Cost allocation and cooperation stability in spare parts inventory pooling

Karsten, F.J.P.

Award date:
2009

Eindhoven, May 2009

Cost allocation and cooperation stability in spare parts inventory pooling

by
Frank Karsten

BSc Industrial Engineering & Management Science — TU/e 2006
Student identity number 513114

in partial fulfilment of the requirements for the degree of

Master of Science
in Operations Management and Logistics

Supervisors:
dr. M. Slikker, TU/e, OPAC
prof. dr. ir. G.J. van Houtum , TU/e, OPAC

TUE. Department Technology Management.
Series Master Theses Operations Management and Logistics

Subject headings: game theory; balancedness; lateral transshipments; inventory pooling; spare parts

Abstract

Companies that stock low-demand expensive spare parts for complex machines can cooperate with other companies that stock the same parts by inventory pooling. This may be implemented via keeping own stockpoints but allowing lateral transshipments in case a demand cannot be satisfied by its own stockpoint. A reduction in total system costs can be achieved this way. This paper deals with the problem of how to distribute these costs between cooperating companies, using cooperative game theoretical models. An investigation on existence of stable cost allocations and selection of proper cost allocations is made.

Preface

This Master Thesis is the result of my graduation project of the Master program Operations Management and Logistics at Eindhoven University of Technology (TU/e). The project took place from October 2008 to April 2009 within the Operations, Planning, Accounting, and Control (OPAC) group of the Industrial Engineering & Innovation Sciences department of the TU/e.

First of all, I would like to thank Marco Slikker, my first supervisor, for your guidance and all the time you invested in my project. I greatly appreciate your constructive feedback, valuable suggestions, critical remarks and helpful ideas and I learned a lot from all the discussions we had. You also helped me to stay on the right track during my project. I would also like to express my thanks to Geert-Jan van Houtum, my second supervisor, for your knowledge and your feedback on my reports. Furthermore, I am grateful to both of you for helping me in my choice on whether to continue the research as a PhD after this project.

Finally, I would like to thank my family for their love and support. And thanks also go out to my fellow students and friends. Without you, my years of study at the university in Eindhoven would not have been as pleasant and enjoyable as they have been.

Frank Karsten

Eindhoven, May 2009

Summary

Topic introduction

Equipment-intensive "high-tech" industries are often confronted with the difficult task of maintaining high availability of their systems. To combat costly downtimes, spare parts for these machines are kept on stock, such that a failed component can be replaced quickly. Lateral transshipments between locations (also referred to as inventory pooling) represent an effective strategy to improve a company's system availability while reducing the total system costs. Lateral transshipments are used to satisfy a demand at a location that is out of stock from another location with a surplus of on-hand inventory.

In settings where each location constitutes an independent company, the analysis of cost sharing amongst the participating companies in a pooling group becomes important. Game theoretic models are appropriate for this. A cooperative spare parts pooling game is characterized by a cost function that determines for a group of cooperating players (a coalition) the yearly expected costs they will have to pay. When we venture into the topic of cost distributions, an important concept is the core; the set of cost allocations that allocate all costs fully and in such a way that no coalition has to pay more than they would have had to pay by acting alone. So, if the core is non-empty, then it is possible to split costs in such a way that no coalition has an incentive to leave the grand coalition (the one including all players) and form a smaller coalition on its own.

Problem statement and research questions

We made the following problem statement:
The scientific literature currently gives no insight into the non-emptiness of the core in a spare parts pooling game and there is insufficient knowledge about proper cost allocation policies that are proven to be in the core of the cooperative game. This lack of managerial insights may impede profitable collaboration on spare parts pooling.

Based on this, we posed the following research questions:
Research question 1a: Does a simple spare parts pooling game, with the base-stock levels already pre-determined at arbitrary values, have a non-empty core?
Research question 1b: Does a simple spare parts pooling game, where the base-stock levels are not yet determined and can still be jointly optimized, have a non-empty core?
Research question 2: What is a proper cost allocation policy for a simple spare parts pooling game?
Research question 3: Can we generalize results to a more complex setting?

Mathematical model

The most important assumptions made in the spare parts inventory model are:

- If a spare part fails, it is replaced with a spare part if one is on stock at the corresponding company. If no spare part is available at the corresponding company when the part fails, but another cooperating company has a surplus on-hand inventory, a lateral transshipment is used. Full pooling is assumed.
- If none of the cooperating companies has a part available, an emergency supply is instigated and the failed machine goes down until the emergency part arrives.
- A failed part is immediately sent into repair (therefore, the inventory system at one company can be seen as being controlled by a base stock policy), after which it is returned to the company.
- There are two relevant types of costs: holding costs and emergency (downtime and shipment) costs. Transshipment (downtime and shipment) costs are assumed to be zero in order to simplify calculations. These parameters, together with given base-stock levels, make up a cost function that gives the expected yearly costs for a coalition of companies.

We discern two cases:

- Base-stock levels are already pre-determined at fixed arbitrary values.
- Base-stock levels can be jointly optimized within a coalition. Then their characteristic costs are found by using the cost-minimizing base-stock levels.

Research question 1: The cores of simple spare parts pooling games

When we assume that base-stock levels are fixed, we proved that:

- For a game where all companies have the same demand rate and the same fixed base-stock levels (other parameters may be asymmetrical), the core is non-empty.
- For a game where all companies have the same emergency costs and the same demand rates (and possibly different base-stock levels), the core is non-empty.

When we assume that base-stock levels are to-be-optimized, we proved that:

- For a game where companies are fully identical, the imputation set (the set of all individually rational efficient cost allocations) is non-empty. But a non-empty imputation set does not always imply a non-empty core.
- For a three-player game where companies are fully identical with realistic real-life base-stock levels, the core is non-empty.

Based on the results of a numerical experiment, we conjectured that any game associated with a spare parts inventory situation where all companies have the same emergency costs has a non-empty core. Games with empty cores have also been found. These are rare, but most often found for games associated with spare parts inventory situations in which the emergency costs differed largely between companies, and where companies had very low repair rates and/or very high holding cost rates. In these games, emergency costs dominate holding costs and companies with low emergency costs take spare parts that would have better been saved for companies with high emergency costs.

Research question 2: Cost allocations

There is a trade-off between (i) simplicity, (ii) always being in the core, (iii) fairness (various fairness properties appropriate for the spare parts setting were defined, e.g., if a company gets a higher demand rate, it should not be allocated less costs). Many allocation methods were considered and tested in a large numerical experiment, but so far no policy was found that satisfies all three requirements. Currently, no allocation rule is available that that easily handles the intricacies of large-scale cooperations of asymmetrical companies well. Two allocations rules that performed reasonably well and that can specifically be applied to spare parts pooling games are:

Allocation SPLIT: Total holding costs are allocated based on the demand rate of each company. Each company pays their own local emergency costs.
The Shapley value: A well-established allocation rule in game theory literature. The idea is to allocate total costs based on the average contribution made to each possible coalition to which a company could belong.

Observations that we can draw from the results of the numerical experiment are:

- For games with 2 companies and/or games with fixed base-stock levels, the Shapley value was often in the core (compared to other allocation rules).
- For games with 3 or 4 companies and/or games with to-be-optimized base-stock levels, allocation rule SPLIT relatively often gave core elements.

Research question 3: Can we generalize results to a more complex setting?

