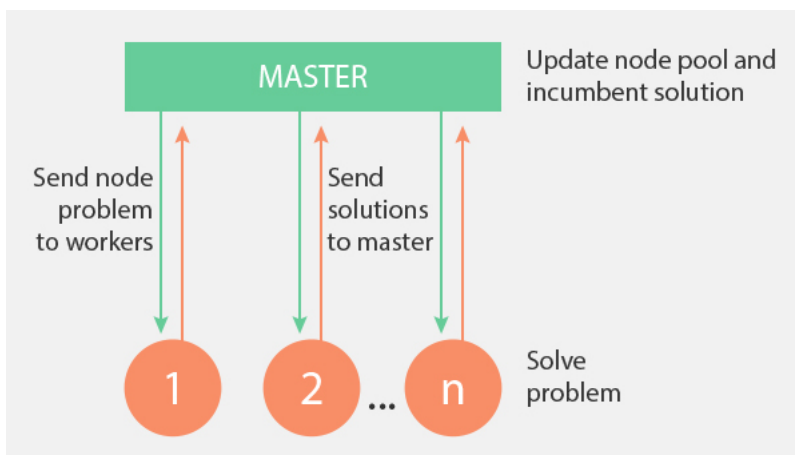


Figure 6.1: Illustration of a parallel branch-and-bound framework.

6.5.1 Parallel work on nodes

The first framework parallelizes the work of solving branch-and-bound nodes. Figure 6.2 shows the implemented classes with the most important fields and functions. The search process is governed by *bbclass* and is run from the master. Each node, represented by *nodeclass*, contains a list of which binary variables are relaxed or fixed and a lower bound derived from its parent's solution value.

The master starts by creating a node where all binary restrictions on variables are relaxed. This node is sent to a worker process, governed by *workerclass*, which solves the relaxed problem and returns a solution. If the solution is infeasible, the problem is infeasible and the algorithm stops. If the solution is integer feasible, the optimal solution is found. If the solution is feasible, but not integer feasible, the problem is branched into two subproblems. A variable is fixed to either 0, creating the down-child, or 1, creating the up-child. The parent node is removed from the nodepool, and the two children nodes are added. Subsequent solutions that are infeasible or integer feasible result in nodes being pruned, meaning they are simply removed from the nodepool. If a solution is integer feasible and better than an incumbent solution, the master saves this node as the incumbent node.

When all workers have returned their solutions, the master distributes the next problems to be solved, meaning that the algorithm is synchronous. Solving each node problem takes little time, since all binary variables are either relaxed or fixed. Therefore, waiting for other workers should not lead to high idle times. However, the master can become a bottleneck since it has to collect all solutions from workers, branch or prune nodes, and distribute new problems.

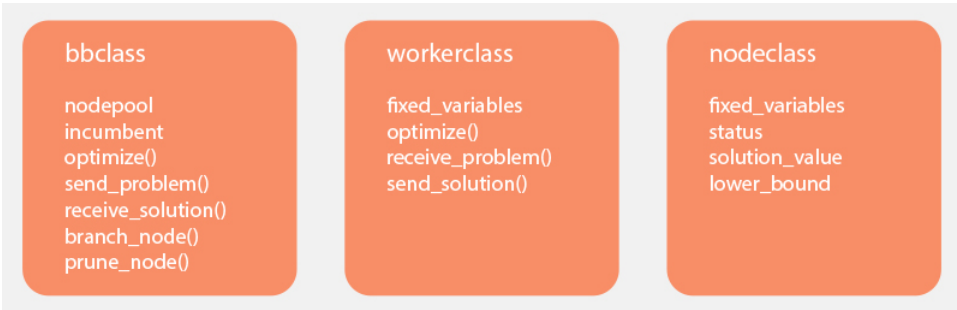


Figure 6.2: Classes implemented in the node-based parallel branch-and-bound framework.

The decisions on which variable to branch on and which node to solve next are made rather simply. If a problem is feasible, but not integer feasible, the worker returns the variable with a value closest to 0.5 and the master branches on this. The node list is sorted in a depth first search order, and the deepest node is always sent first. We have tested sorting the down-child before the up-child and vice versa. There are no heuristics or cuts added during the search or sophisticated branch-and-bound rules. Due to this and the fact that our implementation is synchronous, we do not expect to achieve linear speedup.

The node-based framework has been tested on computers from Solstorm’s rack 3, since these computers are faster than those in rack 5. The computers in rack 3 have 24 gigabytes of memory each, while the computers in rack 5 have 128 gigabytes. Since the first framework is based on having several workers solve LP problems, little memory is needed for each worker. The master process could risk running out of memory, since it needs to keep track of all open nodes, but we view the increased speed as more important and chose to use rack 3.

6.5.2 Parallel work on subtrees

In our second framework, the master divides the original problem into n subproblems by fixing $\log_2 n$ binary variables. After this, the branch-and-bound search is governed by Xpress, since each worker solves its problem as a mixed integer program with the given fixed binary variables. When a worker is done optimizing or the maximum time limit is reached, it reports back to the master whether a problem was infeasible or feasible, and the value of the best integer solution, if any. When receiving a solution, the master saves the best solution found so far as the incumbent solution. Figure 6.2 shows the two classes implemented, which are similar to the classes *bbclass* and *workerclass* for the first framework. No representation of nodes is needed here, since the master only distributes subtrees once.

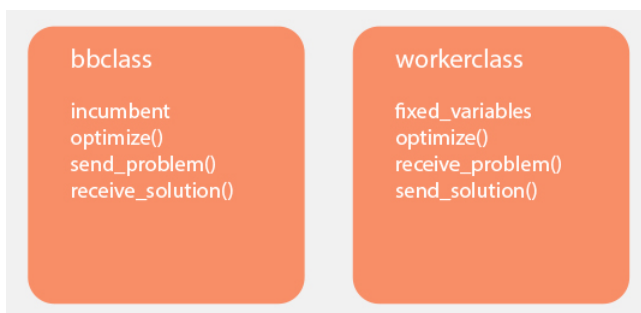


Figure 6.3: Classes implemented in the tree-based parallel branch-and-bound framework

Solving each mixed integer program takes a long time. To achieve an effective use of resources, the algorithm should be asynchronous. This means that workers receive new work once they are free and the master reports on the current upper bound to workers in order for them to terminate if working on a subtree with a worse lower bound. However, developing an asynchronous algorithm is complex and we opted for a simple synchronous algorithm where each worker solves their designated integer programs, and the master waits for all workers to finish. The choice of which variables to fix are made rather simple, but we have chosen to fix variables that indicate whether an arc between two closely located fish farms should be used or not, since we believe that these are likely to be used in a good solution. In order to avoid workers becoming idle due to infeasible subtrees, we have ensured that the binary variables to fix do not yield infeasible subproblems for any combination of fixed variables.

The tree-based framework was tested on rack 5, since every worker needs a large amount of memory to solve their assigned subtrees. A computing node in rack 5 has four processors, each consisting of 16 cores, meaning that there can be up to 64 workers on one computer. With multiple workers sharing one computer, the total number of workers can be increased and hence more binary variables can be fixed. However, Xpress Optimizer is able to utilize up to 50 parallel threads during the branch-and-bound search and when many workers share one computer, there will be less threads available for internal parallelization. This could lead to Xpress working slower. The workers also risk running out of memory, as this will be shared among them. With only one worker per computer, the worker can fully utilize the internal parallelization of Xpress and does not need to share the 128 gigabytes of memory with other workers.

We have tested the framework using several configurations, both with one and multiple workers per computer. Due to strained availability of resources, we could only use 16 computers from rack 5 for parallel testing. Therefore, the option with one worker per computer was limited to a total of 16 workers. With several workers sharing one computer, we have used 32 workers, where two workers share

one computer, 64 workers, where four workers share one computer and 128 workers, where eight workers share one computer. With 16, 32, 64 and 128 workers, we can fix four, five, six and seven binary variables, respectively.

Chapter 7

Computational study

This chapter gives an overview of the results from testing our three formulations using three various-sized test cases. For each combination of formulation and test case, tightened constraints and valid inequalities have been tested, both separately and in combination. These are hereby referred to as test instances. Firstly, we present the different test instances and the notation of these in Section 7.1. In Section 7.2 we present results from the smallest test case. Here, we have tested the effect of using special ordered sets of type 1 (SOS1) for the time window variables σ_{imd} , as well as the effect of different cuts. Sections 7.3 and 7.4 present the results from model runs for the medium and large test case, respectively. For the large test case, we have also tested the effects of further improvements and problem reduction. Section 7.5 gives a comparison of the three formulations. In Section 7.6 we present the results from testing our two parallel frameworks on the instance that turned out to perform best. Lastly, Section 7.7 explores the best solution found with regards to costs, underage and utilization of production and transportation capacity.

7.1 Test Instances

Each test instance is named according to which test case it belongs to, which formulation is used and which cuts have been added. The abbreviations used are listed in Table 7.1. For example, 20_AL_T denotes the small test case, using the arc-load formulation with limits on time variables. The TM instance is an extension of the T instance, where the limits on time variables are used as Big M in linearized time constraints. MV, S and ML denotes the valid inequalities described in Chapter 6, related to minimum number of visits, subtour elimination and minimum load to a destination. B is the basic formulation without tightening of constraints or valid inequalities added.

Test cases:	Notation
20 fish farms	20
40 fish farms	40
60 fish farms	60
Formulations:	
Arc-load	AL
Arc-flow	AF
Multi-commodity flow	MCF
Cuts:	
Basic - no cuts	B
Limits on time variables	T
Tightened linearizations of time constraints	TM
Minimum number of visits	MV
Subtour elimination constraints	S
Minimum loads to a destination	ML

Table 7.1: Notation for test instances.

In the result tables, we will list the best integer solution found, the lower bound and the gap between these. Values are given in 1000 NOK. Note that we have used our own cost estimates and hence the values should not be interpreted as actual costs. The best solution and lower bound among the different formulations and instances for a test case are boldfaced. If a subsequent model run finds a better solution or bound for the test case, this value will also be boldfaced. A dash, $-$, means that no integer solution was found within the set running time. An asterisk, $*$, means that the instance ran out of memory before the set running time had passed. The ML instance only applies to the multi-commodity flow model, and results are not applicable, N/A, for the other two models.

We started our testing with the small-sized test case, then the medium-sized and lastly, the real-sized test case. Along the way, we have made certain conclusions and not all configurations are tested for all test cases. If a cut did not seem to give any improvement or a formulation proved inefficient, it was excluded from subsequent tests.

7.2 Small-Sized Test Case

In the small-sized test case, there are 20 fish farms to supply. We use a partial split of fish farms, as explained in Chapter 6, and allow two visits to each fish farm. The problem size of the basic instance for each formulation is shown in Table 7.2. We see that the number of binary variables is the same for all formulations, while the multi-commodity flow formulation has a much larger number of variables and constraints compared to the other two. This is because of the many load variables, $l_{imjnkov}$. The arc-flow formulation has more variables than the arc-load formulation, but fewer constraints, since the nonlinear loading constraints (5.9) are avoided.

		20_AL_B	20_AF_B	20_MCF_B
Variables:	Before presolve	2,493	4,130	46,020
	After presolve	2,398	3,973	44,127
Binary variables:	Before presolve	2,099	2,099	2,099
	After presolve	2,034	2,033	2,032
Constraints:	Before presolve	6,358	4,889	90,408
	After presolve	6,017	4,460	83,264

Table 7.2: Problem size of the basic instances for the small test case.

Xpress Optimizer has an integrated presolve function which manages to reduce the number of variables and constraints for all three formulations. The instances with cuts added have more constraints than the basic instance before presolve. After presolve is applied, the problem sizes are similar to the basic instance. This could be because some of the added constraints are redundant and removed, but it could also be that the added cuts help remove other constraints and variables.

7.2.1 LP bound

The solution of the linear programming (LP) relaxation for each combination of formulation and instance is given in Table 7.3. As expected from theory, the LP optimum of the arc-flow formulation is higher than for the arc-load formulation. The multi-commodity flow formulation has a slightly higher LP optimum than the arc-flow formulation for all but the MV instance. If constraints (5.61) and (5.62) had not been aggregated over products, the multi-commodity flow formulation probably would have been even tighter. We remark that the arc-load and arc-flow models find the LP optimum in an instant, while the multi-commodity flow model takes around 20 minutes to find the LP solution. A tighter formulation results in a smaller branch-and-bound tree, but this often comes at the cost of an increased number of variables and constraints. Then, the solution time of each branch-and-bound node increases, as observed for the multi-commodity flow formulation.

	B	T	TM	MV	S	ML	MV_S
20_AL	94.50	94.50	94.50	101.04	98.60	N/A	105.5
20_AF	99.11	99.11	99.11	105.38	100.74	N/A	107.01
20_MCF	99.19	99.19	99.19	105.38	100.82	99.19	107.01

Table 7.3: LP optima for the small test case, all formulations and instances.

For all formulations, the LP optimum increases when adding the constraints for minimum number of visits and subtour elimination, where the first has the largest impact. The subtour elimination constraints have a larger impact on the arc-load formulation than the two other formulations, since it has more subtours in its LP solution. When combining the instances MV and S, the LP bound increases further

for all three formulations. The tightenings related to time variables do not cut off the LP optimum of the basic instance for any of the formulations and neither do the minimum load constraints for the multi-commodity flow formulation.

7.2.2 SOS1

We have tested the basic instance for all three formulations with and without SOS1 for the σ_{imd} variables. The maximum running time was set to 10 hours. As can be seen from Table 7.4, the instances with SOS1 achieve smaller gaps for all three formulations. The best solution and bound for each formulation are found with SOS1, but the arc-flow formulation finds the same solution without SOS1.

		20_AL_B	20_AF_B	20_MCF_B
Without SOS1	Best solution	143.49	137.27	149.00
	Lower bound	123.71	130.03	114.78
	Gap	13.8%	5.3%	23.0%
With SOS1	Best solution	137.72	137.27	138.39
	Lower bound	126.64	131.25	108.27
	Gap	8.0%	4.4%	21.8%

Table 7.4: Results for the small test case, with and without SOS1. Running time: 10 hours.

The branch-and-bound tree will be different for implementations using SOS1 compared to only using binary branching. If the set is ordered in a good way, SOS1 could have a positive effect on the branch-and-bound search. However, Xpress Optimizer use efficient strategies for binary branching as well. Even with a natural order on the SOS1 variables, the effect could sometimes be positive, sometimes negative. Since the effect appears to be positive for all formulations, we have decided to use SOS1 in subsequent tests, although we have experienced that the extra capacity obtained when visiting within service hours is rarely utilized. Hence, all the σ_{imd} variables in a set are often set to zero, and if none of the variables in a set is used, the set will not be branched on.

We are not sure if the positive effect results from random differences from a few diverging branching decisions or from improved branching due to using SOS1. We believe that with more correct data, the service hour aspect would be more utilized and using SOS1 could prove to have a more pronounced positive effect. Even though the σ_{imd} variables are rarely used, we wish to keep the aspect of service hours in the model. Marine Harvest has emphasized that being able to load up to 100% of capacity is important.

7.2.3 Tightened constraints and valid inequalities

Xpress Optimizer’s presolve function generates sophisticated cuts and improves the bounds of the problem before the actual branch-and-bound search starts. In general, disabling presolve should lead to a larger search tree and worse upper and lower bounds. When testing the different cuts, we have run the models with presolve disabled, as well as enabled, in order to test the effect of cuts without having Xpress Optimizer generate improvements of its own. All instances have been run for one hour, due to the large number of instances to test, even though it is more common with longer running times. The results are given in Table 7.5. As can be seen, some instances of the multi-commodity flow model did not find any integer solutions.

	Presolve		B	T	TM	MV	S	ML
20_AL	Off	Best solution	148.00	147.72	138.71	138.39	149.74	N/A
		Lower bound	115.70	114.06	116.73	120.05	114.50	
		Gap	21.8%	22.8%	15.9%	13.3%	23.5%	
	On	Best solution	137.72	138.72	147.00	142.72	139.57	N/A
		Lower bound	123.00	120.00	119.00	121.04	122.43	
		Gap	10.69%	13.5%	19.1%	15.2%	12.3%	
20_AF	Off	Best solution	141.74	141.55	138.72	137.72	141.49	N/A
		Lower bound	120.03	118.23	122.35	125.51	117.37	
		Gap	15.5%	16.5%	11.8%	8.9%	17.1%	
	On	Best solution	137.72	140.72	138.72	137.72	137.72	N/A
		Lower bound	123.29	122.54	121.50	122.35	127.18	
		Gap	10.5%	12.9%	12.4%	11.2%	7.65%	
20_MCF	Off	Best solution	-	179.74	47,674	157.77	173.45	183.14
		Lower bound	103.82	104.12	101.90	106.80	103.26	104.26
		Gap	-	42.1%	99.8%	32.7%	40.5%	43.1%
	On	Best solution	-	182.75	149.00	-	151.90	-
		Lower bound	108.27	109.54	109.64	108.85	109.31	107.85
		Gap	-	40.1%	26.4%	-	28.1%	-

Table 7.5: Results for the small test case, with presolve enabled and disabled.
Running time: One hour.

The best solution among the three formulations is found using the arc-load and arc-flow formulations, both with presolve enabled. The best bound is found by the arc-flow instance S, which also finds the best solution. The best solution and lower bound of 137.72 and 127.18 give a gap of 7.65%. In Table 7.5, these values are not boldfaced as they are not better than the solution and bound found when testing without and with SOS1 for 10 hours, see Table 7.4.

For the arc-load formulation with presolve disabled, three of the instances with cuts are better than the basic instance. However, the basic instance with presolve enabled is better than all instances for this formulation, indicating that Xpress Optimizer is able to add these cuts more efficiently by itself. This is also the case for the arc-flow formulation, where the basic instance with presolve enabled is better than instances T, TM and S with presolve disabled.

For the arc-flow formulation, the subtour elimination constraints only have a positive effect on the lower bound of the basic instance with presolve enabled, possibly because presolve is able to utilize the extra constraints better. The MV instance with presolve disabled achieves a better lower bound than all but the S instance with presolve enabled. This is not as expected and it is difficult to explain, as we do not have sufficient knowledge about how presolve is performed by Xpress.

The multi-commodity flow formulation with presolve disabled finds integer solutions for all but the basic instance. With presolve enabled no solution is found for the MV and ML instances and they seem to complicate the formulation more than improving it. This is not as expected, since presolve in general should improve the branch-and-bound search, but we believe it is due to random differences in branching decisions. All bounds for this formulation are better with presolve enabled.

Of the three formulations, it is the arc-flow model that performs best and it has the smallest gaps for all instances. None of the cuts appear to perform significantly better or worse than others. Since using presolve in general should give better solutions and bounds faster, subsequent tests are performed with presolve enabled. We have increased the running time to 10 hours., which will also be used for subsequent tests.

The results from running all instances for 10 hours can be found in Table 7.6. As can be seen, the arc-load instance T ran out of memory before 10 hours had passed, due to a very large number of active branch-and-bound nodes.

		B	T	TM	MV	S	ML
20_AL	Best solution	137.72	138.72*	140.57	140.27	138.85	
	Lower bound	126.64	123.03*	123.63	124.16	126.12	N/A
	Gap	8.0%	11.3%	12.1%	11.5%	9.2%	
20_AF	Best solution	137.27	138.72	137.27	137.27	137.27	
	Lower bound	131.25	129.81	131.34	131.43	137.26	N/A
	Gap	4.4%	6.4%	4.3%	4.3%	0.0%	
20_MCF	Best solution	138.39	141.17	137.72	146.37	141.36	144.26
	Lower bound	108.27	110.20	114.89	109.53	117.65	112.96
	Gap	21.8%	21.9%	16.7%	25.2%	16.8%	21.7%

Table 7.6: Results for the small test case. Running time: 10 hours.

The best solution and bound are found using the arc-flow formulation with subtour elimination constraints added. The gap is negligible and we consider 137.27 to be the optimal solution for the small-sized test case. Optimality was proved after around seven hours, but the optimal solution was found after less than two hours by instance S. The B, TM and MV instances also found this solution, but after much longer time and with weaker bounds. The arc-load and multi-commodity formulations finds a near-optimal solution of 137.72 using instances B and TM, respectively. Based on the results in Table 7.6, we will for the remainder of this section, analyze the different cuts.