We investigated a setting where transshipment costs were non-negligible with a small numerical experiment. Initial findings indicate that adding transshipment costs does not have a large effect on non-emptiness of the core. However, the cost allocation methods considered above less often produced core elements. However, we stress that this was a very limited study. We also investigated a setting with a smart partial pooling rather than full pooling. Initial findings indicate that using this partial pooling approach leads to significantly fewer games with empty cores. Finally, allocation rules SPLIT and the Shapley value performed best for these more complex settings.

Table of contents

Chapter 1: Introduction

In this chapter, an overview of relevant literature is given. In section 1.1, we introduce the spare parts inventory problem and a well-known model: METRIC. In section 1.2, we review spare parts inventory models with lateral transshipments. In section 1.3, we introduce cooperative game theory and the core concept. In section 1.4, we review the literature on cost allocations.

1.1: Introduction to spare parts and METRIC

Equipment-intensive "high-tech" industries such as airlines, nuclear power plants, medical equipment manufacturers, and complex lithography machines are often confronted with the difficult task of maintaining high availability of their systems. A random failure of just one component can cause the system to go down. To combat these costly downtimes, spare parts for these machines are kept on stock, such that a failed component can be replaced quickly. However, spare parts tend to be quite expensive, which leads to a trade-off between downtimes and inventory investment. As such, important questions to answer are: how many of each spare part should be acquired and kept on stock? And in case the complex machines are geographically dispersed, where should the spare parts be placed?

Spare part inventory systems have been analyzed quite extensively in the literature. An appropriate mathematical model to study the repairable spare parts stocking problem is METRIC (Multi-Echelon Technique for Recoverable Item Control) developed by Sherbrooke (1968). This model considers multi-location inventory systems. For each location we use a base-stock level of S, or one-for-one replenishment, i.e. a failed part will be added back to the spares stock after it is repaired. This can be seen as an (S-1,S) ordering policy. METRIC can help to determine optimal base-stock levels during the acquisition phase of repairable components. Key elements of the METRIC analysis are the assumptions that a location faces Poisson distributed failures (with subsequent demand for spare parts) and that all locations have ample repair capacity. These assumptions facilitate the analysis. See Sherbrooke (2004) and Rustenburg et al. (2003) for more information on METRIC-type models.

1.2: Literature overview of lateral transshipments models

Lateral transshipments between locations (also referred to as inventory pooling) represent an effective strategy to improve a company's system availability while reducing the total system costs, particularly in cases where the transshipment costs are low compared to the downtime costs. Pooling refers to an arrangement in which different locations cooperate by sharing their inventories. Lateral transshipments are used to satisfy a demand at a location that is out of stock from another location with a surplus of on-hand inventory.

The basic METRIC model does not allow lateral transshipments. However, lateral transshipment models have been analyzed quite extensively in the literature. Of the lateral transshipment models available in the literature, we are interested in those models with characteristics that are typical for METRIC (applicable to repairable expensive spare parts): one-for-one replenishment, continuous review policies, and Poisson demands. A comprehensive review of these inventory models is provided in Appendix 1. A short overview and classification of the models used in these papers is given in Table 1.1.

Table 1.1: A brief review of the literature on METRIC-type models with transshipments. In the column "costs considered", (BO_{max}) and ($W_{j,max}$) implies that downtime/backorder costs are not considered but rather shipment and holding costs are minimized subject to a maximum on backorders (BO_{max}) or waiting times ($W_{j,max}$). In the column "delayed laterals allowed", "yes" means that a lateral transshipment is instigated when a repaired spare part arrives at a certain location while another location has a spare part in backorder, and "pro-active laterals" means that lateral transshipments are allowed *before* a location faces demand while out of stock.

Paper	Number of echelons	Number of items	Lateral transship-ment time	Backorders or emergency shipment	Costs con-sidered	Delayed laterals allowed?	Type of pooling	Lateral transshipment sourcing rule	Remarks
Lee (1987)	2	Single	Negligible	Backorders	Holding, backorder, transship	No	complete pooling; N groups	random or maximum stock	Focus on modeling outstanding orders; identical locations
Axsäter (1990)	2	Single	Negligible	Backorders	None	No	complete pooling; N groups	random	Focus on modeling effective demand rate; exponential μ
Alfredsson and Verrijdt (1999)	2	Single	Non-negligible, but identical for all local warehouses	Emergency shipment	Holding, replenish, transship, emergency, waiting penalty	No	complete pooling	random	Only one pooling group; uses Markov analysis
Kukreja et al (2001)	1	Single	Negligible	Backorders	Holding, transship, (BO_{max})	No	complete pooling	Lowest transshipment cost	Show that service performance is not very sensitive to the type of repair time distribution
Grahovac and Chakravar-ty (2001)	2	Single	Non-negligible, determinis-tic	Backorders and emergency shipment	Holding, shipment, waiting	pro-active laterals	complete pooling	random	Retailers place emergency orders once inventory is ≤ trigger level
Wong et al (2005a)	1	Single	Depends on distance	Backorder	Holding, transship, downtime	Yes	complete pooling	Closest neighbor	Multi-hub setting applicable to airline industry
Wong et al (2006)	1	Multi	Depends on item	Emergency shipment	Holding, transship, emergency, ($W_{j,max}$)	No	complete pooling	N/A (only 2 companies)	System approach
Wong et al (2007a)	1	Single	Depends on distance	Backorder	Holding, transship, downtime	Yes	Partial pooling	Closest neighbor	Game theoretic models used
Wong et al (2007b)	2	Multi	Depends on distance	Emergency shipment	Holding, transship, emergency, ($W_{j,max}$)	No	complete pooling	Closest neighbor	Shows that two-echelon system is not worthwhile when transship-ments possible
Kranen-burg & Van Houtum (2008)	1	Multi	Depends on distance	Emergency shipment	Holding, transship, emergency, ($W_{j,max}$)	No	From main ware-houses only	From main warehouses only	Show that only a small number of main local warehouses is sufficient

1.3: Introduction to cooperative game theory and the core concept

So far, we have silently assumed *centralized control*, in which there is only one company with a central planner with the objective to minimize the total system costs. As the use of lateral transshipments generally[1] reduces system costs, this central planner will force all locations to use them. However, in settings where each location constitutes an independent company[2], each company will only agree to pool their spare parts with other companies if doing so will bring more profits to himself. In such a case, the analysis of the cost sharing amongst the participating companies in a pooling group becomes important. Game theoretic models are appropriate for this.

The following preliminary on cooperative game theory and the core concept is taken from Slikker (2007). Cooperative game theory primarily deals with joint profits that can be obtained by groups of players if they coordinate their actions. Let us assume we have n different companies that stock spare parts and that can cooperate via lateral transshipments. In game theory, these companies are referred to as players, with $N=\{1,2,\ldots,n\}$ the set of players. A subset of N is called a coalition and is denoted by M. The *grand coalition* refers to $M=N$. The costs of the coalition M (i.e., the total holding, shipment, and downtime costs made by all companies in this pooling group) can be stated in a single number (in \$) and are freely transferable between the players of M (this means that companies can make transfer payments to each other). The costs payable by the group of cooperating players M is denoted as $c(M)$, i.e. c is the *characteristic cost function* that assigns to every nonempty coalition $M \subseteq N$ a value $c(M)$. A pair (N,c) constitutes a cooperative game. If $\sum_{i\in N} c(\{i\}) \leq c(N)$ then cooperation is beneficial for the grand coalition as a whole.