Time windows (T, TM)

For the arc-load formulation, neither the solution nor the lower bound improves when using the T or TM instance. If only the time windows are created, the arc-flow formulation is not improved, while the best solution is found if the time windows are also used as Big M in linearizations. For the multi-commodity formulation, the TM instance finds a near-optimal solution and improved bound, while the T instance only slightly improves the bound and finds a worse solution.

Since the T instance appears to have little positive effect, except for a slight improvement of the lower bound for the multi-commodity flow formulation, we have decided to exclude this instance. The TM instance finds the optimal and a near-optimal solution for the arc-flow and multi-commodity flow formulations, respectively and improves their bounds. Therefore, we have chosen to continue testing the TM instance.

Minimum number of visits (MV)

The constraints setting the minimum number of visits to locations do not improve the solution or lower bound compared to the basic instance for the arc-load formulation. For the arc-flow formulation it finds the optimal solution and approximately the same bound as the basic and TM instances. The multi-commodity flow formulation does not find a better solution when these constraints are added, but the lower bound increases slightly.

These valid inequalities appear to complicate the arc-load and multi-commodity flow models more than improving them and they appear to have little effect on the arc-flow model. Still, the instance finds the optimal solution and it is the instance that increases the LP bound the most for all formulations. Therefore, we will continue testing the MV instance.

Subtour elimination constraints (S)

The arc-load formulation is not improved compared to the basic instance when adding subtour elimination constraints. However, the arc-flow formulation finds the optimal solution and proves optimality. The multi-commodity flow model achieves a significantly stronger bound using this instance, compared with the basic instance. Since the subtour elimination constraints appear to have a positive effect on lower bounds and the arc-flow formulation proves optimality with this instance, we will continue testing these cuts.

Minimum load constraints (ML)

The valid inequalities requiring loads to a destination to be at least as large as the destination’s net demand improves the bound of the basic multi-commodity flow instance, but the solution is not better. Since the lower bound is improved, we will continue testing this instance.

Combinations of cuts

We have tested the combination of all remaining instances, as well as this combination with each cut excluded. Instance T is not included in the combinations. The results can be found in Table 7.7. The first combination, TM_MV_S means all cuts for the arc-load and arc-flow formulations, while the instance TM_MV_S_ML is the combination of all cuts for the multi-commodity flow formulation. Again, the arc-load formulation ran out of memory for one instance.

		TM_MV_S	TM_MV(_ML)	TM_S(_ML)	MV_S(_ML)	TM_MV_S_ML
20_AL	Best solution	137.72	139.35*	140.74	137.27	
	Lower bound	124.00	125.00*	128.06	129.55	N/A
	Gap	10.0%	10.3%	9.0%	5.6%	
20_AF	Best solution	137.27	137.27	137.72	137.27	
	Lower bound	137.26	131.52	129.80	137.26	N/A
	Gap	0.0%	4.2%	5.8%	0.0%	
20_MCF	Best solution	140.72	137.72	138.39	137.72	138.72
	Lower bound	115.28	109.80	119.54	116.71	115.93
	Gap	18.1%	20.3%	13.7%	15.3%	16.5%

Table 7.7: Results for the small test case, combinations of effective cuts. Running time: 10 hours.

When combining cuts MV and S, the arc-load formulation finds the optimal solution. The bound is improved as well, giving a gap of 5.6%. For the arc-flow formulation, optimality is proved for the combination with all cuts and MV_S. When combining all cuts, optimality is proved after less than five hours, faster than both the S and MV_S instances. The multi-commodity flow formulation does not find the optimal solution, but its previously best bound is improved with the combination TM_S_ML.

The multi-commodity flow model spends much time solving each branch-and-bound node and therefore searches through fewer solutions during 10 hours than the other two formulations. Using the MV_S_ML instance, the multi-commodity flow formulation finds the solution 137.72, after around three hours and 20,292 processed nodes. The MV_S instance for the arc-load formulation finds this solution after 20 minutes, but the number of processed nodes is 798,786. The same instance with the arc-flow formulation finds this solution after around two hours and 2,493,234 nodes.

The fact that the multi-commodity flow formulation finds good solutions with a much smaller branch-and-bound tree, indicates that this is a tight formulation. Since the formulation is tighter than the other two formulations, its solution space should also be smaller. However, since the formulation has larger gaps than the other two formulations after 10 hours, it does not seem like the increased solution time of each branch-and-bound node is balanced out by the smaller solution space. The formulation also suffers from scaling issues and we conclude that the multi-commodity flow model is not viable for larger test cases. The problem will be too large and complex and the solving of each branch-and-bound node will be even slower. We will abandon this formulation, and continue with the arc-load and arc-flow formulations. Therefore, the ML instance will also be abandoned.

7.3 Medium-Sized Test Case

The medium-sized test case has 40 fish farms to supply. We still use a partial split, but none of the formulations found any integer solutions after 10 hours if fish farms are allowed to have two visits. Therefore, we have reduced the problem size by only allowing one visit to each fish farm. This is a reasonable simplification, since all but one fish farm have only one visit in the optimal solution found for the small test case. The problem size of the basic instance for each formulation is shown in Table 7.8. We see the same differences as for the small-sized test case. Due to the reduced number of allowed visits, the problem size of this test case is not much larger than the problem size of the small-sized test case.

		40_AL_B	40_AF_B
Number of variables:	Before presolve	2,579	4,302
	After presolve	2,479	4,153
Number of binary variables:	Before presolve	2,185	2,185
	After presolve	2,079	2,079
Number of constraints:	Before presolve	6,538	5,183
	After presolve	6,195	4,525

Table 7.8: Problem size of the basic instances for the medium test case.

7.3.1 LP bound

The solution of the LP relaxation for each formulation is given in Table 7.9. As for the small-sized test case, only the instances MV and S tighten the LP bound, but here it is the S instance that has the largest effect. We tested whether this change was due to the reduction in the allowed number of fish farm visits, but when two visits are allowed, the S instance still has the strongest LP bound. The fish farms are spread over the same area in all test cases. With a larger test case, there are more fish farms with short travel distances between them. After examining the

LP solution, we discovered that this increases the number of subtours in the LP solution, causing the subtour elimination constraints to have more effect.

	B	TM	MV	S	MV_S
40_AL	102.81	102.81	107.31	125.50	130.00
40_AF	113.72	113.72	118.22	128.82	133.47

Table 7.9: LP optima for the medium test case.

7.3.2 Tightened constraints and valid inequalities

Results from running the remaining formulations on the medium-sized test case for 10 hours can be found in Table 7.10. As can be seen, three of the arc-load instances ran out of memory before 10 hours had passed.

		B	TM	MV	S
40_AL	Best solution	235.04*	207.81*	253.42*	215.39
	Lower bound	138.00*	138.76*	136.72*	138.77
	Gap	41.3%	33.2%	46.1%	35.6%
40_AF	Best solution	197.58	199.11	193.46	192.56
	Lower bound	152.85	154.24	152.79	154.76
	Gap	22.6%	22.5%	21.0%	19.6%

Table 7.10: Results for the medium test case. Running time: 10 hours.

For the arc-load formulation, the best solution is found using the TM instance, while the S instance is the second best. These instances also have the best bounds for this formulation. The arc-flow formulation has better solutions and bounds than the arc-load formulation for all instances. With instance S, the arc-flow model finds the best solution, 192.56, and the strongest lower bound, 154.76, giving a gap of 19.6%.

Combinations of cuts

For the small-sized test case, combinations of cuts turned out to improve the arc-load model. Therefore, we have tested combinations of cuts for both formulations, even though the arc-flow formulation again performs best. The S instance improves the basic instance of both formulations, while the TM and MV instances behave different for the two formulations. Therefore, we will not exclude any cuts before testing combinations. As for the small-sized test case, we have tested the combination of all cuts, as well as the exclusion of each cut. The results can be found in Table 7.11. Again, several of the arc-load combinations ran out of memory.

		TM_MV_S	TM_MV	TM_S	MV_S
40_AL	Best solution	213.29*	214.89*	212.02	248.10*
	Lower bound	139.30*	136.51*	143.05	136.65*
	Gap	34.7%	36.5%	32.5%	44.9%
40_AF	Best solution	225.71	192.56	199.11	193.21
	Lower bound	155.89	156.37	152.45	155.04
	Gap	30.9%	18.8%	23.4%	19.8%

Table 7.11: Results for the medium test case, combinations of effective cuts.
Running time: 10 hours.

The arc-load formulation improves its best bound most when combining instances TM and S, but the best solution is found using the TM instance alone. Compared to combining all cuts, removing the instance MV improves both the bound and solution, indicating that the constraints on minimum number of visits have a negative effect on this formulation. The best solution and bound of 207.81 and 143.05 give a gap of 31.2%, meaning the arc-load formulation is still weaker than the arc-flow formulation.

The arc-flow formulation again finds the solution 192.56, using the combination TM_MV. Strangely enough, this solution was found by the S instance in the previous run, while here it was found using the combination without subtour elimination constraints. The combination TM_MV also finds a better lower bound of 156.37, which gives a gap of 18.8%. Again, it is the arc-flow formulation that performs best, with smallest gaps for all instances. Therefore, we will continue using only the arc-flow formulation for the largest test case.

7.4 Large-Sized Test Case

In the large-sized test case, there are 60 fish farms to supply. As for the medium-sized test case, we use a partial split and only allow one visit to each fish farm. The problem size of the basic instance for the arc-flow formulation is shown in Table 7.12.

		60_AF_B
Number of variables:	Before presolve	8,464
	After presolve	8,117
Number of binary variables:	Before presolve	4,293
	After presolve	4,214
Number of constraints:	Before presolve	9,655
	After presolve	8,541

Table 7.12: Problem size of the basic instance for the large-sized test case.

7.4.1 LP bound

The solution of the LP relaxation for each formulation is given in Table 7.13. As for the medium-size test case, only the instances MV and S tighten the LP bound and the S instance has the largest effect. If combining these, the LP bound is further increased for all three formulations.

	B	TM	MV	S	MV_S
60_AF	116.08	116.08	120.58	137.85	142.35

Table 7.13: LP optima for the large test case.

7.4.2 Tightened constraints and valid inequalities

Results from running instances of the large test case for 10 hours can be found in Table 7.14. No integer solutions was found using the MV instance, and the basic instance has a gap of 63.2%. The best solution, 219.72, is found by the TM instance, while S has the strongest lower bound, 166.19. This gives a gap of 24.4%. The solution 219.72 is found after around four hours.

		B	TM	MV	S
60_AF	Best solution	447.92	219.72	-	242.87
	Lower bound	164.97	165.47	163.14	166.19
	Gap	63.2%	24.7%	-	31.6%

Table 7.14: Results for the large test case. Running time: 10 hours.

Combinations of cuts

We have tested the combinations of all cuts and the results can be found in Table 7.15. A stronger lower bound was found when combining all cuts, but none of the combinations found a better solution than the TM instance alone. Therefore, we have chosen the instance TM for further testing.

		TM_MV_S	TM_MV	TM_S	MV_S
60_AF	Best solution	225.18	-	236.61	235.07
	Lower bound	166.58	161.23	161.20	164.07
	Gap	26.0%	-	31.9%	30.2%

Table 7.15: Results for the large test case, combinations of cuts. Running time: 10 hours.

7.4.3 Effects of further improvements

In order to achieve better solutions and bounds, we have attempted to both reduce the problem size and further improve the arc-flow formulation. We have tested using a full split, as explained in Chapter 6, and tried excluding the aspect of service hours in an attempt to reduce the problem size. We have also tried to improve the formulation by dynamically generating clique inequalities and by adding an aggregated variable to branch on. Lastly, we have increased the minimum unloading quantities and added end-of-horizon (EOH) constraints, in an attempt to make the model more realistic and possibly easier to solve. All subsequent tests are done using the arc-flow instance TM.

Problem reduction

We have reduced the problem size by using a full split and by excluding the aspect of service hours, separately and in combination. We had hoped that this would make the problem easier to solve, since the number of binary variables is reduced. The results can be found in Table 7.16. Using a full split did not find better solutions or bounds. The reduced problem size does not seem to outweigh the negative effect from the reduced flexibility of a full split. Similar results are found for the exclusion of service hours, since only 90% of storage capacity can be utilized at all fish farms. Therefore, we continue using a partial split and the aspect of service hours included. This is also what we consider to be the most realistic formulation of Marine Harvest’s planning problem.

		Full split	Without service hours
60_AF_TM	Best solution	498.61	258.92
	Lower bound	164.38	162.21
	Gap	67.0%	37.4%

Table 7.16: Results of problem reduction for the large test case. Running time: 10 hours.

Dynamic clique inequalities

Motivated by research done in Agra et al. (2014), we implemented a dynamic generation of clique inequalities, as explained in Chapter 6. This turned out to significantly slow down the solution of every branch-and-bound node, since creating the conflict matrix and finding a clique are time consuming activities. In addition to this, the heuristic approach to create clique inequalities proved to be inefficient. Very few usable clique inequalities were generated, because the sum of the fractional values for clique members were not larger than 1, making it meaningless to add the cut. We do not provide any results from these runs, as no integer solutions or tighter bounds were achieved.

Aggregated variables for branching

The w_{imv} variables in the multi-commodity flow formulation are aggregated over the $x_{im,jnv}$ variables. Branching on aggregated variables could lead to more efficient branching and we have added these variables to the arc-flow formulation as well. However, neither the solution nor bound was improved, as can be seen in Table 7.17. This could be because of the increased number of binary variables. We also tried prioritizing branching on these variables by using the Xpress function *setmipdir*, with the same results. Therefore, we abandoned this approach.

Aggregated variables w_{imv}		
60_AF_TM	Best solution	235.90
	Lower bound	160.23
	Gap	32.1%

Table 7.17: Results for the large test case with aggregated variables for branching. Running time: 10 hours.

Stricter constraints

Our last attempt to improve the results was to make the constraints related to minimum unloading quantities stricter and add EOH constraints. We have tested minimum unloading quantities of two, three and four feed days. The tests were run for 10 hours and the results can be found in Table 7.18. No better solutions were found, and we decided not to change these constraints.

Minimum unloading quantities		2	3	4
60_AF_TM	Best solution	238.65	249.23	-
	Lower bound	165.99	164.28	162.47
	Gap	30.5%	34.1%	-

Table 7.18: Results for the large test case with larger minimum unloading quantities. Running time: 10 hours.

As explained in Chapter 5, EOH effects should be taken into account. We have tested the aggregated EOH constraint for fish farms, where the total stock in the end must be more than two times and three times the total safety stock level of all fish farms, denoted by AGG_2 and AGG_3. We have also tested the constraint requiring the stock level at the factory to be below 80% of capacity in the end, denoted by 80%. The tests were run for 10 hours and the results can be found in Table 7.19.

No integer solutions were found using the instances 80% and AGG_2, while AGG_3 finds a solution with a gap of 67%. We have chosen to not add any

EOH		80%	AGG_2	AGG_3
60_AF_TM	Best solution	-	-	480.94
	Lower bound	161.67	163.1	157.91
	Gap	-	-	67.2%

Table 7.19: Results for the large test case with EOH constraints. Running time: 10 hours.

EOH constraints, meaning that stock levels will be at their extreme values after 10 days. Therefore, the model should only be used to plan for the first seven days or less and as new information is revealed, the model should be rerun.

None of the further improvements managed to find a better solution than the TM instance. Figure 7.1 illustrates how the solution and lower bound changes with time for this instance. We see that after around four hours, no better solution is found and the bound is not improved much.

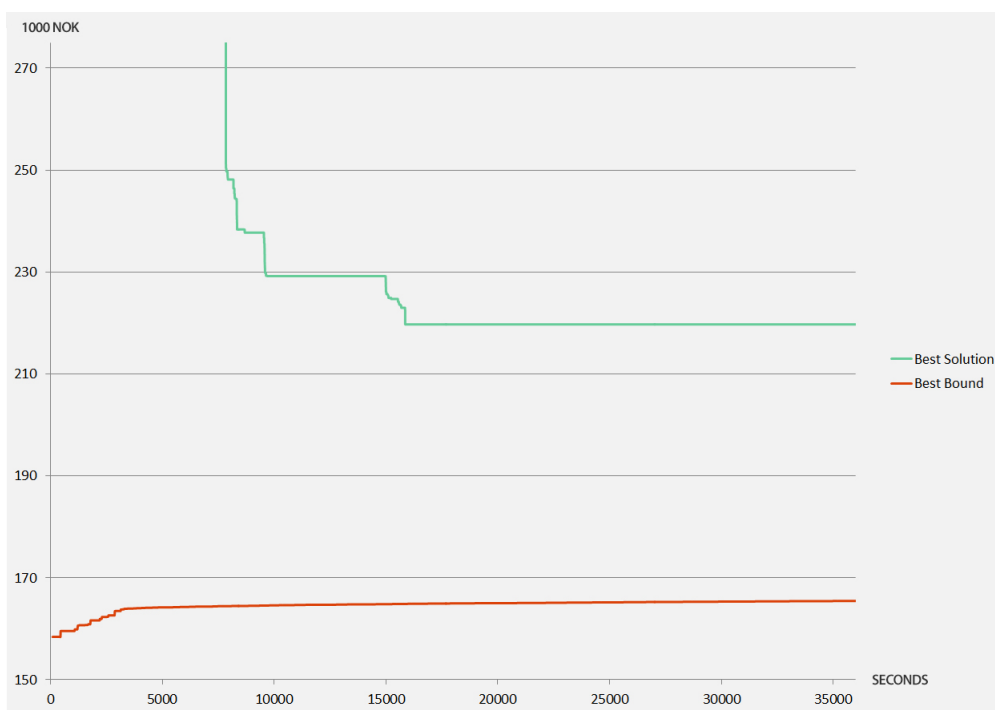


Figure 7.1: Illustration of how the solution and lower bound improves with time.

7.5 Comparison of Formulations

The arc-load formulation has the weakest LP bound, but performs well for the smallest test case. For the medium-sized test case, it is weaker than the arc-flow formulation both in terms of bounds and solutions. The arc-flow formulation has the strongest lower bounds and is the only formulation that proves optimality for the smallest test case. The multi-commodity formulation has the strongest LP bound, but the large number of variables and constraints makes the solving of each branch-and-bound node slow. Therefore, its search tree after 10 hours is significantly smaller than for the other two models. Since it is the tightest formulation, it still finds good solutions after searching through a much smaller number of nodes.

For the small-sized test case, the arc-load formulation have branch-and-bound trees from 75 to 110 gigabytes for the different instances after 10 hours. The trees for the arc-flow instances range from two gigabytes for the instance reaching optimality to 75 gigabytes. The largest branch-and-bound tree of the multi-commodity flow formulation is around five gigabytes. Therefore, the multi-commodity flow formulation does not need the large memory available on rack 5 computers. The model could have been run on the faster computers in rack 3, but we do not believe that the increased speed would be enough to make the formulation viable for larger test cases.

The arc-load model runs out of memory for several instances of the medium-sized test case, even though each computer has 128 gigabytes of memory. This is partly because it manages to search through more nodes than the arc-flow formulation within the running time. Even though both formulations solve the LP relaxation in what appears to be an instant, the arc-load model is faster, due to the smaller number of variables. For some instances the arc-load formulation searches through more than twice as many nodes as the arc-flow formulation. However, it also has more active nodes relative to the number of processed nodes, meaning it is able to prune less. Also, the arc-flow model appears to branch more efficient, since it finds better solutions and bounds, with a smaller branch-and-bound tree. Overall, we can conclude that the arc-flow formulation performs best.

7.6 Parallelization

We have tested our two parallelization frameworks on the large-sized test case, in an attempt to achieve better solutions to the real-sized problem. For this test case, only the arc-flow formulation was used and it was the instance TM that found the best solution.