We now return to the interesting issue of how to distribute the total system costs to each member of a pooling group. Individual companies will, after all, be primarily interested in the individual benefits they can get out of a pooling arrangement. These allocations can be represented with an allocation vector $x \in \mathbb{R}^N$, which specifies for each player $i \in N$ the costs that this player will have to pay if all players cooperate (i.e., they form the grand coalition). There are a couple of interesting conditions that allocation vectors can satisfy:

- Efficiency: $\sum_{i\in N} x_i = c(N)$
- Individual rationality: $x_i \leq c(\{i\})$ for all $i \in N$.
- Stability: $\sum_{i\in M} x_i \leq c(M)$ for all $M \subseteq N$.

The first condition (efficiency) says that all the costs created are in fact split fully among members of the grand coalition.

[1] If transshipment costs are too high, the use of transshipments may actually increase costs. Furthermore, if full pooling is used while companies have different downtime costs, this may also have adverse effects, as will be shown in Chapter 5.

[2] A real life example may be multiple airline companies, who use the same type of aircrafts and independently stock the same spare parts at separate locations.

The second condition (individual rationality) implies that the costs allocated to a player are at most as much as what he had to pay by staying alone. The third condition (stability) says that the costs allocated to any subset of players should be at most as much as what they had to pay by only cooperating together. The set of all individually rational and efficient allocations is called the *imputation set*. The set of all stable and efficient allocations is called the *core*. The core is a subset of the imputation set.

This well-established core concept can be used to "solve" the problem of obtaining a cost allocation that all players can accept. If the core of a game is non-empty, then the total costs for the grand coalition can be distributed to each player in such a way that the costs allocated to any coalition are not larger than the costs that this coalition would have had to pay while acting independently. So, if the costs are split according to a core-element then no coalition has an incentive to leave the grand coalition and form a smaller coalition on its own. Non-emptiness of the core is therefore an important property.

We will now describe the concept of balancedness. A game has a non-empty core *if and only if it is balanced*. Define for all $M \subseteq N$ the vector e^M by $e_i^M = 1$ for all $i \in M$ and $e_i^M = 0$ for all $i \in N \setminus M$. Let 2^N denote the set consisting of all subsets of *N*.

A map $\kappa : 2^N \setminus \{\emptyset\} \to [0,1]$ is called a balanced map if: $\sum_{M \in 2^N \setminus \phi} \kappa(M) e^M = e^N$.

A game (*N*,*c*) is balanced if for every balanced map κ: $\sum_{M \in 2^N \setminus \phi} \kappa(M) c(M) \geq c(N)$.

In Appendix 2, all balancedness conditions for games with 3 and 4 players are stated.

1.4 Literature overview of cost allocations

If the core of a game is non-empty but consists of more than one element, an interesting question is "which core element should we choose to fairly allocate costs to each pooling member?" In this section, we will first describe two cost allocations that are well-established in the literature: the Shapley value and the nucleolus (descriptions based on Slikker, 2007 and Young, 1994, which provides interesting properties of these cost allocations). Subsequently, we provide a brief overview of relevant studies on inventory pooling and/or cost allocation.

The Shapley value $\Phi(N,c)$ is the unique allocation method that is efficient, symmetric, charges zero players nothing, and is additive (see Appendix 2 for definitions):

$$\Phi_i(N,c) = \sum_{M \subseteq N \setminus \{i\}} \frac{|M|! \cdot (|N - M| - 1)!}{|N|!} \cdot \left(c(M \cup \{i\}) - c(M)\right) \text{ for all } i \in N.$$

The nucleolus $\nu(N,c)$ is the unique allocation *x* that maximizes the vector θ(*x*) lexico-graphically, where θ(*x*) has the satisfactions of payoff vector *x* to coalition *M*⊆*N*, $c(M) - \sum_{i \in M} x_i$, ordered increasingly. See Appendix 2 for a more detailed explanation.

Hartman and Dror (1996) suggested three criteria to judge the value of an allocation method:

- Stability (the allocation is in the core of the cooperative game);
- Justifiability (the allocation methods for a cost and benefit game are consistent; i.e. each player's benefit is equal to his individual costs minus his allocated cost);
- Polynomial computability (a computationally easy algorithm).

It was shown that the Shapley value (for a convex game) and the nucleolus satisfy the first two conditions, but they require computations of order $O(2^n)$.

To the best of our knowledge, Wong et al. (2007a) is the first study in the context of spare parts pooling that deals with cost allocations. Four cost allocation policies are proposed:

- Allocate the inventory holding cost based on the number of spare parts stocked at each company; allocate the downtime cost based on the local downtime at each company; and a lateral transportation cost is always paid by the receiving company.
- Allocate the inventory holding cost and lateral transportation cost based on the demand rate of each company; and allocate the downtime cost based on the local downtime at each company.
- Allocate the total cost based on the demand rate of each company.
- The Shapley value.

They apply these four cost allocation policies to a numerical example of a three-company pooling problem, and all four give cost allocations that are in the core of the game. Note that they do not prove non-emptiness of the core, nor do they show that these allocation policies will always be in the core for any input parameters, nor are explanations for the choice of these allocation policies provided. Afterwards, they analyze a non-cooperative setting and a setting with imperfect information. The future research directions identified in their paper provide the inspiration for the research proposal in the next chapter.

Kilpi et al. (2008), who study cooperative strategies for the availability service of repairable aircraft components, mention three cost allocations:

- Share the benefits according to annual demand volume.
- Share the benefits to obtain equal relative savings from joining the pool.
- Share the benefits according to relative incremental pool contribution.

Unfortunately, no clear definitions of these allocations are provided. The first allocation is similar to the third allocation of Wong et al. (2007a), except that benefits rather than costs are allocated.

Chapter 2: Research proposal

In this chapter, we give an overview of the proposed research. In section 2.1, we formulate a problem statement. In section 2.2, we formulate research questions, which flow naturally from the problem statement. In section 2.3, we lay out the contents of the remainder of this thesis.

2.1: Problem statement

In the previous chapter, it was shown that the issue of lateral transshipments in spare parts inventory systems has been a fruitful area of research. However, there is currently a lack of knowledge regarding the stability of cooperation and fair allocation of costs among individual companies that stock spare parts and that can cooperate by means of lateral transshipments. Wong et al. (2007a) have recognized the need to address this problem, but do not attend to the matter of core non-emptiness. They merely state: "Many previous studies have shown that pooling is beneficial in most cases having non-extreme situations ... Moreover, increased savings are realized when more companies are involved. Consequently, most games associated to the spare part pooling have a non-empty core." While this supports the notion that the imputation set of a spare parts pooling game is non-empty, non-emptiness of the core is not a foregone conclusion yet. And while Wong et al. (2007a) provide examples of cost allocation policies, no policy is developed that is proven to be in the core of the game. They do identify these issues as future research directions, however.

Based on these observations, we make the following problem statement:
The scientific literature currently gives no insight into the non-emptiness of the core in a spare parts pooling game and there is insufficient knowledge about proper cost allocation policies that are proven to be in the core of the cooperative game. This lack of managerial insights may impede profitable collaboration on spare parts pooling.

2.2: Research objectives and research questions

Now that the problem has been stated, we formulate research questions which aim to provide a solution to this problem. During the research, we will look at two types of situations, which will be introduced first:

<u>Situation FIX:</u> Every company has already chosen their base-stock levels and they are fixed, i.e. they cannot be altered anymore after a company would join a coalition. This situation could come up when companies consider a short-term cooperation and the costs of buying and/or selling expensive spare parts to optimize base-stock levels just for that short time period are too high. This situation could also happen when the spare part in question is not in production anymore and hence changing inventory levels is practically impossible.

Situation OPT: The companies have not yet determined their base-stock levels and/or they can still be jointly optimized, i.e. for each coalition an optimal (cost-minimizing) base-stock vector (which contains the base-stock levels of all companies in a coalition) can be found. This situation could come up when companies want to go for a long-term cooperation and/or when a new complex machine is installed for which no spare parts have yet been bought.

The goal of the proposed research is to answer the following questions:

Research question 1a: Does a simple[3] spare parts pooling game, with the base-stock levels already pre-determined at arbitrary values, have a non-empty core?

- Is the imputation set non-empty for any combination of input values?
- Is the core non-empty for any combination of input values?
- If not, what are sufficient conditions for non-emptiness?

Research question 1b: Does a simple spare parts pooling game, where the base-stock levels are not yet determined and can still be jointly optimized, have a non-empty core?

- Is the imputation set non-empty for any combination of input values?
- Is the core non-empty for any combination of input values?
- If not, what are sufficient conditions for non-emptiness?

Research question 2: What is a proper cost allocation policy for a spare parts pooling game?

- What are relevant criteria of stability, fairness and simplicity that are appropriate for a cost allocation in the context of spare parts pooling?
- What is a cost allocation policy for a simple spare parts pooling game, with pre-determined arbitrary base stock levels, that adheres to these criteria?
- What is a cost allocation policy for a simple spare parts pooling game, with to-be-optimized base stock levels, that adheres to these criteria?

Research question 3: Can we generalize results to a more complex setting?

- What insights can we provide regarding stability of cooperation and proper cost allocations for a game with non-negligible lateral transshipment costs and for a game with a partial pooling approach?

[3] We aim to provide a mathematical proof for non-emptiness of the core for the most complex spare parts pooling game that we can still properly analyze within the time frame of this project. Hence, a simple spare parts pooling game is one associated with a single-echelon spare parts inventory system, consisting of *N* companies with the same repair rates (but potentially non-identical in other parameters), with negligible lateral transshipment times. This will all be described more clearly/formally in subsequent chapters.

2.3. Layout of the report

The remainder of the report is structured as follows. Chapter 3 provides an overview of the modeling assumptions and defines the concept of spare parts inventory situations. Chapter 4 details the characteristic cost function of simple spare parts pooling games and finishes with an example game. We deal with research question 1 in Chapter 5. This chapter investigates the core of simple spare parts pooling games; it provides certain proofs of non-emptiness and includes a large numerical experiment. We deal with research question 2 in Chapter 6. This chapter introduces interesting cost allocations and cost allocation properties, and includes a large numerical experiment. We deal with research question 3 in Chapter 7. This chapter details the characteristic cost function of a general spare part pooling game and a spare part partial pooling game and includes a small numerical experiment. We end with conclusions in Chapter 8.

Chapter 3: Spare parts inventory situations

In this chapter we introduce spare parts inventory situations. In section 3.1, we provide an overview of the assumptions that are used throughout this thesis. In sections 3.2 and 3.3, we define spare parts inventory situations. In section 3.4 we provide reasonable values for the input parameters.

3.1. Assumptions

The spare parts inventory model analyzed in this thesis can be characterized as a single-echelon, multi-location, single-item model. The main assumptions on the spare parts inventory model throughout this thesis are as follows (for a justification of these assumptions, see Appendix 3):

- **Demand process:** Failures (demands for spare parts) occur according to independent Poisson processes with constant rate (i.e. there is an infinite source of failures), although each company may have a different rate. If a part fails, it is replaced with a spare part if one is on stock at the corresponding company. We consider only one type of spare part (i.e. a single-item model).
- **Cooperation process:** If no spare part is available at the corresponding company when the part fails, but another cooperating company has a surplus on-hand inventory, a lateral transshipment is used from the neighbor that leads to the lowest transshipment costs (with ties broken by sourcing from the neighbor with largest stock on-hand). Complete pooling is applied between cooperating companies[4].
- **Repair process:** A failed part is immediately sent into repair (therefore, the inventory system at one company *i* can be seen as being controlled by a base stock policy with base-stock level $S_i \in \mathbb{N}_0$) and repair lead times are independent and exponentially distributed. Repaired parts are returned to the company that fulfilled the demand for the spare part. All parts are perfectly repairable and there is no condemnation.
- **Emergency supply:** If none of the cooperating companies has a part available, an emergency supply is instigated from an outside infinite source and the machine with the failed part goes down until the emergency part arrives. It is assumed that the expected emergency (downtime and shipment) costs are smaller than the expected downtime costs during a repair, but larger than the expected lateral (downtime and shipment) costs. The failed part is lost to the emergency supplier and does not return to the inventory system.
- **Cost parameters:** There are three relevant types of costs. The first is holding costs, which refers to the yearly capital and storage costs per unit on stock. These costs are incurred when the unit is in the on-hand inventory, but also when it is in repair. The second type of costs is the expected emergency (downtime and shipment) costs. This is the expected total costs incurred when an emergency shipment has to be done.

[4] This assumption of complete pooling will be relaxed in Sections 7.4-7.6 only.

The third type of costs is the expected transshipment (downtime and shipment) costs. This is the expected total costs incurred when a lateral transshipment has to be done. Note that the total cost for emergency and lateral transshipments includes both downtime costs and transportation costs. These are combined in a single parameter in order to keep the number of parameters as low as possible while not significantly diminishing the richness of the model.

- **<u>Goal of individual companies:</u>** Each company aims to minimize expected yearly costs. We have an infinite horizon.

3.2 General spare parts inventory situation

A general spare parts inventory situation is a single-echelon situation where several companies stock a certain spare part, in order to combat costly downtimes of their machines. Lateral transshipments between cooperating companies can be used. A general spare parts inventory situation is uniquely characterized by the set of companies, the demand and repair rates of each company, and the holding, emergency, and transshipment costs of each company. We denote the set of all spare parts inventory situations with Γ. Let $\varphi \in \Gamma$ denote an actual instance. Then φ can be represented by a tuple: $\varphi : \left(N, (\lambda_i)_{i\in N}, (\mu_i)_{i\in N}, (h_i)_{i\in N}, (c_i^{emer})_{i\in N}, (c_{ij}^{trans})_{i\in N, j\in N}\right)$, where:

N	: the set of companies,
λ_i	: the exponential demand rate at company $i \in N$,
μ_i	: the exponential repair rate at company $i \in N$,
h_i	: the holding cost per unit on stock per unit of time at company $i \in N$,
c_i^{emer}	: the expected total costs incurred when an emergency shipment to at company $i \in N$ is done, including both downtime costs and shipment costs,
c_{ij}^{trans}	: the expected total costs incurred when a lateral transshipment from company $i \in N$ to $j \in N$ is done, including both downtime costs and shipment costs.

Note that base-stock levels are not contained in spare parts inventory situations (because when we consider a situation where the base-stock levels are to-be-optimized, they are not input parameters like the variables above).

3.3 More restrictive spare parts inventory situation classes

We can lower the complexity of analyzing a general spare parts inventory situation by assuming certain parameters to be identical for all companies and/or by assuming that lateral transshipments are free and instantaneous. It is easier to make definite statements for these less complex situations and we will do that in subsequent chapters. We need some additional notation for this, however.