7.6.1 Node-based parallelization

The node-based framework was tested with 17 and 129 workers. First, we tried a running time of one hour, but no integer solutions were found and the lower bound was not improved above the LP bound of 116.08. Even when increasing the running time to 10 hours, the results were the same. The master process becomes a bottleneck and only around 1,200,000 nodes have been processed after 10 hours. During the branch-and-bound search for the same instance performed by Xpress on a single computer, around 2,500,000 nodes are searched. The lack of good lower and upper bounds results in few nodes being pruned. We are not able to match Xpress Optimizer's superior branching strategies in our framework. The lack of sophisticated methods using heuristics and cuts, makes our framework highly inefficient. We had hoped that sorting the node list with the up-child first would give better results as more nodes could be pruned earlier, but there was no difference from sorting with the down-child first. Due to the disappointing results, we did not test this framework further.

7.6.2 Tree-based parallelization

The second framework was first tested with one worker per computer, with the hope that internal parallelization and memory could be fully utilized by each worker. However, it turned out that each worker could only utilize 32 gigabytes of the 128 gigabytes of memory, and we were not able to resolve this issue. Therefore, the benefit of running one worker on one computer was reduced to being able to run internal parallelization in Xpress. Xpress Optimizer is able to utilize up to 50 parallel threads and having one worker per computer would waste 14 threads, since each computer has 64 cores. Also, using 50 threads could lead to communication overhead and slow down the processing. Therefore, we decided to not test the configuration with one worker per computer.

Instead, we tested three configurations with 32, 64 and 128 workers. We had 16 computers available, meaning that two, four and eight workers shared one computer in the respective configurations. When parallelizing work, the solution time should decrease to justify the increased use of resources. In theory, we should obtain the same solution as the sequential 10 hour run after around 40 minutes, since we use 16 computing nodes instead of one. We did not believe that our synchronous algorithm could achieve linear speedup and therefore we used a running time of five hours when testing. This is an overconsumption of resources, but we wanted to see if better solutions could be obtained. The focus has been on comparing solutions, and not lower bounds, since these are given by the weakest lower bound among all workers.

With 32 workers, there are two workers per computer and memory is wasted, since each worker is only able to utilize 32 gigabytes of the memory. However, each worker could utilize 32 threads for internal parallelization. The best solution, 222.00, is found after almost three hours. The best solution when running the sequential

TM instance, 219.72, was found after four hours. Therefore, no improvement was achieved with this parallel configuration.

Due to the limitation of only being able to utilize 32 gigabytes of the memory, we could run four workers on each computer, before each computer has less memory available. This gives a total of 64 workers. The best solution is 219.27 and it is found after a little more than two hours. This solution is slightly better than the best solution found by the sequential run and it is found faster. However, it is not nearly a linear decrease in solution time compared to the increase in processing power.

Eight workers per computer gives a total of 128 workers, allowing seven binary variables to be fixed. Each worker has 16 gigabytes of memory available on average. The different worker processes share the available memory of 32 gigabytes in a dynamic fashion, in stead of splitting it evenly. However, it appears that less nodes have been processed by each worker after 10 hours, possibly due to the reduced number of threads available for internal parallelization in Xpress. This configuration has a best solution of 222.66 after five hours and no improvement was achieved.

The configuration with 64 workers where four workers share one computer performed best among the parallel configurations tested. Increasing from 32 workers to 64 workers, makes it possible to have six binary variables fixed instead of five. This could help find better solutions faster. However, when increasing to 128 workers, the extra variable fixed does not seem to balance out the reduced number of cores available for internal parallelization. The best solution found using 64 workers in parallel is only marginally better than the best found by the sequential run and the decreased solution time does not justify the increased use of computing resources. In our case an asynchronous algorithm would not have performed better, since none of the workers found a lower bound worse than the best solution found so far. However, with more sophisticated fixing strategies or more variables fixed, an asynchronous algorithm could give improvements.

Figure 7.2 illustrates how the solution and lower bound improves with time for the TM instance when run sequentially, and by the worker who finds the best solution with the parallel configuration with four workers per node. We see that the parallel configuration finds good solutions faster, but they end at almost the same solution after around four hours. Note that it is only the solution process of one worker that is shown. If we had illustrated the solutions found by all 64 workers, the graph of the parallel algorithm would have shown more good solutions after less time.

7.7 Economical Results

In this section we will explore the solution found by the sequential run of the arc-flow instance TM for the largest test case. The objective function value of this

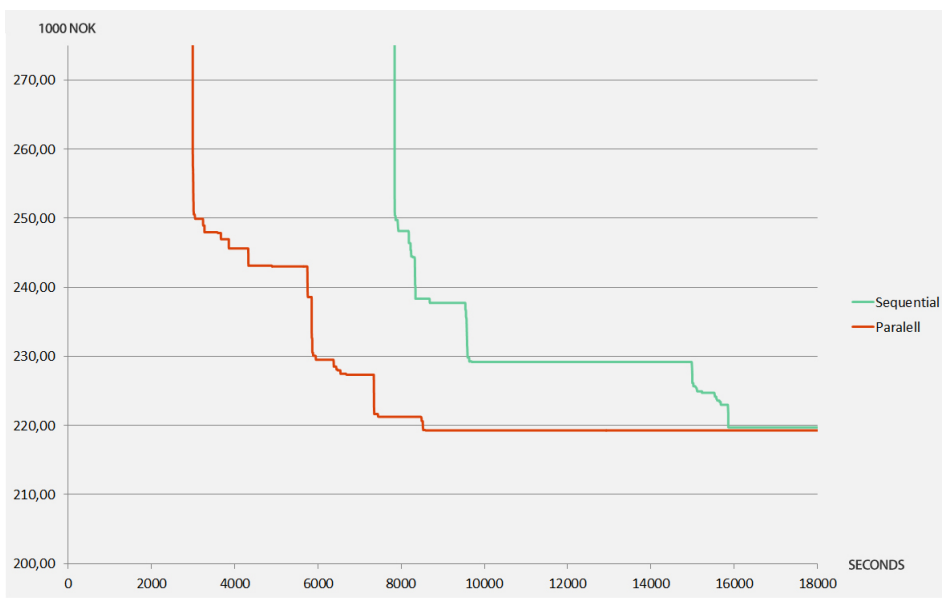


Figure 7.2: Illustration of the solutions found by the sequential and parallel run.

solution is 219,720 NOK, of which 192,000 NOK are transportation costs and the remaining 27,720 NOK are penalty costs for low stock levels. Penalty costs are not real monetary costs, but are used to encourage the creation of robust schedules. 12 fish farms have underage during the planning period, but for two of these, the underage is negligible. The stock levels of two other fish farms come down to half the safety stock level. If Marine Harvest prefers to avoid stock levels as low as this, the penalty cost should have been slightly higher. If not, the current parameterization is adequate, since the underage at most of the fish farms are between zero and less than 20% of the safety stock.

An illustration of the solution can be found in Figure 7.3. External supply is not used for any of the fish farms. Initial stock levels plus the production during the planning period is more than enough to cover the total demand of fish farms. The ships are not strained on capacity and it is reasonable that the model chooses internal supply of all fish farms since this is the cheapest option. However, it is possible that the absence of external supply comes from the cost parameters for internal and external supply being incorrect relative to each other, or that the production and ship capacity have been overestimated.

After investigating the solution further, we found that all fish farm visits are made within the first six days of the planning period, except the last visit to the factory made on the eighth day. The stock levels at fish farms end at their minimum stock levels, while the factory's stock level ends at 84% of capacity. Due to these

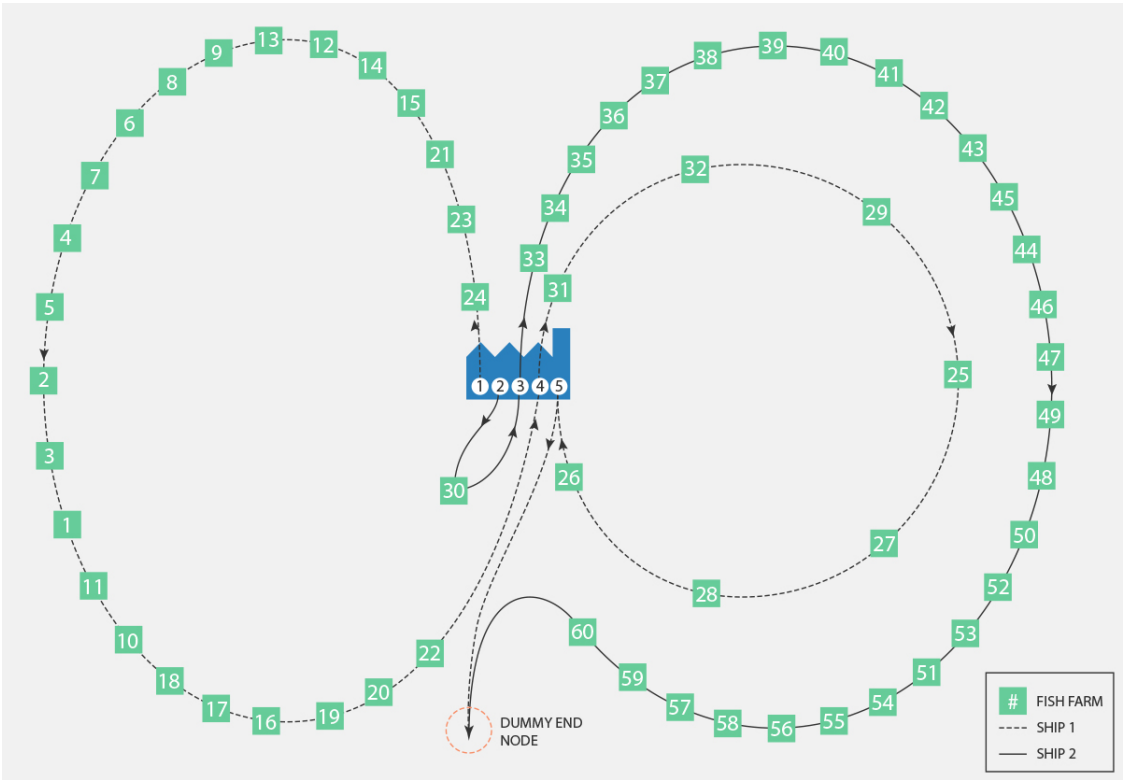


Figure 7.3: Illustration of the best solution found by the sequential run of the arc-flow instance TM.

EOH effects, the model should not be used to create a plan for the entire planning horizon. Instead, a rolling horizon should be used where the model is run for 10 days and then rerun at an appropriate replanning point after less than 10 days. In our solution, the results after approximately six days are acceptable, and this is a suitable replanning point. An easy way to implement this is to use the stock levels and loads at the point of replanning as initial data for the next model run. However, more sophisticated frameworks exist, as presented in Chapter 4.

Table 7.20 shows the production utilization. With a replanning point of around six days, full production utilization is achieved and the factory ends just below its initial stock level at 50% of capacity. Feed delivered is slightly more than feed produced. If this situation repeats, external supply will be necessary at some point. It is possible that the estimated production rate is too high, since operation of the factory has not yet started. Maintenance and changeover time between product types could lead to a lower production rate, and external supply could be necessary due to this as well.

	10 days	~6 days
Production	58%	105%

Table 7.20: Utilization of production for different replanning points.

Table 7.21 shows the ship utilization. The ship utilization increases significantly when changing the replanning point from ten to approximately six days and when initial waiting is disregarded. In our test cases, both ships start at the factory with zero initial load. The first ship waits for the factory to build up stock, before it leaves. The second ship has to wait at least 24 hours after the first ship starts loading before it can load, which gives it a lower utilization than the first ship. If we had used the situation after around six days as initial data, one ship would be empty, while the other would have a small load. Then, the empty ship could go directly to the factory for loading, while the other could unload at fish farms until it is empty and then sail to the factory. This would reduce the initial waiting time at the factory and give a more correct view of ship utilization. As can be seen in Figure 7.2, the second ship attempts to utilize the time while waiting for the factory to build up stock. First, it loads at the factory, before it supplies one fish farm. Then it returns to the factory and loads up to almost full capacity, before starting a longer route.

		10 days	~6 days
Ship 1	Initial waiting included	46%	82%
	Initial waiting excluded	50%	93%
Ship 2	Initial waiting included	28%	51%
	Initial waiting excluded	36%	81%

Table 7.21: Utilization of ship capacity for different replanning points.

Unloading starts within service hours for 22 of the fish farm visits. This is as expected since service hours make up one third of the total time. However, the additional capacity achieved when arriving within service hours is only utilized for five of these fish farm visits. This is in contrast with Marine Harvest’s view on the importance of filling up to maximum capacity. The additional capacity will probably be more important in real-life, were we believe the ships will be more strained on capacity. With multiple product types that can not be mixed, less capacity will be available on board ships. Delays or longer sailing times would also lead to less capacity. Then, it will become more important to utilize full storage capacity at fish farms and also external supply could be necessary.

Another important economical aspect of the solutions obtained by our model is that the cost from one test case to another does not increase with the same rate as the number of fish farms. In fact, when the number of fish farms double from 20 to 40, the cost of the best solution found only increases by 40%, while the increase is 60% when the number of fish farms is tripled. This is mainly because the area covered by fish farms does not increase much between test cases, it is only the

density of fish farms that increases. Therefore, it is not twice as costly to serve twice as many fish farms, as long as the ships have enough time and capacity. The difference could be even smaller, since the solutions found by the medium and large test cases are not optimal.

Chapter 8

Concluding Remarks

Secure and robust feed deliveries are important in the salmon industry where demands depend on biological factors. In order to remain competitive, this must be done in a cost-effective way. Marine Harvest is trying to achieve this through vertical integration and implementation of vendor managed inventory (VMI). In order to aid Marine Harvest in planning of feed deliveries to fish farms, we have in this thesis modeled their planning problem as an inventory routing problem (IRP).

Three formulations were developed and applied to three various-sized test cases, where the largest is the real-sized planning problem. The arc-flow formulation outperformed the arc-load and multi-commodity flow formulations, but it could not solve the two larger test cases to optimality. For the real-sized problem, the arc-flow formulation with tightening of timing constraints found a best solution of 219.72. The best bound, 166.58 was found when combining all cuts and this gives a gap of 24.2%. All formulations are data sensitive and we were only able to get good solutions when using appropriate initial stock levels relative to the planning horizon. We tried to balance this by using a rolling horizon approach, and succeeded to find a suitable replanning point after approximately six days.

Tightening of constraints and valid inequalities have been tested, but none of the attempted improvements were consistently performing better than others. Adding subtour elimination constraints performed best for the arc-flow formulation in the small and medium test case, while the best solution for the large test case was found when tightening timing constraints. Adding tightenings and inequalities did improve lower bounds in most cases, but the effects on the solutions were more volatile. Also, combining the different tightenings and inequalities gave stronger lower bounds, but solutions were not improved.

The best instance of the arc-flow formulation for the largest test case was embedded within two parallel branch-and-bound frameworks. The node-based framework could not find any integer solutions and the lower bound was not improved above the LP bound. The tree-based framework found a slightly better solution than

the sequential run, but the solution time was not decreased sufficiently to justify the increased use of computing resources. We conclude that a more sophisticated fixing strategy and an asynchronous implementation could improve the tree-based parallel framework.

In the best solution found, all fish farms are supplied internally. Since there are many estimations in our data, we believe that more fish farms in reality would need external supply, due to both lower production and ship capacity. With a replanning point of around six days, the model shows that production capacity is fully utilized. At this point, the factory is filled up to half capacity, one ship has a small load and the other is empty. If initial waiting is disregarded, ships are utilized at 81% and 93%, respectively, meaning there is some waiting on arrival when delivering to fish farms. This waiting can be regarded as slack in the schedule, making it more robust regarding transportation delays. The model creates robust schedules with little underage and high utilization of production capacity and ships are achieved.

Chapter 9

Future Research

Due to the complexity of the problem explored in this thesis, more research on the presented formulations and parallel frameworks are needed to be able to solve the real-sized problem to optimality. Possible improvements and extensions are plentiful and some will be presented in this chapter.

In order to further improve the arc-flow formulation, one approach is to find more valid inequalities or other ways to tighten the formulation. Other solution methods could also be explored. A common approach for IRP models is reformulation and decomposition. Due to the size of the problem, full enumeration of routes and schedules would not be possible, but a branch-and-price approach utilizing column generation could be viable. Another alternative is to include heuristic methods, which is also commonly used for IRPs.

Further work on the parallelization frameworks could be fruitful. Parallelizing the work of solving each branch-and-bound node was not efficient and it is difficult to compete with the advanced branching and bounding strategies of Xpress Optimizer. However, we believe that the second framework with parallel solving of subtrees could be a viable method, as a much larger part of the solution space can be searched in shorter time. One possible improvement is creating more sophisticated strategies on fixing variables. Another step forward would be to develop an asynchronous implementation of this framework. Also, it would be interesting to test with more workers and thereby, more fixed variables.

It would be relevant to add aspects that were simplified in this thesis. If data on different product types is made available, the model could be run as originally formulated. The simplification to aggregate silo capacities could then be evaluated, and if solutions are not feasible, necessary model adjustments should be made. Other aspects mentioned by Marine Harvest are the inclusion of production scheduling to the model and evaluation of different sailing speeds.

The formulations presented in this report are deterministic. Since feed demand

depends on salmon growth, it is subject to many uncertain factors. It could therefore be relevant to use a stochastic model instead, as this would be a more realistic description of the planning problem. Different demand scenarios could be generated, to account for variations and create more robust plans. Uncertainties related to sailing times and weather could also be included as stochastic elements. These extensions would come at the cost of increased complexity, which could make it difficult to solve even small versions of the problem.

Bibliography

- Adulyasak, Y., Cordeau, J.-F., and Jans, R. (2014). Formulations and branch-and-cut algorithms for multivehicle production and inventory routing problems. *INFORMS Journal on Computing*, 26(1):103–120.
- Agra, A., Christiansen, M., and Delgado, A. (2013a). Discrete time and continuous time formulations for a short sea inventory routing problem. University of Aveiro. Working paper. URL: <http://hdl.handle.net/10773/10561>.
- Agra, A., Christiansen, M., and Delgado, A. (2013b). Mixed integer formulations for a short sea fuel oil distribution problem. *Transportation Science*, 47(1):108–124.
- Agra, A., Christiansen, M., Delgado, A., and Simonetti, L. (2014). Hybrid heuristics for a short sea inventory routing problem. *European Journal of Operational Research*, 236(3):924 – 935.
- Al-Khayyal, F. and Hwang, S.-J. (2007). Inventory constrained maritime routing and scheduling for multi-commodity liquid bulk, part i: Applications and model. *European Journal of Operational Research*, 176(1):106–130.
- Andersson, H., Christiansen, M., and Fagerholt, K. (2011). The maritime pickup and delivery problem with time windows and split loads. *INFOR: Information Systems and Operational Research*, 49(2):79–91.
- Andersson, H., Hoff, A., Christiansen, M., Hasle, G., and Løkketangen, A. (2010). Industrial aspects and literature survey: Combined inventory management and routing. *Computers and Operations Research*, 37(9):1515–1536.
- Anthony, R. (1965). *Planning and control systems: a framework for analysis*, pages 15–23. Studies in management control. Division of Research, Graduate School of Business Administration, Harvard University.
- Archetti, C., Bertazzi, L., Laporte, G., and Speranza, M. G. (2007). A branch-and-cut algorithm for a vendor-managed inventory-routing problem. *Transportation Science*, 41(3):382–391.
- Balland, O. (2014). Data received from Océane Balland, Associate Professor

- II, Department of Marine Technology, Norwegian University of Science and Technology, Trondheim .
- Beale, E. M. L. and Tomlin, J. A. (1970). Special facilities in a general mathematical programming system for non-convex problems using ordered sets of variables. *OR*, 69:447–454.
- Bertazzi, L., Bosco, A., Guerriero, F., and Laganà, D. (2011). A stochastic inventory routing problem with stock-out. *Transportation Research Part C: Emerging Technologies*, 27:89–107.
- Bredström, D. and Rönnqvist, M. (2006). Supply chain optimization in pulp distribution using a rolling horizon solution approach. Norwegian School of Economics and Business Administration, Bergen. Discussion paper.
- Christiansen, M. (1999). Decomposition of a combined inventory and time constrained ship routing problem. *Transportation Science*, 33(1):3–16.
- Christiansen, M. and Fagerholt, K. (2009). Maritime inventory routing problems. In Floudas, C. and Pardalos, P., editors, *Encyclopedia of optimization*, pages 1947–1955. Springer Berlin, 2 edition.
- Christiansen, M., Fagerholt, K., Flatberg, T., Haugen Ø., Kloster, O., and Lund, E. H. (2011). Maritime inventory routing with multiple products: A case study from the cement industry. *European Journal of Operational Research*, 208(1):86–94.
- Christiansen, M., Fagerholt, K., Nygreen, B., and Ronen, D. (2006). Maritime transportation. *Transportation*, 14:189–284.
- Coase, R. H. (1937). The nature of the firm. *Economica*, 4(16):386–405.
- Coelho, L. C., Cordeau, J.-F., and Laporte, G. (2012). Dynamic and stochastic inventory-routing. Technical report, CIRRELT, Montreal.
- Coelho, L. C., Cordeau, J.-F., and Laporte, G. (2014). Thirty years of inventory routing. *Transportation Science*, 48(1):1–19.
- Coelho, L. C. and Laporte, G. (2013). The exact solution of several classes of inventory-routing problems. *Computers & Operations Research*, 40(2):558–565.
- Corrêa, R. and Ferreira, A. (1996). Parallel best-first branch-and-bound in discrete optimization: a framework. In *Solving Combinatorial Optimization Problems in Parallel*, pages 171–200. Springer.
- Dash Optimization (2007). Xpress optimizer reference manual. Dash Optimization Ltd., Englewood Cliffs, New Jersey.
- Dauzère-Pérès, S., Nordli, A., Olstad, A., Haugen, K., Koester, U., Myrstad, P. O., Teistklub, G., and Reistad, A. (2007). Omya hustadmarmor optimizes its supply chain for delivering calcium carbonate slurry to european paper manufacturers. *Interfaces*, 37(1):39–51.