<u>Class of simple spare parts inventory situation:</u> $\Gamma_{simple} \subset \Gamma$

For every $\varphi_{simple} \in \Gamma_{simple}$, transshipments are assumed to be free and instantaneous, i.e. $c_{ij}^{trans} = 0$ for all $i, j \in N$. Furthermore, repair rates are assumed to be identical for all companies as well, i.e. $\mu_i = \mu_j = \mu$ for all $i, j \in N$. Hence, a simple spare parts inventory situation can be represented by a tuple: $\varphi_{simple} = \left(N, (\lambda_i)_{i\in N}, \mu, (h_i)_{i\in N}, (c_i^{emer})_{i\in N}\right)$.[5]

<u>Class of simple spare parts inventory sit. with identical c^{emer}:</u> $\Gamma_{simple,id:c^{emer}} \subset \Gamma_{simple}$

For every $\varphi_{simple,id:c^{emer}} \in \Gamma_{simple,id:c^{emer}}$, the emergency costs are assumed to be identical for all companies. It can be represented a tuple: $\varphi_{simple,id:c^{emer}} = \left(N, (\lambda_i)_{i\in N}, \mu, (h_i)_{i\in N}, c^{emer}\right)$.

<u>Class of simple fully identical spare parts inventory situation:</u> $\Gamma_{simple,id:all} \subset \Gamma_{simple}$

For every $\varphi_{simple,id:all} \in \Gamma_{simple,id:all}$, all parameters are assumed to be identical for all companies. It can be represented a tuple: $\varphi_{simple,id:all} = \left(N, \lambda, \mu, h, c^{emer}\right)$.[6]

We can also define many classes of spare parts inventory situations "in between" the above ones, i.e. where some parameters are assumed to be identical for all companies, whereas other parameters are allowed to be different between companies. The definition of these follows along the same lines as above. For reference, they can be found in Appendix 4. The above classes, however, are pivotal in this thesis. Particularly, in chapters 4 through 6 we limit ourselves to games associated with simple spare parts inventory situations.

3.4 Reasonable numerical values

One might wonder what realistic values for these characteristic parameters could be. Selection of reasonable values will be relevant later in this thesis when we do numerical experiments. A literature study has been done in order to find good bounds for real-life company parameters. A number of articles were selected that present either real-life data or that include parameter values that should, according to the article, be representative of real-life data, for various industries using spare parts. The results of this literature study can be found in Appendix 5.

---

[5] We slightly abuse notation here for convenience. In order to make Γ_{simple} a proper subset of Γ, we would actually have to keep the tuple structure identical, i.e. still including c_{ij}^{trans} (=0 for all $i, j \in N$). Hence, the tuple as given in the main text should be seen as a easy shorthand notation of the correct tuple shown in this footnote. Similar remarks can also be made for the other tuples.

[6] We remark that base-stock levels are not contained in spare parts inventory situations. So, even for a simple fully identical spare parts inventory situation, companies may have different base-stock levels.

Based on the values we found in the literature, we selected a minimum and maximum value for each parameter that should provide reasonable bounds for real-life values. These values were rounded and adjusted slightly to get round values and/or to make them usable for the numerical experiments[7]. They can be found in Table 3.1. We remark that in the articles reviewed, emergency costs and transshipment costs were often split up in multiple distinct parts. In our model, only the *total* emergency shipment and *total* lateral transshipment costs are relevant, so we added up parts in order to arrive at a usable total. Similarly, we transformed λ/μ into μ.

Table 3.1: Minimum and maximum values for each spare parts inventory situation parameter that should provide reasonable bounds for real-life values.

Parameter name and unit	Minimum value	Maximum value
λ (demands per year)	0.5	50
μ (repairs per year)	1.67	500
h (\$ per unit per year)	400	28000
c^{emer} (\$ per emergency shipment)	2600	78000
c^{trans} (\$ per lateral transshipment)	300	18000

[7] We required that the calculation time for the optimization algorithm that is used in the numerical experiments was limited. For example, we did not want extremely low holding costs as inputs, since then the optimization algorithm for situation OPT - which works like a "smart enumeration" algorithm - would need to check too large state spaces including "solutions" where companies hold a very large inventory (as low holding costs intuitively lead to high optimal base-stock levels).

Chapter 4: Simple spare parts pooling games

In this chapter we define simple spare parts pooling games, which are introduced in section 4.1. In section 4.2, we define the stock-out probability. Sections 4.3, 4.4, and 4.5 will handle characteristic cost functions. In Section 4.6 an example game is shown.

4.1. Introduction

A *spare parts pooling game* is defined by two elements. The first element is a spare parts inventory situation and the second element is a "rule on base-stock levels". As mentioned in section 2.2, there are two situations that we consider in this thesis:
Situation FIX: Every company has already chosen their base-stock levels and they are fixed, i.e. they cannot be altered anymore after a company would join a coalition. In this case, the "rule on base-stock levels" is a fixed base-stock vector ***S*** that represents the chosen base-stock levels of all companies. Hence, in this case, a game can be associated with a given spare parts inventory situation and a given base-stock vector.
Situation OPT: The companies have not yet determined their base-stock levels and they can still be jointly optimized, i.e. the "rule on base-stock levels" is that for each coalition an optimal (cost-minimizing) base-stock vector ***S*** can be found. Hence, in this case, a game can be associated with just a given spare parts inventory situation.

In this section we will look at the class of simple spare parts inventory situations, Γ_{simple} and describe the characteristic cost functions of *simple spare parts pooling games*[8] that can be associated with it. First, we will need an expression for the fraction of time in which no company has any spare parts on stock and an expression for the expected costs per unit of time that companies will face.

4.2 Stock-out probability for a simple spare parts inventory situation

Consider a simple spare parts inventory situation (recall that this is a situation with identical μ and negligible transshipment costs) and a coalition $M \subseteq N$. Let S_M be the sum of the chosen base-stock levels of all companies in *M* together[9]. Let $\lambda_M = \sum_{i \in M} \lambda_i$ be the total demand rate for all companies in *M* together.

[8] Throughout this thesis, the term *simple spare parts pooling game* refers to a game associated with a simple spare parts inventory situation (and, for situation FIX, a base-stock vector as well). So it is a specific type of spare parts pooling game, for which all statements in the first paragraph of Section 4.1 still hold.
[9] This can be due to an optimal cost-minimizing choice or a non-optimal "dumb" choice for the base-stock levels; for the analysis of the stock-out probability this does not matter. Only the total base-stock level of all companies combined is relevant. Which part is stocked at which company does not matter for the stock-out probability, as they can be shipped back and forth instantaneously at no cost.

Since transshipments are free, companies have an identical exponential repair process, and complete pooling is applied, the system behavior may be described by a one-dimensional Markov process with state *x*, where *x* represents the on-hand inventory at all companies together ($0 \le x \le S_M$).This is represented in Figure 4.1.

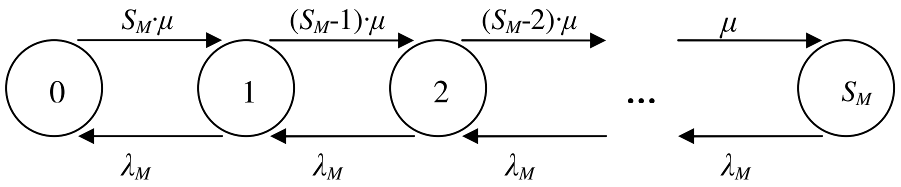

Figure 4.1: The Markov process of a simple spare parts inventory situation.