- Eckstein, J. (1994a). Control strategies for parallel mixed integer branch and bound. In *Proceedings of the 1994 ACM/IEEE conference on Supercomputing*, pages 41–48. IEEE Computer Society Press.
- Eckstein, J. (1994b). Parallel branch-and-bound algorithms for general mixed integer programming on the cm-5. *SIAM Journal on Optimization*, 4(4):794–814.
- Eckstein, J., Phillips, C. A., and Hart, W. E. (2001). Pico: An object-oriented framework for parallel branch and bound. *Studies in Computational Mathematics*, 8:219–265.
- Egil Ulvan Rederi (2014). Data received from Egil Ulvan Rederi. Contact: Ivar Christian Ulvan.
- Engineer, F. G., Furman, K. C., Nemhauser, G. L., Savelsbergh, M. W. P., and Song, J.-H. (2012). A branch-price-and-cut algorithm for single-product maritime inventory routing. *Operations Research*, 60(1):106–122.
- Federgruen, A. and Simchi-Levi, D. (1995). Chapter 4 analysis of vehicle routing and inventory-routing problems. In M.O. Ball, T.L. Magnanti, C. M. and Nemhauser, G., editors, *Network Routing*, volume 8 of *Handbooks in Operations Research and Management Science*, pages 297–373. Elsevier Amsterdam.
- Fiskeridirektoratet (2013). Lønnsomhetsundersøkelse for matfiskproduksjon. <http://www.fiskeridir.no/statistikk/akvakultur/loennsomhet/matfiskproduksjon-laks-og-regnbueoerret/>. Data retrieved: 1. April, 2014.
- Gendron, B. and Crainic, T. G. (1994). Parallel branch-and-branch algorithms: Survey and synthesis. *Operations Research*, 42(6):1042–1066.
- Grama, A. Y., Gupta, A., and Kumar, V. (1993). Isoefficiency: Measuring the scalability of parallel algorithms and architectures. *IEEE Parallel Distrib. Technol.*, 1(3):12–21.
- Griva, I., Nash, S. G., and Sofer, A. (2009). *Linear and Nonlinear Optimization*. Society for Industrial Mathematics Pennsylvania.
- Grønhaug, R. and Christiansen, M. (2009). Supply chain optimization for the liquefied natural gas business. In Nunen, J. A., Speranza, M. G., and Bertazzi, L., editors, *Innovations in Distribution Logistics*, volume 619 of *Lecture Notes in Economics and Mathematical Systems*, pages 195–218. Springer Berlin.
- Grønhaug, R., Christiansen, M., Desaulniers, G., and Desrosiers, J. (2010). A branch-and-price method for a liquefied natural gas inventory routing problem. *Transportation Science*, 44(3):400–415.
- Hwang, S.-J. (2005). *Inventory constrained maritime routing and scheduling for multi-commodity liquid bulk*. PhD thesis, Georgia Institute of Technology, Atlanta.

- iLaks.no (2013). Størst og mest lønnsom. <http://ilaks.no/storst-og-mest-lonnsom/>. Data retrieved: 4. March, 2014.
- Ivarsøy, K. S. and Solhaug, I. E. (2013). Feed delivery from production facility to salmon farms. Norwegian University of Science and Technology, Trondheim. Project thesis.
- Kreps, D. M. (2004). *Microeconomics for managers*. W.W. Norton & Co. New York.
- Kumar, V. P. and Gupta, A. (1994). Analyzing scalability of parallel algorithms and architectures. *Journal of parallel and distributed computing*, 22(3):379–391.
- Lenstra, J. K. and Kan, A. H. G. R. (1981). Complexity of vehicle routing and scheduling problems. *Networks*, 11(2):221–227.
- Marine Harvest (2010). Seafood value chain. <http://www.marineharvest.com/en/Seafood-Value-Chain1/>. Data retrieved: 15. December, 2013.
- Marine Harvest (2012). Marine Harvest Norway’s regions. <http://marineharvest.com/no/Marine-Harvest-Norge/Om-Marine-Harvest1/Vare-regioner/>. Data retrieved: 4. March, 2014.
- Marine Harvest (2013a). Presentation Capital Markets Day. http://marineharvest.com/Global/Investor/Presentations/MHG_2013_CMD_PRESENTATION.pdf. Data retrieved: 4. March, 2014.
- Marine Harvest (2013b). The Marine Harvest Salmon Industry Handbook. <http://hugin.info/209/R/1698446/559980.pdf>. Data retrieved: 4. March, 2014.
- Marine Harvest (2014). Data received from Marine Harvest. Contact: Espen Moksnes, Nora Hindar, Vidar Myhre and Eivind Osnes.
- Olafsen, T., Winther, U., Olsen, Y., and Skjermo, J. (2012). Verdiskaping basert på produktive hav i 2050. http://www.sintef.no/upload/Fiskeri_og_havbruk/Publikasjoner/%20basert%20p%C3%A5%20produktive%20hav%20i%202050.pdf. Data retrieved: 4. March, 2014.
- Popović, D., Vidović, M., and Radivojević, G. (2012). Variable neighborhood search heuristic for the inventory routing problem in fuel delivery. *Expert Syst. Appl.*, 39(18):13390–13398.
- Rakke, J. G., Andersson, H., Christiansen, M., and Desaulniers, G. (2014). A new formulation based on customer delivery patterns for a maritime inventory routing problem. <http://pubsonline.informs.org/doi/abs/10.1287/trsc.2013.0503>.
- Rakke, J. G., Stålhane, M., Moe, C. R., Christiansen, M., Andersson, H., Fagerholt, K., and Norstad, I. (2011). A rolling horizon heuristic for creating a liquefied natural gas annual delivery program. *Transportation Research Part C: Emerging Technologies*, 19(5):896–911.

- Ralphs, T. K. (2003). Parallel branch and cut for capacitated vehicle routing. *Parallel Computing*, 29(5):607–629.
- Ralphs, T. K., Ladányi, L., and Saltzman, M. J. (2004). A library hierarchy for implementing scalable parallel search algorithms. *The Journal of Supercomputing*, 28(2):215–234.
- Ronen, D. (2002). Marine inventory routing: shipments planning. *Journal of the Operational Research Society*, 53(1):108–114.
- Simchi-Levi, D., Simchi-Levi, E., and Kaminsky, P. (1999). *Designing and managing the supply chain: Concepts, strategies, and cases*. McGraw-Hill United-States.
- Skretting (2009). Formler og beregninger. [http://www.skretting.no/Internet/SkrettingNorway/webInternet.nsf/wprid/2E81AEC2B788F022C125757F0036473C/\\$file/Foring_formler.pdf](http://www.skretting.no/Internet/SkrettingNorway/webInternet.nsf/wprid/2E81AEC2B788F022C125757F0036473C/$file/Foring_formler.pdf). Data retrieved: 4. March, 2014.
- Solyali, O. and Süral, H. (2011). A branch-and-cut algorithm using a strong formulation and an a priori tour-based heuristic for an inventory-routing problem. *Transportation Science*, 45(3):335–345.
- Song, J.-H. and Furman, K. C. (2013). A maritime inventory routing problem: Practical approach. *Computers & Operations Research*, 40(3):657 – 665.
- Stålhane, M., Andersson, H., Christiansen, M., Cordeau, J.-F., and Desaulniers, G. (2012). A branch-price-and-cut method for a ship routing and scheduling problem with split loads. *Computers & Operations Research*, 39(12):3361–3375.
- Statistisk sentralbyrå (2013). Akvakultur, 2012. <http://www.ssb.no/fiskeoppdrett/>. Data retrieved: 1. April, 2014.
- Statistisk sentralbyrå (2014). Eksport av laks. <http://www.ssb.no/laks/>. Data retrieved: 30. May, 2014.
- Trienekens, H. W. and Bruin, A. d. (1992). Towards a taxonomy of parallel branch and bound algorithms. Technical report, Erasmus School of Economics (ESE), Rotterdam.
- Vanderbeck, F. and Wolsey, L. (2010). Reformulation and decomposition of integer programs. In Jünger, M., Liebling, T. M., Naddef, D., Nemhauser, G. L., Pulleyblank, W. R., Reinelt, G., Rinaldi, G., and Wolsey, L. A., editors, *50 Years of Integer Programming 1958-2008*, pages 431–502. Springer Berlin Heidelberg.

Appendix A

Complete Arc-Load Formulation

A.1 Definitions

A.1.1 Sets

\mathcal{N}^P	Set of factories
\mathcal{N}^C	Set of fish farms
\mathcal{M}_i	Set of visit numbers for location i
\mathcal{V}	Set of ships
\mathcal{S}^P	Set of factory visits (i, m) where $i \in \mathcal{N}^P$ and $m \in \mathcal{M}_i$
\mathcal{S}^C	Set of fish farm visits (i, m) where $i \in \mathcal{N}^C$ and $m \in \mathcal{M}_i$
\mathcal{S}	Set of visits, $\mathcal{S} = \mathcal{S}^P \cup \mathcal{S}^C$
\mathcal{S}_v	Set of feasible visits for ship v , $\mathcal{S}_v = \mathcal{S} \cup \{d(v)\}$
\mathcal{P}	Set of products
\mathcal{D}	Set of days within the planning horizon

A.1.2 Indices

i, j	Locations
$o(v)$	Start node of ship v
$d(v)$	Dummy end node of ship v
m, n	Visit numbers
v	Ships
p	Products
d	Days

A.1.3 Parameters

B	Penalty cost for stock levels below the safety stock level	[NOK/hour]
C_{ij}	Transportation cost for sailing from location i to location j	[NOK]
E_p	Unit cost of buying product p externally	[NOK/ton]
E_i^T	Transportation cost for external feed delivery to fish farm i	[NOK]
L_{vp}^0	Initial load of product p on ship v	[tons]
K_v^{MAX}	Maximum capacity for ship v	[tons]
Q_{ip}^{MIN}	Minimum unloading quantity of product p for fish farm i	[tons]
S_i^0	Initial inventory level at factory i	[tons]
S_{ip}^0	Initial inventory level of product p at fish farm i	[tons]
S_{ip}^{MIN}	Minimum inventory level of product p at fish farm i	[tons]
S_i^{MAX}	Maximum inventory level at location i	[tons]
A_i	Reduction in storage capacity at fish farm i outside working hours	[%]
T^{MAX}	Length of planning period	[hours]
T_{ij}^S	Sailing time from location i to location j	[hours]
T_i^L	Loading or unloading time per ton of feed for location i	[hours/ton]
T_d^{WS}	Start of service hours for day d	[hours]
T_d^{WE}	End of service hours for day d	[hours]
J_i	Location type for location i , 1 for factories and -1 for fish farms	
R_{ip}	Consumption rate of product p at fish farm i	[tons/hour]
P_i	Production rate at factory i	[tons/hour]
H_i	Time between visits to location i	[hours]

A.1.4 Decision variables

x_{imjnv}	1 if ship v sails from visit (i, m) to visit (j, n) , else 0	
y_{im}	1 if visit (i, m) is not made by any ship, else 0	
u_i	1 if fish farm i is supplied internally, 0 if supplied externally	
q_{imvp}	Amount of product p loaded/unloaded by ship v during visit (i, m)	[tons]
l_{imvp}	Amount of product p on board ship v when leaving visit (i, m)	[tons]
s_{im}	Amount of feed in stock at the start of visit $(i, m) \in \mathcal{S}^P$	[tons]
s_{im}^E	Amount of feed in stock at the end of visit $(i, m) \in \mathcal{S}^P$	[tons]
s_{imp}	Amount of product p in stock at the start of visit $(i, m) \in \mathcal{S}^C$	[tons]
s_{imp}^E	Amount of product p in stock at the end of visit $(i, m) \in \mathcal{S}^C$	[tons]
d_{imp}	Amount of product p below S_{ip}^{MIN} at the start of visit $(i, m) \in \mathcal{S}^C$	[tons]
t_{im}	Time for start of service for visit (i, m)	[hours]
t_{im}^E	Time for end of service for visit (i, m) ,	[hours]
σ_{imd}	1 if visit $(i, m) \in \mathcal{S}^C$ is within service hours on day d , else 0	

A.2 Model Formulation

A.2.1 Objective function

$$\begin{aligned} \min z = & \sum_{(i,m) \in \mathcal{S}} \sum_{(j,n) \in \mathcal{S}} \sum_{v \in \mathcal{V}} C_{ij} x_{imjnv} + T^{MAX} \sum_{i \in \mathcal{N}^C} \sum_{p \in \mathcal{P}} E_p R_{ip} (1 - u_i) \\ & + \sum_{i \in \mathcal{N}^C} E_i^T (1 - u_i) + \sum_{(i,m) \in \mathcal{S}^C} \sum_{p \in \mathcal{P}} B \frac{d_{imp}}{R_{ip}} \end{aligned} \quad (\text{A.1})$$

A.2.2 Routing constraints

$$\sum_{(j,n) \in \mathcal{S}_v} x_{o(v)jnv} = 1 \quad v \in \mathcal{V} \quad (\text{A.2})$$

$$\sum_{(j,n) \in \mathcal{S}} x_{jnimv} - \sum_{(j,n) \in \mathcal{S}_v} x_{imjnv} = 0 \quad (i,m) \in \mathcal{S} \setminus \{o(v)\}, v \in \mathcal{V} \quad (\text{A.3})$$

$$\sum_{(j,n) \in \mathcal{S}} x_{jnd(v)v} = 1 \quad v \in \mathcal{V} \quad (\text{A.4})$$

$$\sum_{(j,n) \in \mathcal{S}_v} \sum_{v \in \mathcal{V}} x_{imjnv} = 1 - y_{im} \quad (i,m) \in \mathcal{S} \quad (\text{A.5})$$

$$y_{im} - y_{im-1} \geq 0 \quad (i,m) \in \mathcal{S} | m > 1 \quad (\text{A.6})$$

$$u_i = 1 - y_{i1} \quad i \in \mathcal{N}^C \quad (\text{A.7})$$

A.2.3 Loading and unloading constraints

$$L_{vp}^0 + J_i q_{o(v)vp} = l_{o(v)vp} \quad v \in \mathcal{V}, p \in \mathcal{P} \quad (\text{A.8})$$

$$\begin{aligned} x_{imjnv} (l_{imvp} + J_j q_{jnv} - l_{jnv}) = 0 \\ (i,m) \in \mathcal{S}, (j,n) \in \mathcal{S}_v \setminus \{o(v)\}, v \in \mathcal{V}, p \in \mathcal{P} \end{aligned} \quad (\text{A.9})$$

$$\sum_{p \in \mathcal{P}} l_{imvp} \leq \sum_{(j,n) \in \mathcal{S}_v} K_v^{MAX} x_{imjnv} \quad (i,m) \in \mathcal{S}^P, v \in \mathcal{V} \quad (\text{A.10})$$

$$\sum_{p \in \mathcal{P}} l_{imvp} \leq \sum_{(j,n) \in \mathcal{S}_v} K_v^{MAX} x_{imjnv} - \sum_{p \in \mathcal{P}} q_{imvp} \quad (i, m) \in \mathcal{S}^C, v \in \mathcal{V} \quad (\text{A.11})$$

$$\sum_{p \in \mathcal{P}} q_{imvp} \leq \sum_{(j,n) \in \mathcal{S}_v} S_i^{MAX} x_{imjnv} \quad (i, m) \in \mathcal{S}^C, v \in \mathcal{V} \quad (\text{A.12})$$

$$\sum_{(j,n) \in \mathcal{S}_v} Q_{ip}^{MIN} x_{imjnv} \leq q_{imvp} \quad (i, m) \in \mathcal{S}^C, v \in \mathcal{V}, p \in \mathcal{P} \quad (\text{A.13})$$

$$\sum_{v \in \mathcal{V}} \sum_{p \in \mathcal{P}} q_{imvp} \leq s_{im} + S_i^{MAX} y_{im} \quad (i, m) \in \mathcal{S}^P \quad (\text{A.14})$$

$$\sum_{v \in \mathcal{V}} \sum_{p \in \mathcal{P}} q_{imvp} \geq s_{im} - S_i^{MAX} y_{im} \quad (i, m) \in \mathcal{S}^P \quad (\text{A.15})$$

A.2.4 Inventory constraints

$$S_i^0 + P_i t_{im} = s_{im} \quad (i, m) \in \mathcal{S}^P | m = 1 \quad (\text{A.16})$$

$$S_{ip}^0 - R_{ip} t_{im} = s_{imp} \quad (i, m) \in \mathcal{S}^C | m = 1, p \in \mathcal{P} \quad (\text{A.17})$$

$$s_{im} + P_i (t_{im}^E - t_{im}) - \sum_{v \in \mathcal{V}} \sum_{p \in \mathcal{P}} q_{imvp} = s_{im}^E \quad (i, m) \in \mathcal{S}^P \quad (\text{A.18})$$

$$s_{imp} - R_{ip} (t_{im}^E - t_{im}) + \sum_{v \in \mathcal{V}} q_{imvp} = s_{imp}^E \quad (i, m) \in \mathcal{S}^C, p \in \mathcal{P} \quad (\text{A.19})$$

$$s_{i(m-1)}^E + P_i (t_{im} - t_{i(m-1)}^E) = s_{im} \quad (i, m) \in \mathcal{S}^P | m > 1 \quad (\text{A.20})$$

$$s_{i(m-1)p}^E - R_{ip} (t_{im} - t_{i(m-1)}^E) = s_{imp} \quad (i, m) \in \mathcal{S}^C | m > 1, p \in \mathcal{P} \quad (\text{A.21})$$

$$s_{im} \leq S_i^{MAX} \quad (i, m) \in \mathcal{S}^P \quad (\text{A.22})$$

$$\sum_{p \in \mathcal{P}} s_{imp}^E \leq (1 - A_i) S_i^{MAX} + A_i S_i^{MAX} \sum_{d \in \mathcal{D}} \sigma_{imd} \quad (i, m) \in \mathcal{S}^C \quad (\text{A.23})$$

$$S_{ip}^{MIN} u_i \leq s_{imp} + d_{imp} \quad (i, m) \in \mathcal{S}^C, p \in \mathcal{P} \quad (\text{A.24})$$

$$d_{imp} \leq S_{ip}^{MIN} u_i \quad (i, m) \in \mathcal{S}^C, p \in \mathcal{P} \quad (\text{A.25})$$

$$s_{im}^E + P_i(T^{MAX} - t_{im}^E) \leq S_i^{MAX} \quad (i, m) \in \mathcal{S}^P | m = |\mathcal{M}_i| \quad (\text{A.26})$$

$$\begin{aligned} S_{ip}^{MIN} u_i + R_{ip} T^{MAX} u_i - R_{ip} t_{im}^E &\leq s_{imp}^E \\ (i, m) \in \mathcal{S}^C | m = |\mathcal{M}_i|, p \in \mathcal{P} \end{aligned} \quad (\text{A.27})$$