Note that this Markov process corresponds to an M/M/S_M/S_M queue, also known as a loss system with no additional waiting buffer, where $\rho_M = \lambda_M / \mu$ units of traffic are offered to S_M servers. The steady-state probability of being in state *x*=0 (which is the probability that a new demand arriving at the pooling group has to be fulfilled via an emergency channel because all spare parts are in repair) is equal to the well-known Erlang loss probability (see e.g., Kulkarni, 1999 or Zeng, 2003):

$$\pi_0(\rho_M, S_M) = \left[\frac{\dfrac{\rho_M^{S_M}}{S_M!}}{\displaystyle\sum_{y=0}^{S_M} \frac{\rho_M^{y}}{y!}} \right] \tag{4.1}$$

Note that if companies would have different repair rates, then the state space would have to be extended in order to accommodate the information on the location of each part in repair and then equation (4.1) would not hold. That is the reason why we limit ourselves to simple spare parts inventory situations (i.e., with identical μ) in this chapter.

The Erlang loss function has several useful properties, which are captured in the following Lemma's. These properties will be used in the next chapter in proofs of non-emptiness of the cores of games.

Lemma 4.1: The Erlang loss function is non-decreasing in ρ.
Proof: See problem 2 in Whitt (2002).

Lemma 4.2: The Erlang loss function is decreasing in multiplication, i.e. $\pi_0(t\rho, tS)$ is strictly decreasing in $t \ge 0$.
Proof: See the appendix in Schmidt & Whitt (1981). Furthermore, see Appendix 6 for a different proof methodology, which was independently derived and may be of independent interest.

Lemma 4.3: The Erlang loss function is a convex function of *S* (for every ρ>0).
Proof: See Theorem 1 in Jagers & van Doorn (1986).

Lemma 4.4: $\pi_0(\rho, S)$ can be computed recursively by:

$\pi_0(\rho,0) \equiv 1$ and $\pi_0(\rho,S) = \dfrac{\rho \cdot \pi_0(\rho,S-1)}{S+\rho \cdot \pi_0(\rho,S-1)}$ for $S \in \mathbb{N}$.

Proof: See problem 2 in Whitt (2002).

4.3 Structure of the cost function

Let φ_{simple} be a simple spare parts inventory situation and consider coalition $M \subseteq N$. Let the base-stock vector for this coalition be $\boldsymbol{S} \in \mathbb{N}_0^M$. Figure 4.2 graphically shows all input parameters that are relevant for costs calculation for this type of game.

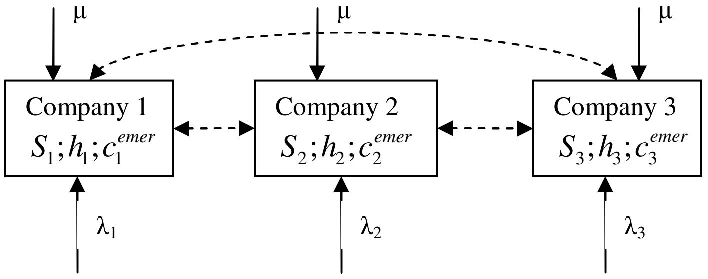

Figure 4.2: The relevant parameters graphically represented for a situation with 3 cooperating companies (*M*={1,2,3}). Dashed arrows indicate a possible transshipment route. Bold arrows indicate demands and repairs.

The total expected costs per unit of time that coalition M has to pay is:

$$\mathrm{K}^{\varphi_{simple};S}(M) = \sum_{i \in M} h_i \cdot S_i + \pi_0\left(\frac{\lambda_M}{\mu}, S_M\right) \cdot \sum_{i \in M} p_i \text{ , where}^{10}: \tag{4.2}$$

h_i	: the holding cost per stocked part per unit of time at company *i*.
μ	: The exponential repair rate.
$S_M = \sum_{i \in M} S_i$	: The sum of the base-stock levels of all companies in *M* together.
$\lambda_M = \sum_{i \in M} \lambda_i$	: The total demand rate for all companies in *M* together.
$\pi_0(\frac{\lambda_M}{\mu}, S_M)$	: The fraction of time in which the on-hand inventory at all companies in *M* together is zero, given by equation (4.1).
$p_i = \lambda_i \cdot c_i^{emer}$	: The expected total cost per unit of time, in which the on-hand inventory at all companies in *M* together is zero, for emergency shipments to company *i*.

[10] Note that (4.2) holds for every $\varphi_{simple} \in \Gamma_{simple}$, hence also for every element of subsets of Γ_{simple}.

4.4. Characteristic cost function for a simple spare parts pooling game with fixed base-stock levels

We consider a situation in which the base-stock levels are already pre-determined (situation FIX). Let φ_{simple} be a simple spare parts inventory situation and let the chosen fixed base-stock vector be $\boldsymbol{S} \in \mathbb{N}_0^N$. With the combination of φ_{simple} and $\boldsymbol{S}$ we can associate a simple spare parts pooling game (with fixed base-stock levels) (N, c) that is defined by $c(M) = \mathrm{K}^{\varphi_{simple};S}(M)$ and K given by equation (4.2). [11]

4.5. Characteristic cost function for a simple spare parts pooling game with to-be-optimized stock levels

We consider a situation in which the base-stock levels are not determined yet and can still be jointly optimized (Situation OPT). Let φ_{simple} be a simple spare parts inventory situation. Consider a coalition $M \subseteq N$. This coalition can choose the base-stock levels for every company $i \in M$. Let the base-stock vector chosen be $\boldsymbol{S} \in \mathbb{N}_0^M$. The combination of φ_{simple} and $\boldsymbol{S}$ leads to a certain expected costs per time unit $\mathrm{K}^{\varphi_{simple};S}(M)$. An optimal base-stock vector for coalition M is $\boldsymbol{S}_M^{opt}$ with minimal cost $\mathrm{K}^{\varphi_{simple};S_M^{opt}}(M)$. So, with φ_{simple} we can associate a simple spare parts pooling game (with to-be-optimized base-stock levels) (N, c) that is defined by $c(M) = \underset{S \in N_0{}^M}{Min}\left(\mathrm{K}^{\varphi_{simple};S}(M)\right)$ and K given by equation (4.2).[12]

A clarification is needed on the concept of an optimal base-stock vector. It can often happen that there are multiple optimal (cost-minimizing) base-stock vectors, especially when all companies have identical holding cost rates. An example of such a case is given in the next section. Later on in this thesis we require a single well-defined optimal base-stock vector. We construct this unique optimal base-stock vector for coalition *M*, $\boldsymbol{S}_M^*(\varphi_{simple})$, by taking an optimal base-stock vector with the lowest possible total sum of base-stock levels and then allocating the base-stock levels to the companies with the lowest holding cost rates in *M* as evenly as possible, with remaining items allocated to companies with the lowest index first. See the example in the next section for an illustration. A more formal definition can be found in Appendix 8.

[11] We slightly abuse notation here (as K is a function $\Gamma * \mathbb{N}^M \rightarrow \mathbb{R}$ rather than $\Gamma * \mathbb{N}^N \rightarrow \mathbb{R}$) in order to avoid excessive notation. The proper definition is $c(M) = \mathrm{K}^{\varphi_{simple};S^M(S)}(M)$, where for all $M \subseteq N$ the base-stock vector of coalition *M*, $\boldsymbol{S}^{\boldsymbol{M}}(S) \in \mathbb{N}_0^M$ is defined via $S_i^M = S_i$ for all $i \in M$.