A.2.5 Timing constraints

$$t_{im} + \sum_{v \in \mathcal{V}} \sum_{p \in \mathcal{P}} T_i^L q_{imvp} = t_{im}^E \quad (i, m) \in \mathcal{S} \quad (\text{A.28})$$

$$\sum_{v \in \mathcal{V}} x_{imjnv} (t_{im}^E + T_{ij}^S - t_{jn}) \leq 0 \quad (i, m), (j, n) \in \mathcal{S} \quad (\text{A.29})$$

$$t_{im}^E \leq T^{MAX} \quad (i, m) \in \mathcal{S} \quad (\text{A.30})$$

$$t_{im} \leq T^{MAX} u_i \quad (i, m) \in \mathcal{S}^C \quad (\text{A.31})$$

$$y_{im} (t_{im} - t_{i(m-1)}^E) = 0 \quad (i, m) \in \mathcal{S} | m > 1 \quad (\text{A.32})$$

$$\begin{aligned} T_d^{WS} - T^{MAX} (1 - \sigma_{imd} + y_{im}) &\leq t_{im} \leq T_d^{WE} + T^{MAX} (1 - \sigma_{imd} + y_{im}) \\ (i, m) \in \mathcal{S}^C, d \in \mathcal{D} \end{aligned} \quad (\text{A.33})$$

$$\sum_{d \in \mathcal{D}} \sigma_{imd} \leq 1 \quad (i, m) \in \mathcal{S}^C \quad (\text{A.34})$$

$$\sum_{d \in \mathcal{D}} \sigma_{imd} \geq y_{im} \quad (i, m) \in \mathcal{S}^C \quad (\text{A.35})$$

$$t_{im} + H_i (1 - y_{i(m+1)}) \leq t_{im+1} \quad (i, m) \in \mathcal{S} \quad (\text{A.36})$$

A.2.6 Variable constraints

$$x_{imjnv} \in \{0, 1\} \quad (i, m) \in \mathcal{S}, (j, n) \in \mathcal{S}_v, v \in \mathcal{V} \quad (\text{A.37})$$

$$y_{im} \in \{0, 1\} \quad (i, m) \in \mathcal{S} \quad (\text{A.38})$$

$$u_i \in \{0, 1\} \quad i \in \mathcal{N}^C \quad (\text{A.39})$$

$$\sigma_{imd} \in \{0, 1\} \quad (i, m) \in \mathcal{S}^C, d \in \mathcal{D} \quad (\text{A.40})$$

$$q_{imvp} \geq 0 \quad (i, m) \in \mathcal{S}, v \in \mathcal{V}, p \in \mathcal{P} \quad (\text{A.41})$$

$$l_{imvp} \geq 0 \quad (i, m) \in \mathcal{S}, v \in \mathcal{V}, p \in \mathcal{P} \quad (\text{A.42})$$

$$s_{im} \geq 0 \quad (i, m) \in \mathcal{S}^P \quad (\text{A.43})$$

$$s_{im}^E \geq 0 \quad (i, m) \in \mathcal{S}^P \quad (\text{A.44})$$

$$s_{imp} \geq 0 \quad (i, m) \in \mathcal{S}^C, p \in \mathcal{P} \quad (\text{A.45})$$

$$s_{imp}^E \geq 0 \quad (i, m) \in \mathcal{S}^C, p \in \mathcal{P} \quad (\text{A.46})$$

$$d_{imp} \geq 0 \quad (i, m) \in \mathcal{S}^C, p \in \mathcal{P} \quad (\text{A.47})$$

$$t_{im} \geq 0 \quad (i, m) \in \mathcal{S} \quad (\text{A.48})$$

$$t_{im}^E \geq 0 \quad (i, m) \in \mathcal{S} \quad (\text{A.49})$$

Appendix B

Complete Arc-Flow Formulation

B.1 Definitions

B.1.1 Sets

\mathcal{N}^P	Set of factories
\mathcal{N}^C	Set of fish farms
\mathcal{M}_i	Set of visit numbers for location i
\mathcal{V}	Set of ships
\mathcal{S}^P	Set of factory visits (i, m) where $i \in \mathcal{N}^P$ and $m \in \mathcal{M}_i$
\mathcal{S}^C	Set of fish farm visits (i, m) where $i \in \mathcal{N}^C$ and $m \in \mathcal{M}_i$
\mathcal{S}	Set of visits, $\mathcal{S} = \mathcal{S}^P \cup \mathcal{S}^C$
\mathcal{S}_v	Set of feasible visits for ship v , $\mathcal{S}_v = \mathcal{S} \cup \{d(v)\}$
\mathcal{P}	Set of products
\mathcal{D}	Set of days within the planning horizon

B.1.2 Indices

i, j	Locations
$o(v)$	Start node of ship v
$d(v)$	Dummy end node of ship v
m, n	Visit numbers
v	Ships
p	Products
d	Days

B.1.3 Parameters

B	Penalty cost for stock levels below the safety stock level	[NOK/hour]
C_{ij}	Transportation cost for sailing from location i to location j	[NOK]
E_p	Unit cost of buying product p externally	[NOK/ton]
E_i^T	Transportation cost for external feed delivery to fish farm i	[NOK]
L_{vp}^0	Initial load of product p on ship v	[tons]
K_v^{MAX}	Maximum capacity for ship v	[tons]
Q_{ip}^{MIN}	Minimum unloading quantity of product p for fish farm i	[tons]
S_i^0	Initial inventory level at factory i	[tons]
S_{ip}^0	Initial inventory level of product p at fish farm i	[tons]
S_{ip}^{MIN}	Minimum inventory level of product p at fish farm i	[tons]
S_i^{MAX}	Maximum inventory level at location i	[tons]
A_i	Reduction in storage capacity at fish farm i outside working hours	[%]
T^{MAX}	Length of planning period	[hours]
T_{ij}^S	Sailing time from location i to location j	[hours]
T_i^L	Loading or unloading time per ton of feed for location i	[hours/ton]
T_d^{WS}	Start of service hours for day d	[hours]
T_d^{WE}	End of service hours for day d	[hours]
J_i	Location type for location i , 1 for factories and -1 for fish farms	
R_{ip}	Consumption rate of product p at fish farm i	[tons/hour]
P_i	Production rate at factory i	[tons/hour]
H_i	Time between visits to location i	[hours]

B.1.4 Decision variables

x_{imjnv}	1 if ship v sails from visit (i, m) to visit (j, n) , else 0	
y_{im}	1 if visit (i, m) is not made by any ship, else 0	
u_i	1 if fish farm i is supplied internally, 0 if supplied externally	
q_{imvp}	Amount of product p loaded/unloaded by ship v during visit (i, m)	[tons]
l_{imjnv}	Amount of product p on board ship v when traveling on arc (i, m, j, n)	[tons]
s_{im}	Amount of feed in stock at the start of visit $(i, m) \in \mathcal{S}^P$	[tons]
s_{im}^E	Amount of feed in stock at the end of visit $(i, m) \in \mathcal{S}^P$	[tons]
s_{imp}	Amount of product p in stock at the start of visit $(i, m) \in \mathcal{S}^C$	[tons]
s_{imp}^E	Amount of product p in stock at the end of visit $(i, m) \in \mathcal{S}^C$	[tons]
d_{imp}	Amount of product p below S_{ip}^{MIN} at the start of visit $(i, m) \in \mathcal{S}^C$	[tons]
t_{im}	Time for start of service for visit (i, m)	[hours]
t_{im}^E	Time for end of service for visit (i, m) ,	[hours]
σ_{imd}	1 if visit $(i, m) \in \mathcal{S}^C$ is within service hours on day d , else 0	

B.2 Model Formulation

B.2.1 Objective function

$$\begin{aligned} \min z = & \sum_{(i,m) \in \mathcal{S}} \sum_{(j,n) \in \mathcal{S}} \sum_{v \in \mathcal{V}} C_{ij} x_{imjnv} + T^{MAX} \sum_{i \in \mathcal{N}^C} \sum_{p \in \mathcal{P}} E_p R_{ip} (1 - u_i) \\ & + \sum_{i \in \mathcal{N}^C} E_i^T (1 - u_i) + \sum_{(i,m) \in \mathcal{S}^C} \sum_{p \in \mathcal{P}} B \frac{d_{imp}}{R_{ip}} \end{aligned} \quad (\text{B.1})$$

B.2.2 Routing constraints

$$\sum_{(j,n) \in \mathcal{S}_v} x_{o(v)jnv} = 1 \quad v \in \mathcal{V} \quad (\text{B.2})$$

$$\sum_{(j,n) \in \mathcal{S}} x_{jnimv} - \sum_{(j,n) \in \mathcal{S}_v} x_{imjnv} = 0 \quad (i, m) \in \mathcal{S} \setminus \{o(v)\}, v \in \mathcal{V} \quad (\text{B.3})$$

$$\sum_{(j,n) \in \mathcal{S}} x_{jnd(v)v} = 1 \quad v \in \mathcal{V} \quad (\text{B.4})$$

$$\sum_{(j,n) \in \mathcal{S}_v} \sum_{v \in \mathcal{V}} x_{imjnv} = 1 - y_{im} \quad (i, m) \in \mathcal{S} \quad (\text{B.5})$$

$$y_{im} - y_{im-1} \geq 0 \quad (i, m) \in \mathcal{S} | m > 1 \quad (\text{B.6})$$

$$u_i = 1 - y_{i1} \quad i \in \mathcal{N}^C \quad (\text{B.7})$$

B.2.3 Loading and unloading constraints

$$L_{vp}^0 + J_i q_{o(v)vp} = \sum_{(j,n) \in \mathcal{S}_v} l_{o(v)jnv} \quad v \in \mathcal{V}, p \in \mathcal{P} \quad (\text{B.8})$$

$$\begin{aligned} \sum_{(j,n) \in \mathcal{S}} l_{jnimvp} + J_i q_{imvp} - \sum_{(j,n) \in \mathcal{S}_v} l_{imjnv} &= 0 \\ (i, m) \in \mathcal{S} \setminus \{o(v)\}, v \in \mathcal{V}, p \in \mathcal{P} \end{aligned} \quad (\text{B.9})$$

$$\sum_{p \in \mathcal{P}} l_{imjnv} \leq K_v^{MAX} x_{imjnv} \quad (i, m) \in \mathcal{S}, (j, n) \in \mathcal{S}_v, v \in \mathcal{V} \quad (\text{B.10})$$

$$\sum_{p \in \mathcal{P}} q_{imvp} \leq \sum_{(j,n) \in \mathcal{S}_v} S_i^{MAX} x_{imjnv} \quad (i, m) \in \mathcal{S}^C, v \in \mathcal{V} \quad (\text{B.11})$$

$$\sum_{(j,n) \in \mathcal{S}_v} Q_{ip}^{MIN} x_{imjnv} \leq q_{imvp} \quad (i, m) \in \mathcal{S}^C, v \in \mathcal{V}, p \in \mathcal{P} \quad (\text{B.12})$$

$$\sum_{v \in \mathcal{V}} \sum_{p \in \mathcal{P}} q_{imvp} \leq s_{im} + S_i^{MAX} y_{im} \quad (i, m) \in \mathcal{S}^P \quad (\text{B.13})$$

$$\sum_{v \in \mathcal{V}} \sum_{p \in \mathcal{P}} q_{imvp} \geq s_{im} - S_i^{MAX} y_{im} \quad (i, m) \in \mathcal{S}^P \quad (\text{B.14})$$

B.2.4 Inventory constraints

$$S_i^0 + P_i t_{im} = s_{im} \quad (i, m) \in \mathcal{S}^P | m = 1 \quad (\text{B.15})$$

$$S_{ip}^0 - R_{ip} t_{im} = s_{imp} \quad (i, m) \in \mathcal{S}^C | m = 1, p \in \mathcal{P} \quad (\text{B.16})$$

$$s_{im} + P_i(t_{im}^E - t_{im}) - \sum_{v \in \mathcal{V}} \sum_{p \in \mathcal{P}} q_{imvp} = s_{im}^E \quad (i, m) \in \mathcal{S}^P \quad (\text{B.17})$$

$$s_{imp} - R_{ip}(t_{im}^E - t_{im}) + \sum_{v \in \mathcal{V}} q_{imvp} = s_{imp}^E \quad (i, m) \in \mathcal{S}^C, p \in \mathcal{P} \quad (\text{B.18})$$

$$s_{i(m-1)}^E + P_i(t_{im} - t_{i(m-1)}^E) = s_{im} \quad (i, m) \in \mathcal{S}^P | m > 1 \quad (\text{B.19})$$

$$s_{i(m-1)p}^E - R_{ip}(t_{im} - t_{i(m-1)}^E) = s_{imp} \quad (i, m) \in \mathcal{S}^C | m > 1, p \in \mathcal{P} \quad (\text{B.20})$$

$$s_{im} \leq S_i^{MAX} \quad (i, m) \in \mathcal{S}^P \quad (\text{B.21})$$

$$\sum_{p \in \mathcal{P}} s_{imp}^E \leq (1 - A_i) S_i^{MAX} + A_i S_i^{MAX} \sum_{d \in \mathcal{D}} \sigma_{imd} \quad (i, m) \in \mathcal{S}^C \quad (\text{B.22})$$

$$S_{ip}^{MIN} u_i \leq s_{imp} + d_{imp} \quad (i, m) \in \mathcal{S}^C, p \in \mathcal{P} \quad (\text{B.23})$$

$$d_{imp} \leq S_{ip}^{MIN} u_i \quad (i, m) \in \mathcal{S}^C, p \in \mathcal{P} \quad (\text{B.24})$$

$$s_{im}^E + P_i(T^{MAX} - t_{im}^E) \leq S_i^{MAX} \quad (i, m) \in \mathcal{S}^P | m = |\mathcal{M}_i| \quad (\text{B.25})$$

$$S_{ip}^{MIN} u_i + R_{ip} T^{MAX} u_i - R_{ip} t_{im}^E \leq s_{imp}^E \quad (i, m) \in \mathcal{S}^C | m = |\mathcal{M}_i|, p \in \mathcal{P} \quad (\text{B.26})$$

B.2.5 Timing constraints

$$t_{im} + \sum_{v \in \mathcal{V}} \sum_{p \in \mathcal{P}} T_i^L q_{imvp} = t_{im}^E \quad (i, m) \in \mathcal{S} \quad (\text{B.27})$$

$$\sum_{v \in \mathcal{V}} x_{imjnv} (t_{im}^E + T_{ij}^S - t_{jn}) \leq 0 \quad (i, m), (j, n) \in \mathcal{S} \quad (\text{B.28})$$

$$t_{im}^E \leq T^{MAX} \quad (i, m) \in \mathcal{S} \quad (\text{B.29})$$

$$t_{im} \leq T^{MAX} u_i \quad (i, m) \in \mathcal{S}^C \quad (\text{B.30})$$

$$y_{im} (t_{im} - t_{i(m-1)}^E) = 0 \quad (i, m) \in \mathcal{S} | m > 1 \quad (\text{B.31})$$

$$T_d^{WS} - T^{MAX} (1 - \sigma_{imd} + y_{im}) \leq t_{im} \leq T_d^{WE} + T^{MAX} (1 - \sigma_{imd} + y_{im}) \quad (i, m) \in \mathcal{S}^C, d \in \mathcal{D} \quad (\text{B.32})$$

$$\sum_{d \in \mathcal{D}} \sigma_{imd} \leq 1 \quad (i, m) \in \mathcal{S}^C \quad (\text{B.33})$$

$$\sum_{d \in \mathcal{D}} \sigma_{imd} \geq y_{im} \quad (i, m) \in \mathcal{S}^C \quad (\text{B.34})$$

$$t_{im} + H_i (1 - y_{i(m+1)}) \leq t_{i(m+1)} \quad (i, m) \in \mathcal{S} \quad (\text{B.35})$$

B.2.6 Variable constraints

$$x_{imjnv} \in \{0, 1\} \quad (i, m) \in \mathcal{S}, (j, n) \in \mathcal{S}_v, v \in \mathcal{V} \quad (\text{B.36})$$

$$y_{im} \in \{0, 1\} \quad (i, m) \in \mathcal{S} \quad (\text{B.37})$$

$$u_i \in \{0, 1\} \quad i \in \mathcal{N}^C \quad (\text{B.38})$$

$$\sigma_{imd} \in \{0, 1\} \quad (i, m) \in \mathcal{S}^C, d \in \mathcal{D} \quad (\text{B.39})$$

$$q_{imvp} \geq 0 \quad (i, m) \in \mathcal{S}, v \in \mathcal{V}, p \in \mathcal{P} \quad (\text{B.40})$$

$$l_{imjnv} \geq 0 \quad (i, m) \in \mathcal{S}, (j, n) \in \mathcal{S}_v, v \in \mathcal{V}, p \in \mathcal{P} \quad (\text{B.41})$$

$$s_{im} \geq 0 \quad (i, m) \in \mathcal{S}^P \quad (\text{B.42})$$

$$s_{im}^E \geq 0 \quad (i, m) \in \mathcal{S}^P \quad (\text{B.43})$$

$$s_{imp} \geq 0 \quad (i, m) \in \mathcal{S}^C, p \in \mathcal{P} \quad (\text{B.44})$$

$$s_{imp}^E \geq 0 \quad (i, m) \in \mathcal{S}^C, p \in \mathcal{P} \quad (\text{B.45})$$

$$d_{imp} \geq 0 \quad (i, m) \in \mathcal{S}^C, p \in \mathcal{P} \quad (\text{B.46})$$

$$t_{im} \geq 0 \quad (i, m) \in \mathcal{S} \quad (\text{B.47})$$

$$t_{im}^E \geq 0 \quad (i, m) \in \mathcal{S} \quad (\text{B.48})$$

Appendix C

Complete Multi-Commodity Flow Formulation

C.1 Definitions

C.1.1 Sets

\mathcal{N}^P	Set of factories
\mathcal{N}^C	Set of fish farms
\mathcal{M}_i	Set of visit numbers for location i
\mathcal{V}	Set of ships
\mathcal{S}^P	Set of factory visits (i, m) where $i \in \mathcal{N}^P$ and $m \in \mathcal{M}_i$
\mathcal{S}^C	Set of fish farm visits (i, m) where $i \in \mathcal{N}^C$ and $m \in \mathcal{M}_i$
\mathcal{S}	Set of visits, $\mathcal{S} = \mathcal{S}^P \cup \mathcal{S}^C$
\mathcal{S}_v	Set of feasible visits for ship v , $\mathcal{S}_v = \mathcal{S} \cup \{d(v)\}$
\mathcal{S}_v^F	Set of visits to flow receivers for ship v , $\mathcal{S}_v^F = \mathcal{S}^C \cup \{d(v)\}$
\mathcal{P}	Set of products
\mathcal{D}	Set of days within the planning horizon

C.1.2 Indices

i, j	Locations
k	Flow receivers
$o(v)$	Start node of ship v
$d(v)$	Dummy end node of ship v
m, n, o	Visit numbers
v	Ships
p	Products
d	Days

C.1.3 Parameters

B	Penalty cost for stock levels below the safety stock level	[NOK/hour]
C_{ij}	Transportation cost for sailing from location i to location j	[NOK]
E_p	Unit cost of buying product p externally	[NOK/ton]
E_i^T	Transportation cost for external feed delivery to fish farm i	[NOK]
L_{vp}^0	Initial load of product p on ship v	[tons]
K_v^{MAX}	Maximum capacity for ship v	[tons]
Q_{ip}^{MIN}	Minimum unloading quantity of product p for fish farm i	[tons]
S_i^0	Initial inventory level at factory i	[tons]
S_{ip}^0	Initial inventory level of product p at fish farm i	[tons]
S_{ip}^{MIN}	Minimum inventory level of product p at fish farm i	[tons]
S_i^{MAX}	Maximum inventory level at location i	[tons]
A_i	Reduction in storage capacity at fish farm i outside working hours	[%]
T^{MAX}	Length of planning period	[hours]
T_{ij}^S	Sailing time from location i to location j	[hours]
T_i^L	Loading or unloading time per ton of feed for location i	[hours/ton]
T_d^{WS}	Start of service hours for day d	[hours]
T_d^{WE}	End of service hours for day d	[hours]
J_i	Location type for location i , 1 for factories and -1 for fish farms	
R_{ip}	Consumption rate of product p at fish farm i	[tons/hour]
P_i	Production rate at factory i	[tons/hour]
H_i	Time between visits to location i	[hours]

C.1.4 Decision variables

x_{imjnv}	1 if ship v sails from visit (i, m) to visit (j, n) , else 0	
w_{imv}	1 if ship v makes visit (i, m) , else 0	
y_{im}	1 if visit (i, m) is not made by any ship, else 0	
u_i	1 if fish farm i is supplied internally, 0 if supplied externally	
q_{imvp}	Amount of product p loaded/unloaded by ship v during visit (i, m)	[tons]
$l_{imjnkovp}$	Amount of product p on board ship v on arc (i, m, j, n) , destined for visit (k, o)	[tons]
s_{im}	Amount of feed in stock at the start of visit $(i, m) \in \mathcal{S}^P$	[tons]
s_{im}^E	Amount of feed in stock at the end of visit $(i, m) \in \mathcal{S}^P$	[tons]
s_{imp}	Amount of product p in stock at the start of visit $(i, m) \in \mathcal{S}^C$	[tons]
s_{imp}^E	Amount of product p in stock at the end of visit $(i, m) \in \mathcal{S}^C$	[tons]
d_{imp}	Amount of product p below S_{ip}^{MIN} at the start of visit $(i, m) \in \mathcal{S}^C$	[tons]
t_{im}	Time for start of service for visit (i, m)	[hours]
t_{im}^E	Time for end of service for visit (i, m) ,	[hours]
σ_{imd}	1 if visit $(i, m) \in \mathcal{S}^C$ is within service hours on day d , else 0	

C.2 Model Formulation

C.2.1 Objective function

$$\begin{aligned} \min z = & \sum_{(i,m) \in \mathcal{S}} \sum_{(j,n) \in \mathcal{S}} \sum_{v \in \mathcal{V}} C_{ij} x_{imjnv} + T^{MAX} \sum_{i \in \mathcal{N}^C} \sum_{p \in \mathcal{P}} E_p R_{ip} (1 - u_i) \\ & + \sum_{i \in \mathcal{N}^C} E_i^T (1 - u_i) + \sum_{(i,m) \in \mathcal{S}^C} \sum_{p \in \mathcal{P}} B \frac{d_{imp}}{R_{ip}} \end{aligned} \quad (\text{C.1})$$

C.2.2 Routing constraints

$$\sum_{(j,n) \in \mathcal{S}_v} x_{o(v)jnv} = 1 \quad v \in \mathcal{V} \quad (\text{C.2})$$

$$\sum_{(j,n) \in \mathcal{S}} x_{jnimv} - \sum_{(j,n) \in \mathcal{S}_v} x_{imjnv} = 0 \quad (i, m) \in \mathcal{S} \setminus \{o(v)\}, v \in \mathcal{V} \quad (\text{C.3})$$

$$\sum_{(j,n) \in \mathcal{S}} x_{jnd(v)v} = 1 \quad v \in \mathcal{V} \quad (\text{C.4})$$

$$\sum_{(j,n) \in \mathcal{S}_v} x_{imjnv} = w_{imv} \quad (i, m) \in \mathcal{S}, v \in \mathcal{V} \quad (\text{C.5})$$

$$\sum_{(j,n) \in \mathcal{S}_v} \sum_{v \in \mathcal{V}} x_{imjnv} = 1 - y_{im} \quad (i, m) \in \mathcal{S} \quad (\text{C.6})$$

$$y_{im} - y_{im-1} \geq 0 \quad (i, m) \in \mathcal{S} | m > 1 \quad (\text{C.7})$$

$$u_i = 1 - y_{i1} \quad i \in \mathcal{N}^C \quad (\text{C.8})$$

C.2.3 Loading and unloading constraints

$$L_{vp}^0 + J_i q_{o(v)vp} = \sum_{(j,n) \in \mathcal{S}_v} \sum_{(k,o) \in \mathcal{S}_v^F} l_{o(v)jnkovp} \quad v \in \mathcal{V}, p \in \mathcal{P} \quad (\text{C.9})$$

$$\sum_{(j,n) \in \mathcal{S}} \sum_{(k,o) \in \mathcal{S}_v^F} l_{jnimkovp} + J_i q_{imvp} - \sum_{(j,n) \in \mathcal{S}_v} \sum_{(k,o) \in \mathcal{S}_v^F} l_{imjnkovp} = 0 \quad (\text{C.10})$$

$$(i, m) \in \mathcal{S} \setminus \{o(v)\}, v \in \mathcal{V}, p \in \mathcal{P}$$

$$\sum_{(j,n) \in \mathcal{S}} l_{jnimivp} - q_{imvp} = 0 \quad (i, m) \in \mathcal{S}^C, v \in \mathcal{V}, p \in \mathcal{P} \quad (\text{C.11})$$

$$\sum_{(j,n) \in \mathcal{S}_v} l_{imjnkovp} - \sum_{(j,n) \in \mathcal{S}} l_{jnimkovp} = 0 \quad (\text{C.12})$$

$$(i, m) \in \mathcal{S}^C, (k, o) \in \mathcal{S}_v^F, v \in \mathcal{V}, p \in \mathcal{P}$$

$$\sum_{(j,n) \in \mathcal{S}_v} l_{imjnkovp} \leq \sum_{(j,n) \in \mathcal{S}} l_{jnimkovp} + q_{imvp} \quad (\text{C.13})$$

$$(j, n) \in \mathcal{S}^P, (k, o) \in \mathcal{S}_v^F, v \in \mathcal{V}, p \in \mathcal{P}$$

$$\sum_{(j,n) \in \mathcal{S}} l_{jnimkovp} \leq \sum_{(j,n) \in \mathcal{S}_v} l_{imjnkovp} \quad (\text{C.14})$$

$$(i, m) \in \mathcal{S}^P, (k, o) \in \mathcal{S}_v^F, v \in \mathcal{V}, p \in \mathcal{P}$$

$$\sum_{(k,o) \in \mathcal{S}_v^F} \sum_{p \in \mathcal{P}} l_{imjnkovp} \leq K_v^{MAX} x_{imjnv} \quad (i, m) \in \mathcal{S}, (j, n) \in \mathcal{S}_v, v \in \mathcal{V} \quad (\text{C.15})$$

$$\sum_{p \in \mathcal{P}} l_{imjnkovp} \leq S_k^{MAX} x_{imjnv} \quad (i, m) \in \mathcal{S}, (j, n) \in \mathcal{S}_v, (k, o) \in \mathcal{S}_v^F, v \in \mathcal{V} \quad (\text{C.16})$$

$$\sum_{p \in \mathcal{P}} l_{imjnkovp} \leq S_k^{MAX} w_{kov} \quad (i, m) \in \mathcal{S}, (j, n) \in \mathcal{S}_v, (k, o) \in \mathcal{S}^C, v \in \mathcal{V} \quad (\text{C.17})$$

$$Q_{ip}^{MIN} w_{imv} \leq q_{imvp} \quad (i, m) \in \mathcal{S}^C, v \in \mathcal{V}, p \in \mathcal{P} \quad (\text{C.18})$$

$$\sum_{p \in \mathcal{P}} q_{imvp} \leq s_{im} + S_i^{MAX} (1 - w_{imv}) \quad (i, m) \in \mathcal{S}^P, v \in \mathcal{V} \quad (\text{C.19})$$

$$\sum_{p \in \mathcal{P}} q_{imvp} \geq s_{im} - S_i^{MAX} (1 - w_{imv}) \quad (i, m) \in \mathcal{S}^P, v \in \mathcal{V} \quad (\text{C.20})$$

C.2.4 Inventory constraints

$$S_i^0 + P_i t_{im} = s_{im} \quad (i, m) \in \mathcal{S}^P | m = 1 \quad (\text{C.21})$$

$$S_{ip}^0 - R_{ip} t_{im} = s_{imp} \quad (i, m) \in \mathcal{S}^C | m = 1, p \in \mathcal{P} \quad (\text{C.22})$$

$$s_{im} + P_i (t_{im}^E - t_{im}) - \sum_{v \in \mathcal{V}} \sum_{p \in \mathcal{P}} q_{imvp} = s_{im}^E \quad (i, m) \in \mathcal{S}^P \quad (\text{C.23})$$

$$s_{imp} - R_{ip} (t_{im}^E - t_{im}) + \sum_{v \in \mathcal{V}} q_{imvp} = s_{imp}^E \quad (i, m) \in \mathcal{S}^C, p \in \mathcal{P} \quad (\text{C.24})$$

$$s_{i(m-1)}^E + P_i (t_{im} - t_{i(m-1)}^E) = s_{im} \quad (i, m) \in \mathcal{S}^P | m > 1 \quad (\text{C.25})$$

$$s_{i(m-1)p}^E - R_{ip} (t_{im} - t_{i(m-1)}^E) = s_{imp} \quad (i, m) \in \mathcal{S}^C | m > 1, p \in \mathcal{P} \quad (\text{C.26})$$

$$s_{im} \leq S_i^{MAX} \quad (i, m) \in \mathcal{S}^P \quad (C.27)$$

$$\sum_{p \in \mathcal{P}} s_{imp}^E \leq (1 - A_i) S_i^{MAX} + A_i S_i^{MAX} \sum_{d \in \mathcal{D}} \sigma_{imd} \quad (i, m) \in \mathcal{S}^C \quad (C.28)$$

$$S_i^{MIN} u_i \leq s_{imp} + d_{imp} \quad (i, m) \in \mathcal{S}^C, p \in \mathcal{P} \quad (C.29)$$

$$d_{imp} \leq S_{ip}^{MIN} u_i \quad (i, m) \in \mathcal{S}^C, p \in \mathcal{P} \quad (C.30)$$

$$s_{im}^E + P_i(T^{MAX} - t_{im}^E) \leq S_i^{MAX} \quad (i, m) \in \mathcal{S}^P | m = |\mathcal{M}_i| \quad (C.31)$$

$$S_{ip}^{MIN} u_i + R_{ip} T^{MAX} u_i - R_{ip} t_{im}^E \leq s_{imp}^E \quad (i, m) \in \mathcal{S}^C | m = |\mathcal{M}_i|, p \in \mathcal{P} \quad (C.32)$$

C.2.5 Timing constraints

$$t_{im} + \sum_{v \in \mathcal{V}} \sum_{p \in \mathcal{P}} T_i^L q_{imvp} = t_{im}^E \quad (i, m) \in \mathcal{S} \quad (C.33)$$

$$\sum_{v \in \mathcal{V}} x_{imjnv} (t_{im}^E + T_{ij}^S - t_{jn}) \leq 0 \quad (i, m), (j, n) \in \mathcal{S} \quad (C.34)$$

$$t_{im}^E \leq T^{MAX} \quad (i, m) \in \mathcal{S} \quad (C.35)$$

$$t_{im} \leq T^{MAX} u_i \quad (i, m) \in \mathcal{S}^C \quad (C.36)$$

$$y_{im} (t_{im} - t_{i(m-1)}^E) = 0 \quad (i, m) \in \mathcal{S} | m > 1 \quad (C.37)$$

$$T_d^{WS} - T^{MAX} (1 - \sigma_{imd} + y_{im}) \leq t_{im} \leq T_d^{WE} + T^{MAX} (1 - \sigma_{imd} + y_{im}) \quad (i, m) \in \mathcal{S}^C, d \in \mathcal{D} \quad (C.38)$$

$$\sum_{d \in \mathcal{D}} \sigma_{imd} \leq 1 \quad (i, m) \in \mathcal{S}^C \quad (C.39)$$

$$\sum_{d \in \mathcal{D}} \sigma_{imd} \geq y_{im} \quad (i, m) \in \mathcal{S}^C \quad (\text{C.40})$$

$$t_{im} + H_i(1 - y_{i(m+1)}) \leq t_{im+1} \quad (i, m) \in \mathcal{S} \quad (\text{C.41})$$

C.2.6 Variable constraints

$$x_{imjnv} \in \{0, 1\} \quad (i, m) \in \mathcal{S}, (j, n) \in \mathcal{S}_v, v \in \mathcal{V} \quad (\text{C.42})$$

$$w_{imv} \in \{0, 1\} \quad (i, m) \in \mathcal{S}, v \in \mathcal{V} \quad (\text{C.43})$$

$$y_{im} \in \{0, 1\} \quad (i, m) \in \mathcal{S} \quad (\text{C.44})$$

$$u_i \in \{0, 1\} \quad i \in \mathcal{N}^C \quad (\text{C.45})$$

$$\sigma_{imd} \in \{0, 1\} \quad (i, m) \in \mathcal{S}^C, d \in \mathcal{D} \quad (\text{C.46})$$

$$q_{imvp} \geq 0 \quad (i, m) \in \mathcal{S}, v \in \mathcal{V}, p \in \mathcal{P} \quad (\text{C.47})$$

$$l_{imjnkoup} \geq 0 \quad (i, m) \in \mathcal{S}, (j, n) \in \mathcal{S}_v, (k, o) \in \mathcal{S}_v^F, v \in \mathcal{V}, p \in \mathcal{P} \quad (\text{C.48})$$

$$s_{im} \geq 0 \quad (i, m) \in \mathcal{S}^P \quad (\text{C.49})$$

$$s_{im}^E \geq 0 \quad (i, m) \in \mathcal{S}^P \quad (\text{C.50})$$

$$s_{imp} \geq 0 \quad (i, m) \in \mathcal{S}^C, p \in \mathcal{P} \quad (\text{C.51})$$

$$s_{imp}^E \geq 0 \quad (i, m) \in \mathcal{S}^C, p \in \mathcal{P} \quad (\text{C.52})$$

$$d_{imp} \geq 0 \quad (i, m) \in \mathcal{S}^C, p \in \mathcal{P} \quad (\text{C.53})$$

$$t_{im} \geq 0 \quad (i, m) \in \mathcal{S} \quad (\text{C.54})$$

$$t_{im}^E \geq 0 \quad (i, m) \in \mathcal{S} \quad (\text{C.55})$$

Appendix D

Test Case Data

This appendix gives an overview of common problem data and data related to the factory, ships and fish farms for the various-sized test cases. In order to maintain data confidentiality, the names of fish farms are removed and they are listed in a random order.

D.1 Common Data

Number of ships:	2
Number of days:	10
Ship speed, knots:	13
Unloading rate, tons per hour:	180
Loading rate, tons per hour:	270
Transportation cost per hour, NOK:	1600
External transportation cost, NOK :	4400
External feed cost per ton, NOK:	250
Penalty cost per hour, NOK :	400

D.2 Small Test Case

The production rate, storage capacity and initial stock for the factory are given in Table D.1. The ship capacity and initial load are given in Table D.2.

Factory	
Production rate	15 tons/hour
Storage capacity	1000 tons
Initial stock	500 tons

Table D.1: Production rate, storage capacity and initial stock for the factory.

Ships	
Storage capacity	1000 tons
Initial load	0 tons

Table D.2: Capacity and initial load for each ship.

Table D.3 lists initial stock levels, weekly demands, safety stock levels and storage capacities for all fish farms. With a partial split, the first ship serves 12 fish farms, while the other serves 13, and 5 fish farms are shared. With a full split, the ships supply 10 fish farms each.

Fish farm	Initial stock	Weekly demand	Safety stock	Storage capacity
Location 1	167	194	28	450
Location 2	81	95	14	230
Location 3	109	127	18	400
Location 4	179	209	30	228
Location 5	111	130	19	300
Location 6	113	132	19	230
Location 7	101	118	17	200
Location 8	225	26	38	400
Location 9	26	30	4	390
Location 10	124	144	21	372
Location 11	29	34	5	300
Location 12	110	128	18	400
Location 13	89	104	15	120
Location 14	81	94	13	250
Location 15	113	131	19	400
Location 16	135	157	22	250
Location 17	88	102	15	400
Location 18	150	175	25	400
Location 19	120	140	20	240
Location 20	190	222	32	400

Table D.3: Initial stock, demand, safety stock and maximum storage capacity for each fish farm. All numbers are given in tons.

D.3 Medium Test Case

The production rate, storage capacity and initial stock for the factory are given in Table D.4. The ship capacity and initial load are given in Table D.5.

Factory	
Production rate	30 tons/hour
Storage capacity	2000 tons
Initial stock	1000 tons

Table D.4: Production rate, storage capacity and initial stock for the factory

Ships	
Storage capacity	2000 tons
Initial load	0 tons

Table D.5: Capacity and initial load for each ship.

Table D.6 lists initial stock levels, weekly demands, safety stock levels and storage capacities for all fish farms. With a partial split, the first ship serves 24 fish farms, while the other serves 27 and 11 fish farms are shared. With a full split, one ship supplies 17 fish farms and the other 23.

Fish farm	Initial stock	Weekly demand	Safety stock	Storage capacity
Location 1	167	194	28	450
Location 2	94	109	16	400
Location 3	81	95	14	230
Location 4	221	258	37	390
Location 5	109	127	18	400
Location 6	193	225	32	400
Location 7	179	209	30	228
Location 8	146	170	24	260
Location 9	111	130	19	300
Location 10	63	73	10	400
Location 11	113	132	19	230
Location 12	116	135	19	400
Location 13	101	118	17	200
Location 14	105	123	18	200
Location 15	225	263	38	400
Location 16	250	292	42	300
Location 17	26	30	4	390
Location 18	124	144	21	372
Location 19	111	129	18	270
Location 20	49	57	8	130
Location 21	106	124	18	200
Location 22	29	34	5	300
Location 23	110	128	18	400
Location 24	132	154	22	180
Location 25	89	104	15	120
Location 26	147	172	25	400
Location 27	9	11	2	400
Location 28	81	94	13	250
Location 29	59	69	10	200
Location 30	113	131	19	400
Location 31	135	157	22	250
Location 32	91	107	15	200
Location 33	152	177	25	390

Fish farm	Initial stock	Weekly demand	Safety stock	Storage capacity
Location 34	88	102	15	400
Location 35	255	298	43	400
Location 36	150	175	25	400
Location 37	120	140	20	240
Location 38	66	77	11	180
Location 39	144	168	24	400
Location 40	190	222	32	400

Table D.6: Initial stock, demand, safety stock and maximum storage capacity for each fish farm. All numbers are given in tons.

D.4 Large Test Case

The production rate, storage capacity and initial stock for the factory are given in Table D.7. The ship capacity and initial load are given in Table D.8.

Factory	
Production rate	45 tons/hour
Storage capacity	3000 tons
Initial stock	1500 tons

Table D.7: Production rate, storage capacity and initial stock for the factory.

Ships	
Storage capacity	3000 tons
Initial load	0 tons

Table D.8: Capacity and initial load for each ship.

Table D.9 lists initial stock levels, weekly demands, safety stock levels and storage capacities for all fish farms. With a partial split, the first ship serves 36 fish farms, while the other serves 41 and 17 fish farms are shared. With a full split, one ship supplies 24 fish farms and the other 36.