[12] We have silently assumed that no company has a holding cost rate of 0, since in that case it would be optimal to have an infinitely large base-stock level at that company, which is obviously practically infeasible and not well-defined.

We shall now construct an easier variation of the cost function (4.2) that can simplify notation used in proofs in the next chapter. We use two insights:

- Recall that for the stock-out probability π_0 only the total sum of base-stock levels of all companies combined is relevant; which part is stocked at which company does not matter, as they can be shipped back and forth instantaneously at no cost.
- For an optimal base-stock vector of coalition *M*, a company that has a higher holding cost rate than another company in *M*, will certainly have a base-stock level of 0, i.e. only companies that have the lowest holding cost rate in *M* can have positive base-stock levels. Therefore, for the total holding costs, only the total sum of base-stock levels of all companies combined is relevant.

As such, only the total sum of base-stock levels of all companies combined, $S_M^{tot} \in \mathbb{N}_0$, is relevant for the cost function and we can construct an easier variant of (4.2):

$$\mathrm{K}_{tot}^{\varphi_{simple};S_M^{tot}}(M) = h_{\min} \cdot S_M^{tot} + \pi_0\left(\lambda_M / \mu, S_M^{tot}\right) \cdot \sum_{i \in M} p_i \text{, where } h_{\min} = \min(h_i \mid i \in M) \qquad (4.3)$$

This means that with φ_{simple} we can associate a simple spare parts pooling game (with to-be-optimized base-stock levels) (N, c) that is defined by $c(M) = \underset{S_M^{tot} \in N_0}{Min}\left(\mathrm{K}_{tot}^{\varphi_{simple};S_M^{tot}}(M)\right)$ and K$_{tot}$ given by equation (4.3). Note that this cost function definition is identical (albeit formulated differently) to the definition given earlier in this section. The optimal sum of base-stock levels for coalition *M* is $S_M^{tot} * (\varphi)$ (formally defined in Appendix 8).

4.6 Example games

In this section we provide an example in which we apply the formulas and concepts introduced in this chapter. Consider the 3-player simple spare parts inventory situation $\varphi_{simple,id:c^{emer}} \in \Gamma_{simple,id:c^{emer}}$ with identical c^{emer} $\left(N, (\lambda_i)_{i \in N}, \mu, (h_i)_{i \in N}, c^{emer}\right)$ with *N*={1,2,3}, λ_1=0.5, λ_2=5, λ_3=50, μ=25, h_1=2000, h_2=28000, h_3=2000, and c^{emer}=13000. So company 3 has a very high demand rate while company 1 has a very low demand rate. Furthermore, company 1 and 3 have low holding costs while company 2 has high holding costs.

<u>Example costs calculation</u>

Suppose that we have situation FIX with base-stock vector ***S*** given by S_1=4, S_2=0, S_3=4. The costs of coalition {1,2} are given by (values are rounded to two decimals):

$$c(\{1,2\}) = \mathrm{K}^{\varphi_{simple,id:c^{emer}};S}(\{1,2\}) = \left(h_1 \cdot S_1 + h_2 \cdot S_2\right) + \pi_0\left(\frac{\lambda_1 + \lambda_2}{\mu}, S_1 + S_2\right) \cdot \left(\lambda_1 \cdot c^{emer} + \lambda_2 \cdot c^{emer}\right)$$

$$= 8000 + \pi_0(0.22,4) \cdot 71500 = 8000 + \left(\frac{0.22^4}{4!}\right) \bigg/ \left(\sum_{y=0}^{4} \frac{0.22^y}{y!}\right) \cdot 71500 = 8000 + 7.83 \cdot 10^{-5} \cdot 71500$$

$$= 8005.60$$

<u>Example game with fixed base-stock levels</u>

We can do similar calculations for each coalition in N. The spare parts pooling game associated with $\varphi_{simple,id:c^{emer}}$ and $\boldsymbol{S}$, (N,c), is described by (values are rounded):

c({1})	= 8,000.00;	c({2})	= 65,000.00;	c({3})	= 69,904.76;
c({1,2})	= 8,005.60;	c({1,3})	= 16,598.91;	c({2,3})	= 91,372.34;
c({1,2,3})	= 17,147.16.				

Note that this spare parts pooling game is balanced, i.e. it has a non-empty core. This can be shown in two different ways.

- There is at least one element in the core, e.g., an allocation x_1=154.48, x_2=1,544.79, x_3=15,447.89 is in the core (it adheres to efficiency and stability).
- All balancedness conditions (see Appendix 2) hold, i.e.:
  c({1,2,3})=17,147.16 $\leq$ c({1})+c({2})+c({3}) =142,904.76
  c({1,2,3})=17,147.16 $\leq$ 0.5·c({1,2})+0.5·c({1,3})+0.5·c({2,3}) =57,988.43
  c({1,2,3})=17,147.16 $\leq$ c({3})+c({1,2}) =77,910.36
  c({1,2,3})=17,147.16 $\leq$ c({2})+c({1,3}) =81,598.91
  c({1,2,3})=17,147.16 $\leq$ c({1})+c({2,3}) =99,372.34

<u>Example game with to-be-optimized base-stock levels</u>

Now suppose that we have situation OPT, so base-stock vectors are to-be-optimized. The spare parts pooling game associated with $\varphi_{simple,id:c^{emer}}$, (N,c) (and, implicitly, the rule to optimize base-stock vectors), is described by (values are rounded to two decimals):

c({1}) = 2,127.45 (with $S^{tot}_{\{1\}}*$=1 and $\boldsymbol{S}^*_{\{1\}}$ given by $(S^*_{\{1\}})_1$=1)

c({2}) = 38,833.33 (with $S^{tot}_{\{2\}}*$=1 and $\boldsymbol{S}^*_{\{2\}}$ given by $(S^*_{\{2\}})_2$=1)

c({3}) = 16,236.56 (with $S^{tot}_{\{3\}}*$=7 and $\boldsymbol{S}^*_{\{3\}}$ given by $(S^*_{\{3\}})_3$=7)

c({1,2}) = 5,390.69 (with $S^{tot}_{\{1,2\}}*$=2 and $\boldsymbol{S}^*_{\{1,2\}}$ given by $(S^*_{\{1,2\}})_1$=2 and $(S^*_{\{1,2\}})_2$=0)

c({1,3}) = 16,374.08 (with $S^{tot}_{\{1,3\}}*$=7 and $\boldsymbol{S}^*_{\{1,3\}}$ given by $(S^*_{\{1,3\}})_1$=4 and $(S^*_{\{1,3\}})_3$=3)

c({2,3}) = 17,078.76 (with $S^{tot}_{\{2,3\}}*$=8 and $\boldsymbol{S}^*_{\{2,3\}}$ given by $(S^*_{\{2,3\}})_2$=0 and $(S^*_{\{2,3\}})_3$=8)

c({1,2,3}) = 17,147.16 (with $S^{tot}_{N}*$=8 and $\boldsymbol{S}^*_{N}$ given by $(S^*_{N})_1$=4, $(S^*_{N})_2$=0, $(S^*_{N})_3$=4)

Note that this spare parts pooling game is also balanced, i.e. it has a non-empty core. For example, allocation x_1=154.48, x_2=1544.79, x_3=15447.89 is in the core.