Fish farm	Initial stock	Weekly demand	Safety stock	Storage capacity
Location 1	167	194	28	450
Location 2	4	5	1	280
Location 3	94	109	16	400
Location 4	81	95	14	230
Location 5	78	91	13	390
Location 6	29	34	5	300
Location 7	221	258	37	390
Location 8	109	127	18	400
Location 9	193	225	32	400
Location 10	179	209	30	228
Location 11	154	179	26	400
Location 12	146	170	24	260
Location 13	76	89	13	250
Location 14	111	130	19	300
Location 15	63	73	10	400
Location 16	27	31	4	100
Location 17	113	132	19	230
Location 18	116	135	19	400
Location 19	101	118	17	200
Location 20	105	123	18	200
Location 21	225	263	38	400
Location 22	250	292	42	300
Location 23	29	33	5	300
Location 24	26	30	4	390
Location 25	124	144	21	372
Location 26	123	144	21	300
Location 27	106	124	18	280
Location 28	111	129	18	270
Location 29	54	63	9	200
Location 30	49	57	8	130
Location 31	107	124	18	210
Location 32	106	124	18	200
Location 33	29	34	5	300
Location 34	110	128	18	400
Location 35	170	198	28	420
Location 36	132	154	22	180
Location 37	89	104	15	120
Location 38	118	138	20	300
Location 39	147	172	25	400
Location 40	67	78	11	200
Location 41	9	11	2	400
Location 42	10	11	2	300
Location 43	81	94	13	250
Location 44	188	219	31	400

Fish farm	Initial stock	Weekly demand	Safety stock	Storage capacity
Location 45	59	69	10	200
Location 46	158	184	26	400
Location 47	113	131	19	400
Location 48	135	157	22	250
Location 49	91	107	15	200
Location 50	152	177	25	390
Location 51	88	102	15	400
Location 52	82	96	14	220
Location 53	255	298	43	400
Location 54	12	14	2	400
Location 55	150	175	25	400
Location 56	120	140	20	240
Location 57	66	77	11	180
Location 58	144	168	24	400
Location 59	190	222	32	400
Location 60	80	93	13	400

Table D.9: Initial stock, demand, safety stock and maximum storage capacity for each fish farm. All numbers are given in tons.

Appendix E

Cost Calculations

This appendix gives an overview of our calculations of the internal transportation cost and external feed costs.

E.1 Transportation Cost

E.1.1 LNG consumption

In order to find the LNG consumption per hour for the ships, we used data provided by the shipping company building the ships, Egil Ulvan Rederi. We received Table E.2 which gives predicted delivered power, P_D for different speeds with a draft of six meters and a controllable pitch propeller. We also received Table E.1, showing energy consumption for different values of delivered power. With a speed of 13 knots, we get a predicted delivered power of 1,669 kW and a consumption of approximately 14,000 MJ per hour.

	Resistance		Propulsion, Trial Condition (no S. M.), Headwind 0 Bft					
Speed [kts]	R_T [kN]	P_E [kW]	η_0 [-]	η_d [-]	P_D [kW]	P_B [kW]	n [RPM]	P/D [-]
9	68.2	316	0.603	0.674	482	497	103.8	0.667
11	106.4	602	0.607	0.677	916	943	125.7	0.689
12	131.7	813	0.605	0.675	1,241	1,278	138.1	0.693
13	163.3	1,092	0.604	0.674	1,669	1,719	148.8	0.715
15	282.4	2,179	0.575	0.641	3,501	3,606	154.5	0.956

Table E.1: Delivered power and energy consumption for a draft of six meters and controllable pitch propeller (Egil Ulvan Rederi, 2014).

Controllable pitch propeller	%	17%	28%	43%	63%	100%
Delivered power	kW	406	683	1,049	1,539	2,438
Revolutions per minute	RPM	600	700	800	900	1,000
Specific fuel consumption	kJ/kWh	10,922	9,272	8,759	8,439	8,056
Fuel consumption	MJ/h	4,434	6,333	9,188	12,988	19,641
	kW	1,232	1,759	2,552	3,608	5,456

Table E.2: Consumption with controllable pitch propeller (Egil Ulvan Rederi, 2014).

E.1.2 LNG prices

Prices of LNG are volatile, but after discussing with Océane Balland from DNV GL Maritime Advisory, we decided on an estimated price of 120 NOK per million BTU. 14,000 MJ converted to million BTU is 13.26 million BTU. This gives a transportation cost per hour of 1,600 NOK.

Consumption per hour	13.26 million BTU
LNG price per million BTU	120 NOK
Transportation cost per hour	1,600 NOK

Table E.3: Transportation costs, equal to LNG fuel costs.

E.2 External Feed Costs

E.2.1 Fixed transportation cost

Marine Harvest's main external supplier, Skretting, has feed factories located in both region North, West and South. Therefore, the transportation cost related to external feed delivery is set equal for all fish farms. We assume that the supplier has a transportation cost similar to our internal transportation cost and that they deliver to many fish farms and therefore do not travel from a factory to each fish farm separately. Therefore, we have simply calculated the average LNG cost of traveling to fish farms from the internal factory, and set the external transportation cost to be 25% of this. This gives a fixed transportation cost for external delivery of 4,400 NOK.

E.2.2 Unit cost

The unit cost for external feed is calculated using Marine Harvest's current cost of standard feed, around 8,000 NOK per ton, and a profit margin of 3%. This gives a unit cost for external feed of 250 NOK per ton.

Appendix F

Parallel Frameworks

This appendix contains the two parallel frameworks, written in C++. Section F.1 gives the node-based framework, while the tree-based framework can be found in Section F.2.

F.1 Node-Based Parallelization

F.1.1 *main*

```
#include <cstdlib>
#include <cstdio>
#include "bbclass.hpp"
#include "workerclass.hpp"

using namespace std;

int main(int argc, char** argv) {

    //Start the MPI environment
    MPI::Init();
    MPI::Intracomm communicator;

    communicator = MPI::COMM_WORLD;

    MPI::Request request;
    MPI::Status status;

    // Start the solver
    XPRSinit(NULL);

    // If I am the master, I create a bbclass object
    if (communicator.Get_rank() == 0){
        //Create a bbclass object
        bbclass instance;
```

```

        instance.initialize(argv[1], communicator);
        //Call the bbclass routine to solve
        instance.optimize();

    // Else, I am a worker,
    } else {
        // Create a workerclass object
        workerclass instance;
        instance.initialize(argv[1], communicator);
        // Call the worker routing to solve
        instance.optimize();
    }

    // End the solver
    XPRFree();

    // End the MPI environment
    MPI::Finalize();

    return 0;
}

```

F.1.2 *bbclass*

Header file

```

#ifndef BBCLASS_HPP
#define BBCLASS_HPP

#include "nodeclass.hpp"

#include "mpi.h"
#include "xprs.h"

#include <stddef.h>
#include <cstdio>

struct nodestruct {
    //the struct stores one node, and pointers to the next
    //and previous node in the list
    nodeclass node;
    nodestruct *next;
    nodestruct *previous;
};

class bbclass {
public:
    bbclass();
    bbclass(const bbclass& orig);
    virtual ~bbclass();

    // initialize the btree with all binary variables relaxed
    void initialize(char *, MPI::Intracomm);
    //the optimize function called from main
    int optimize();

```

```

//this will take one node and branch it, creating two child nodes
void branch_node(nodestruct *, int);
//prune if a node's bound/solution is infeasible or worse than the incumbent.
void prune_node(nodestruct *);
//send a notification to a worker, 1 means solve, 0 means terminate;
void send_notification(int, int);
//send the status of the binaries of a node to a worker to solve;
void send_binaries(int *, int);
// receive the solution
void receive_solution(int);

private:
MPI::Intracomm communicator;
MPI::Request request;
MPI::Status status;

//the beginning of the linked list
nodestruct *first_node;
// number of processors available
int numProc;
//rank of a processor
int myRank;
//number of nodes in the list
int nodes_open;
//the incumbent node, if any, found so far.
nodestruct *incumbent_node;
// Number of binary variables in the problem
int num_bins;
// Status of the solution from each worker
int *solution_status;
// Value of the solution from each worker
double *solution_value;
// Branch variable from each worker
int *branch_variable;
// current lower bound of the BB tree
double lower_bound;
// Time when algorithm starts
double initial_time;
};

#endif /* BBCLASS_HPP */

```

Source file

```

#include "bbclass.hpp"
#include "nodeclass.hpp"
#include <cstdio>

using namespace std;

bbclass::bbclass() {
}

bbclass::bbclass(const bbclass& orig) {
}

bbclass::~bbclass() {
}

```

```

}

int bbclass::optimize() {

    numProc=communicator.Get_size();
    initial_time = MPI::Wtime();

    int nodes_sent;
    int sendProc;
    int count_received;

    double sendTime;
    double receiveTime;

    nodelist *current = new nodelist[1];

    solution_status = new int[numProc];
    solution_value = new double[numProc];
    branch_variable = new int[numProc];

    nodelist **node_to_processor = new nodelist*[numProc];

    int it_counter = 0;

    while (nodes_open > 0 and (MPI::Wtime() initial_time)<36000) {
        nodes_sent = 0;

        current = first_node;

        // Counter for sending problems to workers
        sendProc=numProc-1;

        // Create array to save which processor receives which nodelist

        while (sendProc>0 and nodes_open>0) {
            sendTime = MPI::Wtime();
            // If a node turns out to have a lower bound higher than the
            //incumbent value, prune this
            if (incumbent_node != NULL && current >node.get_lowerbound() >
                incumbent_node >node.get_value()) {
                nodelist *node_to_prune = current;
                current = current->next;
                prune_node(node_to_prune);
                nodes_open--;
            }
            else {

                // Save which processor receives which nodelist
                node_to_processor[sendProc] = current;

                // Send notification
                send_notification(1, sendProc);

                int *fixed_vars = new int[num_bins];
                current >node.get_binaries(fixed_vars, num_bins);

                //send the current nodelist to a processor

```



```

send_binaries(fixed_vars , sendProc);

delete [] fixed_vars;

//update processor_count
sendProc ;
nodes_open ;
nodes_sent++;

// Set current node to the next node in the list
if (current > next != NULL) {
    current = current > next;
}
else {
    break;
}
}
}

//receive solutions from workers
count_received = 0;

while (count_received < nodes_sent) {
    // Wait until something is received
    while (communicator.Iprobe(MPI::ANY_SOURCE, MPI::ANY_TAG, status) == 0);
    if (count_received == 0) {
        receiveTime = MPI::Wtime();
    }
    // Save identity of origin worker
    int origin_worker = status.Get_source();

    // Receive the solution status and value if any
    receive_solution(origin_worker);

    //Prune or branch according to solution_status received
    if (solution_status[origin_worker]==0) {
        prune_node(node_to_processor[origin_worker]);
    }
    else if (solution_status[origin_worker] == 2 ) {
        int *fixed = new int[num_bins];
        node_to_processor[origin_worker] > node.get_binaries(fixed , num_bins)

        if (incumbent_node == NULL) {
            incumbent_node = node_to_processor[origin_worker];
            incumbent_node > node.set_value(solution_value[origin_worker]);
            prune_node(node_to_processor[origin_worker]);
        }
        else if (solution_value[origin_worker]<
            incumbent_node > node.get_value() ) {
            incumbent_node = node_to_processor[origin_worker];
            incumbent_node > node.set_value(solution_value[origin_worker]);
            prune_node(node_to_processor[origin_worker]);
        }
        else {
            prune_node(node_to_processor[origin_worker]);
        }
    }
}
}

```

```

else if (incumbent_node != NULL
and solution_value[origin_worker]>=
incumbent_node > node.get_value()) {
    prune_node(node_to_processor[origin_worker]);
}
else if (solution_status[origin_worker] == 3 ) {
    return 0;
}
else {
    // Set the node value to be the solution value received
    node_to_processor[origin_worker]
    > node.set_value(solution_value[origin_worker]);
    branch_node(node_to_processor[origin_worker], origin_worker);
}
count_received++;
}
// Set lower bound equal to the first node's lower bound
current = first_node;

if (incumbent_node != NULL) {
    lower_bound = incumbent_node > node.get_value();
}

else {
    lower_bound = current > node.get_lowerbound();
}

// Check all open nodes if they have a lower bound below this
for(int i=0;i<nodes_open;i++) {
    if (current == NULL) {
        break;
    }
    else{
        if (current > node.get_lowerbound() < lower_bound) {
            lower_bound = current > node.get_lowerbound();
        }
        current=current > next;
    }
}
// If current lower_bound is equal to or larger than the incumbent value,
//we have found the optimal solution
if (incumbent_node != NULL and lower_bound >= incumbent_node > node.get_value
    break;
}

it_counter++;
}
delete [] node_to_processor;
delete [] solution_status;
delete [] solution_value;
delete [] branch_variable;

if (incumbent_node != NULL) {
    int *write_binaries = new int(num_bins);
    incumbent_node > node.get_binaries(write_binaries, num_bins);
}

```

```

    //Once the list is empty, select the optimal solution
    //and report 0 to workers (terminate)
    for(int i=1;i<numProc;i++) {
        send_notification(0,i);
    }
    return 0;
}

void bbclass::initialize(char *argv, MPI::Intracomm comm){
    communicator = comm;
    myRank = communicator.Get_rank();

    //declare an xpress problem object
    XPRSprob prob;
    //allocate memory using the dedicated function
    XPRScreateprob(&prob);
    //read the problem from the file path provided since main
    XPRSreadprob(prob, argv, "");
    //read the number of binary variables in the problem
    int nCols = 0;
    XPRSgetintattrib(prob, XPRS_COLS, &nCols);

    char *types = new char[nCols];
    num_bins=0;
    XPRSgetcoltype(prob, types, 0, nCols-1);
    for(int i=0;i<nCols;i++) if (types[i]=='B') num_bins++;

    //start the node list by pointing the first_node pointer to a new node with
    //all it's variables relaxed.

    int *fix_relax = new int[num_bins];

    //fill the "all relaxed" array
    for(int i=0;i<num_bins;i++) fix_relax[i] = 1;

    //create the first nodestruct
    nodestruct *list_node = new nodestruct[1];

    //call the initialization method for the node member of the struct
    list_node->node.initialize(fix_relax, num_bins, 0);

    //for safety, point the next member of the struct to where it can't hurt
    list_node->next = NULL;

    //for safety, point the next member of the struct to where it can't hurt
    list_node->previous = NULL;

    //make the "first_node" field of the class point to this new node
    first_node = list_node;
    nodes_open=1;

    incumbent_node = NULL;

    delete [] types;
    delete [] fix_relax;
    XPRSdestroyprob(prob);
}

```

```

// Function to send notification to worker, 0 means terminate, 1 means solve
void bbclass::send_notification(int notification , int proc) {
    request = communicator.Issend(&notification , 1, MPI::INT, proc , 0);
    request.Wait(status);
}

// Function to send the status of binary variables to a worker
void bbclass::send_binaries(int* binaries , int proc) {
    request = communicator.Issend(binaries , num_bins , MPI::INT, proc , 1);
    request.Wait(status);
}

// Function to receive solution_status from a worker, 0 means infeasible solution
//1 means LP feasible solution and 2 means integer feasible solution
void bbclass::receive_solution(int origin_worker) {

    communicator.Recv(&solution_status[origin_worker] , 1,
MPI::INT, origin_worker , 2, status);

    // If the solution is feasible, the value is received as well
    if (solution_status[origin_worker]==1) {
        communicator.Recv(&solution_value[origin_worker] , 1,
MPI::DOUBLE, origin_worker , 3, status);
        communicator.Recv(&branch_variable[origin_worker] , 1,
MPI::INT, origin_worker , 4, status);
    }

    else if (solution_status[origin_worker]==2) {
        communicator.Recv(&solution_value[origin_worker] , 1,
MPI::DOUBLE, origin_worker , 3, status);
    }
}

void bbclass::branch_node(nodestruct *node_to_branch , int origin_worker) {

    double parent_value = node_to_branch >node.get_value();
    int parent_tag = node_to_branch >node.get_tag();

    // Create nodes for children
    nodeclass child_down , child_up;

    // Create binary arrays for both children
    int *binaries_down = new int [num_bins];
    int *binaries_up = new int [num_bins];

    node_to_branch >node.get_binaries(binaries_down , num_bins);
    node_to_branch >node.get_binaries(binaries_up , num_bins);

    // Choose a variable to branch on
    binaries_down[branch_variable[origin_worker]]=0;
    binaries_up[branch_variable[origin_worker]]=1;

    child_down.initialize(binaries_down , num_bins , parent_value);
    child_up.initialize(binaries_up , num_bins , parent_value);

    delete [] binaries_down;
}

```

```

delete [] binaries_up;

// Create nodestruct for the up child
nodestruct* childstruct_up = new nodestruct[1];

// Set previous node of the up child to be the down child and the next to be
//the next of the branched node
childstruct_up >node = child_up;

if (node_to_branch >next != NULL) {
    childstruct_up >next = node_to_branch >next;
    node_to_branch >next >previous = childstruct_up;
}
else {
    childstruct_up >next = NULL;
}
childstruct_up >previous = node_to_branch;

//Replace branched node with down child and set next node to be the up child
node_to_branch >node = child_down;
node_to_branch >next = childstruct_up;

nodes_open++;
nodes_open++;
}

void bbclass::prune_node(nodestruct *node_to_prune) {
//There is just one in the list
if (node_to_prune >previous == NULL and node_to_prune >next == NULL) {
}
// If the node to prune is first in the list, set the first_node to be the next
else if (node_to_prune >previous == NULL ) {
    first_node = node_to_prune >next;
    node_to_prune >next = NULL;
    first_node >previous = NULL;
}
// If the node is last of the list, set the previous node's next to NULL
else if (node_to_prune >next == NULL) {
    node_to_prune >previous >next = NULL;
    node_to_prune >previous = NULL;
}
// Else, the node to prune is in the middle of the list, remove it
else {
    // Set parent node's next node to the pruned node's next node
    node_to_prune >previous >next = node_to_prune >next;

    // Set next node of pruned node to null
    node_to_prune >next >previous = node_to_prune >previous;

    node_to_prune >previous = NULL;
    node_to_prune >next= NULL;

    if (incumbent_node != node_to_prune) {
delete [] node_to_prune;
    }
}
}
}

```

F.1.3 *workerclass*

Header file

```
#ifndef WORKERCLASS_HPP
#define WORKERCLASS_HPP

#include "xprs.h"
#include "mpi.h"

#include <cstdlib>

class workerclass {
public:
    workerclass();
    workerclass(const workerclass& orig);
    virtual ~workerclass();

    //initializes the problem reading the file
    void initialize(char *, MPI::Intracomm);

    //calls xpress to optimize the problem, function run from main
    int optimize();

    //receives a message from the master, either to solve (1) or terminate (0)
    void receive_notification();
    //receives status of binary variables from the master
    void receive_binaries();
    //returns the worker solution to the master
    void return_solution();

private:
    XPRSProb prob;    //this will store the Xpress object which will be solved

    MPI::Intracomm communicator;
    MPI::Request request;
    MPI::Status status;

    // rank of the worker
    int rank;
    // number of variables in the problem
    int nCols;
    //we need to know how many binary variables are there
    int num_bins;
    //the status of the binary variables in the problem
    int *binaries;
    //the variable index of each binary variable
    int *indexBinVar;

    // instruction from the master, 0 for terminate, 1 for solve
    int instruction;

    //whether the solution is infeasible, feasible or integer feasible
    int solution_status;

    //the value of the feasible or integer feasible solution
```

```

    double solution_value;
    //the value of the variables in the solution
    double *x;
    // which binary variable to branch on
    int branch_variable;
};
#endif /* WORKERCLASS_HPP */

```

Source file

```

#include "workerclass.hpp"

#include <cmath>
#include <sstream>

using namespace std;

workerclass::workerclass() {
}

workerclass::workerclass(const workerclass& orig) {
}

workerclass::~workerclass() {
}

void workerclass::initialize(char* argv, MPI::Intracomm comm) {
    communicator=comm;
    rank = communicator.Get_rank();

    //allocate memory for press problem
    XPRScreateprob(&prob);
    //read the problem from the file path provided from main
    XPRSreadprob(prob, argv, "");

    //Get number of variables in the problem
    XPRSgetintattrib(prob, XPRS_COLS, &nCols);

    char *types = new char[nCols];
    int *temp_bin_indices = new int[nCols];

    //Store type of each variable in the problem
    XPRSgetcoltype(prob, types, 0, nCols - 1);

    num_bins=0;
    // Count number of binary variables in the problem
    for (int i = 0; i < nCols; i++) {
        if (types[i] == 'B') {
            temp_bin_indices[num_bins] = i;
            num_bins++;
        }
    }

    //copy the indices into the proper field and destroy the temporal array
    indexBinVar = new int[num_bins];
    for(int i=0;i<num_bins;i++){

```

```

        indexBinVar[i] = temp_bin_indices[i];
    }
    delete [] temp_bin_indices;
    delete [] types;
}

int workerclass::optimize(){
    //receive from the master a notification of what to do:
    //1 to solve, 0 to terminate
    receive_notification();

    //if I am told to solve
    while (instruction != 0){

        //receive from the master a set of fixed variables
        binaries = new int[num_bins];

        receive_binaries();

        //change the upper bounds according to the fixing vector
        double *bounds = new double[2*num_bins];
        char *type_of_bound = new char[2*num_bins];
        int *var_index = new int[2*num_bins];
        int nn = 0;

        //fill the array to change the upper bounds:
        for(int i=0;i<num_bins;i++){
            if (binaries[i] == 1) {
                type_of_bound[nn]='U';
                bounds[nn] = 1;
                var_index[nn] = indexBinVar[i];
                nn++;
                type_of_bound[nn]='L';
                bounds[nn] = 0;
                var_index[nn] = indexBinVar[i];
                nn++;
            }
            else if (binaries[i] == 1) {
                type_of_bound[nn]='B';
                bounds[nn] = 1;
                var_index[nn] = indexBinVar[i];
                nn++;
            }
            else if (binaries[i] == 0) {
                type_of_bound[nn]='B';
                bounds[nn] = 0;
                var_index[nn] = indexBinVar[i];
                nn++;
            }
        }
    }

    XPRSchgbounds(prob, nn, var_index, type_of_bound, bounds);

    delete [] type_of_bound;
    delete [] bounds;
    delete [] var_index;
}

```



```

//Solve the LP relaxed problem
XPRSlpoptimize(prob, "");

// Get status of the solution
XPRSgetintattrib(prob,XPRS_LPSTATUS,&solution_status);

// Check if the solution is infeasible
if (solution_status == 2) {
    solution_status = 0;
}
// Check if the solution is LP optimal
else if (solution_status == 1) {
    int int_feas = 1;

    // Read the variable values in the solution
    x = new double[nCols];
    XPRSgetlpsol(prob, x, NULL, NULL, NULL);

    //Set initial branch_variable to the first binary variable
    branch_variable = 0;

    // Check if some binary values are not 0 or 1
    for (int i = 0; i < num_bins; i++) {
        if (x[indexBinVar[i]] > 0.0001 and x[indexBinVar[i]] < 0.9999) {

            // Find the variable with solution value closest to 0.5
            //and set as branch variable
            if (abs(x[indexBinVar[i]] - 0.5) <
                abs(x[indexBinVar[branch_variable]] - 0.5)) {
                branch_variable=i;
            }
            //If any of the variables are strictly between upper and
            //lower bounds we no not have an integer feasible solution
            int_feas = 0;
        }
    }

    // Check if the solution is integer feasible
    if (int_feas == 1) {
        solution_status = 2;
    }

    // if not it is a lower bound
    else {
        solution_status = 1;
    }

    // Save the objective value
    XPRSgetdblattrib(prob, XPRS_LPOBJVAL, &solution_value);

    delete[] x;
}

// if LPSTATUS is different from 1 or 2, something is wrong.
else {
    solution_status = 3;
    return_solution();
}

```

```

        break;
    }
    delete [] binaries;

    //Return the solution to the master
    return_solution();

    // Receive new instruction
    receive_notification();
}
//else I am told to terminate
delete [] indexBinVar;
XPRSdestroyprob(prob);
return 0;
}

void workerclass::receive_notification(){
    communicator.Recv(&instruction, 1, MPI::INT, 0, 0, status);
}

void workerclass::receive_binaries() {
    communicator.Recv(binaries, num_bins, MPI::INT, 0, 1, status);
}

void workerclass::return_solution(){
    request=communicator.Issend(&solution_status, 1, MPI::INT, 0, 2);
    request.Wait(status);
    // If solution is LP optimal, return value and branch variable
    if (solution_status==1){
        request=communicator.Issend(&solution_value, 1, MPI::DOUBLE, 0, 3);

        request.Wait(status);

        request=communicator.Issend(&branch_variable, 1, MPI::INT, 0, 4);
        request.Wait(status);
    }
    // if solution is integer feasible, return value
    else if (solution_status==2) {
        request=communicator.Issend(&solution_value, 1, MPI::DOUBLE, 0, 3);
        request.Wait(status);
    }
}
}

```

F.1.4 *nodeclass*

Header file

```

#ifndef NODECLASS_HPP
#define NODECLASS_HPP

#include <float.h>

#include <stddef.h>
#include <cstdio>

class nodeclass {

```

```

public:
    nodeclass ();
    nodeclass (const nodeclass& orig);
    virtual ~nodeclass ();

    void initialize (int *, int, double);
    char get_status () {return status;};
    double get_value () {return value;};
    int *get_binaries () {return fixed_binaries;};
    void get_binaries (int *, int);
    int get_processor () {return processor;};
    double get_lowerbound () {return lower_bound;};
    int get_tag () {return node_tag;};

    void set_status (char p_status) {status = p_status;};
    void set_value (double p_value) {value = p_value;};
    void set_processor (int rank) {processor = rank;};
    void set_lowerbound (double lbound) {lower_bound = lbound;};

    static int master_tag;

private:
    // id of node
    int node_tag;

    //an array of values to indicate whether each binary has been
    //fixed to 1, 0, or relaxed
    int *fixed_binaries;

    //the status of the node, solved or unsolved
    char status;
    //the value of the solution, if solved and feasible
    double value;
    // number of binary variables in the problem
    int num_bins;
    // the node should know which processor, if any, is handling it
    int processor;
    // Value of parent node, optimistic bound on this node
    double lower_bound;
};

#endif /* NODECLASS_HPP */

```

Source file

```

#include "nodeclass.hpp"
#include <iostream>
#include <stdio.h>
#include <float.h>

using namespace std;

int nodeclass::master_tag=0;

nodeclass::nodeclass () {
}

```

```

nodeclass::nodeclass(const nodeclass& orig) {
}

nodeclass::~nodeclass() {
}

void nodeclass::initialize(int *fixed_array, int size, double lbound){
    // every node must know how many binary variables the problem have
    num_bins = size;

    //allocate memory for the array with the status of the binary variables
    fixed_binaries = new int[size];

    //fill in the status of the binary variables
    for(int i=0;i<num_bins; i++) fixed_binaries[i]=fixed_array[i];

    // assign a preliminary value of the node to the
    //maximum positive number for doubles
    value = DBL_MAX;

    // assign a preliminary status of the node as unsolved
    status = 'U';

    // set lower bound as lbound
    lower_bound = lbound;

    // set the id as the current number of master_tag, saying how many nodes
    //have been created so far
    node_tag = master_tag;

    //increment the number of nodes created
    master_tag++;
}

void nodeclass::get_binaries(int* array, int size){
    for(int i=0; i< size; i++) {
        array[i]=fixed_binaries[i];
    }
}

```

F.2 Tree-Based Parallelization

F.2.1 *main*

```

#include <cstdlib>
#include <stdio>
#include "bbclass.hpp"
#include "workerclass.hpp"

using namespace std;

int main(int argc, char** argv) {
    //Start the MPI environment
    MPI::Init();
}

```

```

MPI::Intracomm communicator;

communicator = MPI::COMM_WORLD;

MPI::Request request;
MPI::Status status;

// Start the solver
XPRSinit(NULL);

// If I am the master, I create a bbclass object
if (communicator.Get_rank() == 0){
    //Create a bbclass object
    bbclass instance;
    instance.initialize(argv[1], communicator);
    //Call the bbclass routine to solve
    instance.optimize();

// Else, I am a worker,
} else {
    // Create a workerclass object
    workerclass instance;
    instance.initialize(argv[1], communicator);
    // Call the worker routing to solve
    instance.optimize();
}
// End the solver
XPRSfree();

// End the MPI environment
MPI::Finalize();

return 0;
}

```

F.2.2 *bbclass*

Header file

```

#ifndef BBCLASS_HPP
#define BCLASS_HPP

#include "mpi.h"
#include "xprs.h"

#include <stddef.h>
#include <cstdio>
#include <iostream>

#include <vector>

using namespace std;

class bbclass {
public:
    bbclass();

```

```

    bbclass(const bbclass& orig);
    virtual ~bbclass();

    //initializes the problem
    void initialize(char *, MPI::Intracomm);

    //the optimize function called from main
    int optimize();

    //send a notification to a worker, 1 means solve, 0 means terminate
    void send_notification(int, int);

    //send the status of the binaries of a node to a worker to solve
    void send_binaries(int *, int);

    // receives the solution
    void receive_solution(int);

private:
    MPI::Intracomm communicator;
    MPI::Request request;
    MPI::Status status;

    // number of processors available
    int numProc;

    //rank of a processor
    int myRank;

    //the worker who returned the current incumbent solution
    int incumbent_worker;

    //the incumbent node, if any, found so far.
    double incumbent_value;

    // Number of binary variables in the problem
    int num_bins;

    // Status of the solution from each worker
    int *solution_status;

    // Value of the solution from each worker
    double *solution_value;
};

#endif /* BBCLASS_HPP */

```

Source file

```

#include "bbclass.hpp"
#include <cstdio>
#include <vector>
#include <math.h> /* log2 */
#include <float.h>

using namespace std;

```

```

bbclass::bbclass() {
}

bbclass::bbclass(const bbclass& orig) {
}

bbclass::~bbclass() {
}

void bbclass::initialize(char *argv, MPI::Intracomm comm){
    communicator = comm;

    myRank = communicator.Get_rank();

    numProc=communicator.Get_size();

    //declare an xpress problem object
    XPRSprob prob;
    //allocate memory using the dedicated function
    XPRScreateprob(&prob);
    //read the problem from the file path provided since main
    XPRSreadprob(prob, argv, "");
    //read the number of binary variables in the problem
    int nCols = 0;
    XPRSgetintattrib(prob, XPRS_COLS, &nCols);

    char *types = new char[nCols];
    num_bins=0;
    XPRSgetcoltype(prob, types, 0, nCols-1);
    for(int i=0;i<nCols;i++) if (types[i]=='B') num_bins++;

    incumbent_worker = 0;
    incumbent_value = DBL_MAX;

    delete [] types;
    XPRSdestroyprob(prob);
}

void intToBin(int n, vector<int> &bin){
    int d=n;

    if (n > 1) {
        d = n % 2;
        intToBin(n / 2, bin);
    }

    bin.push_back(d);
}

int bbclass::optimize() {
    // Count how many subtrees have been created
    int node_counter = 0;

    // Number of variables to fix
    int fixed_vars = log2 (numProc-1);

```

```

// Array to send to worker
int *fix_relax = new int[num_bins];

// Arrays to hold solution status and values
solution_status = new int[numProc];
solution_value = new double[numProc];

// Vector to fill in binary representation of a number i up to numProc
vector<int> bin;

for(int i=0; i<numProc-1; i++) {

    // Fill in vector with binary representation of number i
    intToBin(i, bin);

    // Fill vector with 0s before the binary representation
    while(bin.size()<fixed_vars){
        bin.insert(bin.begin(),0);
    }
    // Fill vector with 1 after the binary representation
    while(bin.size()<num_bins){
        bin.push_back(1);
    }
    node_counter++;

    // Fix variables up to fixed_vars
    int fixed = 0;

    fix_relax[0]= 1;

    for(int i=0; i<num_bins-1; i++) {
        // Fix every 40th variable
        if (i % 40 == 0 and fixed<fixed_vars) {
            fix_relax[i+1]=bin[fixed];
            fixed++;
        }
        else {
            fix_relax[i+1]= 1;
        }
    }
    // Send notification
    send_notification(1, node_counter);

    send_binaries(fix_relax, node_counter);

    delete [] fix_relax;
    bin.clear();
}
// Count received solutions
int count_received = 0;

// While not all workers have returned
while (count_received < numProc-1) {
    // Wait until something is received
    while (communicator.Iprobe(MPI::ANY_SOURCE, MPI::ANY_TAG, status) == 0);
    // Save identity of origin worker

```



```

        int origin_worker = status.Get_source();

        // Receive the solution status and value if any
        receive_solution(origin_worker);

        if (solution_status[origin_worker]==1) {
            if(solution_value[origin_worker]<incumbent_value) {
                incumbent_value = solution_value[origin_worker];
                incumbent_worker = origin_worker;
            }
        }
        count_received++;
    }

    delete [] solution_value;
    delete [] solution_status;

    return 0;
}
// Function to send notification to worker, 0 means terminate, 1 means solve
void bbclass::send_notification(int notification, int proc) {
    request = communicator.Issend(&notification, 1, MPI::INT, proc, 0);
    request.Wait(status);
}
// Function to send the status of binary variables to a worker
void bbclass::send_binaries(int* binaries, int proc) {
    request = communicator.Issend(binaries, num_bins, MPI::INT, proc, 1);
    request.Wait(status);
}
// Function to receive solution_status from a worker,
//0 means infeasible solution, 1 means LP feasible solution
//and 2 means integer feasible solution
void bbclass::receive_solution(int origin_worker) {
    communicator.Recv(&solution_status[origin_worker], 1,
        MPI::INT, origin_worker, 2, status);
    // If integer solution was found, receive value as well
    if (solution_status[origin_worker]==1) {
        communicator.Recv(&solution_value[origin_worker], 1,
            MPI::DOUBLE, origin_worker, 3, status);
    }
}
}

```

F.2.3 workerclass

Header file

```

#ifndef WORKERCLASS_HPP
#define WORKERCLASS_HPP

#include "xprs.h"
#include "mpi.h"

#include <cstdlib>

class workerclass {
public:

```

```

workerclass();
workerclass(const workerclass& orig);
virtual ~workerclass();

void initialize(char *, MPI::Intracomm);

//calls xpress to optimize the problem
int optimize();

//receive from the master a notification of what to do:
//1 to solve, 0 to terminate
void receive_notification();

//receives status of binary variables from the master
void receive_binaries();

//returns the worker solution to the master;
void return_solution();

private:
  XPRSProb prob;    //this will store the Xpress object which will be solved

  MPI::Intracomm communicator;
  MPI::Request request;
  MPI::Status status;

  int rank;

  // number of variables in the problem
  int nCols;

  //we need to know how many binary variables are there
  int num_bins;

  //the status of the binary variables in the problem
  int *binaries;

  //the variable index of each binary variable
  int *indexBinVar;

  // instruction from the master, 0 for terminate, 1 for solve
  int instruction;

  //whether the solution is infeasible, feasible or integer feasible
  int solution_status;

  //the value of the feasible or integer feasible solution
  double solution_value;

  //the value of the variables in the solution
  double *x;
};

#endif /* WORKERCLASS_HPP */

```

Source file

```

#include "workerclass.hpp"

#include <cmath>
#include <sstream>

using namespace std;

workerclass::workerclass() {
}

workerclass::workerclass(const workerclass& orig) {
}

workerclass::~workerclass() {
}

void workerclass::initialize(char* argv, MPI::Intracomm comm) {
    communicator=comm;

    rank = communicator.Get_rank();

    //allocate memory using the dedicated function
    XPRScreateprob(&prob);
    //read the problem from the file path provided since main
    XPRSreadprob(prob, argv, "");

    //Get number of variables in the problem
    XPRSgetintattrib(prob, XPRS_COLS, &nCols);

    char *types = new char[nCols];
    int *temp_bin_indices = new int[nCols];

    //Store type of each variable in the problem
    XPRSgetcoltype(prob, types, 0, nCols - 1);

    num_bins=0;
    // Count number of binary variables in the problem
    for (int i = 0; i < nCols; i++) {
        if (types[i] == 'B') {
            temp_bin_indices[num_bins] = i;
            num_bins++;
        }
    }
    //copy the indices into the proper field and destroy the temporal array
    indexBinVar = new int[num_bins];
    for (int i=0;i<num_bins;i++){
        indexBinVar[i] = temp_bin_indices[i];
    }
    delete [] temp_bin_indices;
    delete [] types;
}

int workerclass::optimize(){
    //receive from the master a notification of what to do
    // 1 to solve, 0 to terminate
    receive_notification();

    //if I am told to solve

```

```

while (instruction != 0){

    //receive from the master a set of fixed variables
    binaries = new int[num_bins];
    receive_binaries();

    //change the upper bounds according to the fixing vector
    double *bounds = new double[2*num_bins];
    char *type_of_bound = new char[2*num_bins];
    int *var_index = new int[2*num_bins];

    int nn = 0;
    //fill the array to change the upper bounds:
    for(int i=0;i<num_bins;i++){
        if (binaries[i] == 1) {
            type_of_bound[nn]='U';
            bounds[nn] = 1;
            var_index[nn] = indexBinVar[i];
            nn++;
            type_of_bound[nn]='L';
            bounds[nn] = 0;
            var_index[nn] = indexBinVar[i];
            nn++;
        }
        else if (binaries[i] == 1) {
            type_of_bound[nn]='B';
            bounds[nn] = 1;
            var_index[nn] = indexBinVar[i];
            nn++;
        }
        else if (binaries[i] == 0) {
            type_of_bound[nn]='B';
            bounds[nn] = 0;
            var_index[nn] = indexBinVar[i];
            nn++;
        }
    }
    XPRSchgbounds(prob, nn, var_index, type_of_bound, bounds);

    delete [] type_of_bound;
    delete [] bounds;
    delete [] var_index;

    XPRSsetintcontrol(prob, XPRS_MAXTIME, 2250;

    //Solve the integer problem with the fixed variables
    XPRSmipoptimize(prob, "");

    // Get status of the solution
    XPRSgetintattrib(prob,XPRS_MIPSTATUS,&solution_status);

    // MIP is infeasible.
    if (solution_status == XPRS_MIP_INFEAS) {
        solution_status = 0;
    }
    // Integer solution found, but search is not complete.
    else if (solution_status == XPRS_MIP_SOLUTION) {

```

```

        solution_status = 1;
        XPRSgetdblattrib(prob, XPRS_MIPOBJVAL, &solution_value);
    }
    //MIP is solved to optimality
    else if (solution_status == XPRS_MIP_OPTIMAL) {
        solution_status = 1;
        XPRSgetdblattrib(prob, XPRS_MIPOBJVAL, &solution_value);
    }
    //Search is complete, but no solution found
    else if (solution_status == XPRS_MIP_NO_SOL_FOUND) {
        solution_status = 0;
    }
    // LP has not been optimized.
    else if (solution_status == XPRS_MIP_LP_NOT_OPTIMAL) {
        solution_status = 0;
    }
    //Problem was not loaded
    else if (solution_status == XPRS_MIP_NOT_LOADED) {
        solution_status = 0;
    }
    //Optimization halted after LP optimization
    else if (solution_status == XPRS_MIP_LP_OPTIMAL) {
        solution_status = 0;
        XPRSgetdblattrib(prob, XPRS_LPOBJVAL, &solution_value);
    }
    else {
        solution_status = 0;
    }
    //Return the solution to the master
    return_solution();

    delete [] binaries;
    delete [] indexBinVar;

    XPRSdestroyprob(prob);

    // Receive new instruction
    instruction=0;
}
//else I am told to terminate
return 0;
}
void workerclass::receive_notification(){
    communicator.Recv(&instruction, 1, MPI::INT, 0, 0, status);
}
void workerclass::receive_binaries() {
    communicator.Recv(binaries, num_bins, MPI::INT, 0, 1, status);
}
void workerclass::return_solution(){
    request=communicator.Issend(&solution_status, 1, MPI::INT, 0, 2);
    request.Wait(status);
    // If an integer solution was found, send the value
    if (solution_status == 1) {
        request=communicator.Issend(&solution_value, 1, MPI::DOUBLE, 0, 3);
        request.Wait(status);
    }
}
}

```


Appendix G

Contents of Enclosed CD

Root folder	Contents
Input	Input data files for all three testcases, both in .xlsx and .dat format.
Mosel	Mosel files for the basic instance, and the instance with all cuts, for all three formulations. Mosel code used to generate dynamic clique inequalities.
C++	C++ source and header files for both parallelization frameworks, in separate subfolders.
Output	Output .dat file for the solution illustrated
Report	This report in PDF format.

Table G.1: Contents of enclosed CD.