We remark that the optimal base-stock vector for coalition {2,3} clearly puts all stock at the company with the lower holding cost rate. Furthermore, note that in the optimal base-stock vectors of coalitions {1,3} and {1,2,3} the base-stock levels are allocated as equally as possible to the companies with the lowest holding costs rates in M, with remaining items allocated to companies with the lowest index (i.e. company 1) first. An algorithm to find $S^{tot}_M *$ and $\boldsymbol{S}^*_M$ for all $M \subseteq N$ is used in the numerical experiments of the subsequent chapters and is described in Appendix 8. Finally, we remark that the $\boldsymbol{S}$ that was chosen for situation FIX is identical to $\boldsymbol{S}^*_N$. Hence, the costs of the grand coalition for both games are the same. But costs for subsets of N are not the same for both games. Particularly, company 1 is stocking too much for its own good in situation FIX.

Chapter 5: The core of simple spare parts pooling games

At the end of the last chapter, we saw example games that had non-empty cores. In this chapter we attempt to answer research question 1 ("Does a simple spare parts pooling game always have a non-empty core?"). In section 5.1, we cover simple pooling games with fixed base-stock levels and in section 5.2, we cover simple pooling game with to-be-optimized base-stock levels. In these sections, we shall provide certain proofs of core non-emptiness. In section 5.3, we provide counter-examples showing that for games associated with certain classes of spare parts inventory situations, the core is not always non-empty. Section 5.4 describes a large numerical experiment that investigates core non-emptiness further. We summarize all findings in section 5.5.

5.1: Simple pooling games with fixed base-stock levels

In this section, we give proofs that games associated with certain types of simple spare parts inventory situations and base-stock vectors in fact always have non-empty cores. Recall that for simple spare parts inventory situations, transshipments are assumed to be free and repair rates are assumed to be identical.

5.1.1: Situations with identical λ and S

Let $\varphi_{simple,id:\lambda}$ be a simple spare parts pooling game (with identical λ) and let $\boldsymbol{S}$ be a base-stock vector with identical base-stock levels for each company, i.e. S_i=S for all $i \in N$. The associated simple spare parts pooling game has a non-empty core, as stated in the following theorem. We remark that this holds for any number of companies that may have different holding cost rates and/or emergency (shipment and downtime) costs.

Theorem 5.1: Let $\varphi_{simple,id:\lambda} \in \Gamma_{simple,id:\lambda}$ and let $\boldsymbol{S} \in \mathbb{N}_0^N$ be fixed with S_i=S for all $i \in N$. The associated spare parts pooling game (N,c) has a non-empty core.
Proof:
Let $\lambda_M = \lambda \cdot |M|$ and let $S_M = S \cdot |M|$ for all $M \subseteq N$. Let $p_i = \lambda \cdot c_i^{emer}$ for all $i \in N$.
Let $\boldsymbol{x} \in \mathbb{R}^N$ be a cost allocation vector with $x_i = h_i \cdot S + \pi_0(\lambda_N / \mu, S_N) \cdot p_i$ for all $i \in N$.
We will show that $\boldsymbol{x}$ is (a) efficient and (b) stable, and hence $\boldsymbol{x}$ is a core element.
Efficiency follows from $\sum_{i \in N} x_i = \sum_{i \in N} h_i \cdot S + \pi_0(\lambda_N / \mu, S_N) \cdot \sum_{i \in N} p_i = c(N)$.
In order to prove stability, i.e. $\sum_{i \in M} x_i \leq c(M)$ for all $M \subseteq N$, let $M \subseteq N$ and start with:

$$\pi_0(\lambda_N / \mu, S_N) \leq \pi_0(\lambda_M / \mu, S_M). \tag{5.1}$$

(5.1) holds by Lemma 4.2. Multiply (5.1) by $\sum_{i \in M} p_i$ and then add $\sum_{i \in M} h_i \cdot S$:

$$\sum_{i \in M} h_i \cdot S + \pi_0(\lambda_N / \mu, S_N) \cdot \sum_{i \in M} p_i \leq \sum_{i \in M} h_i \cdot S + \pi_0(\lambda_M / \mu, S_M) \cdot \sum_{i \in M} p_i \tag{5.2}$$

(5.2) is equivalent to $\sum_{i\in M} x_i \le c(M)$. We conclude $\boldsymbol{x}$ is stable. This completes the proof.

■

Note that our proof is based on showing a core element. This particular allocation is easy to administer, as each company simply pays its own holding costs and its own local emergency (downtime and shipment) costs. There are no transfer costs. We will return to cost allocations in more detail in the next chapter.

5.1.2: Situations with identical λ, and c^{emer}

Now, let $\varphi_{simple,id:\lambda,c^{emer}}$ be a simple spare parts pooling game with identical λ and c^{emer} and let $\boldsymbol{S}$ be a base-stock vector. The associated simple spare parts pooling game has a non-empty core, as stated in Theorem 5.2. We remark that this holds for any number of companies that may have different holding costs rates and/or base-stock levels. First, however, we need to state two lemmas that will be used in the proof of Theorem 5.2. These involve balanced maps, for which a definition can be found in Section 1.3. Throughout this section, 2^N denotes the set consisting of all subsets of some player set N.

Lemma 5.1: Let N be a player set, let κ:$2^N\backslash\emptyset\rightarrow[0,1]$ be a balanced map and let $f(i)$ be a function f:$N\rightarrow\mathbb{R}$. Then: $\sum_{M\in 2^N\backslash\phi}\kappa(M)\cdot\sum_{i\in M} f(i)=\sum_{i\in N} f(i)$.

Proof:

Let $e_i^M=\begin{cases}1 & ,i\in M\\ 0 & ,i\in N\backslash M\end{cases}$.

For all $i\in N$: $\sum_{M\in 2^N\backslash\phi}\kappa(M)\cdot e_i^M=1$. (5.3)

(5.3) holds by definition of κ as a balanced map.

Multiply both sides of (5.3) by $f(i)$, then sum both sides of (5.3) over all $i\in N$:

$$\sum_{M\in 2^N\backslash\phi}\kappa(M)\cdot\sum_{i\in N} e_i^M\cdot f(i)=\sum_{i\in N} f(i). \tag{5.4}$$

Note that $\sum_{i\in N} e_i^M\cdot f(i)=\sum_{i\in M} f(i)$. So, rewriting the left side of (5.4) completes the proof.

■

Lemma 5.1 implies the following relations that will be used in later proofs in this section:

- $\sum_{M\in 2^N\backslash\phi}\kappa(M)\cdot|M|=|N|$ (take $f(i)=1$).
- $\sum_{M\in 2^N\backslash\phi}\kappa(M)\cdot\sum_{i\in M} h_i\cdot S_i=\sum_{i\in N} h_i\cdot S_i$ (take $f(i)=h_i\cdot S_i$).
- $\sum_{M\in 2^N\backslash\phi}\kappa(M)\cdot\sum_{i\in M} S_i=\sum_{i\in N} S_i$ (take $f(i)=S_i$).

The following lemma considers the stock-out probability in a simple spare parts inventory situation with identical demand rates $\varphi_{simple,id:\lambda}$. For notational ease, we use $\rho = \lambda / \mu$. Lemma 5.2 states that the weighed combination of total stock-out probabilities of all coalitions $M \subseteq N$ is at least as large as the total stock-out probabilities of the grand coalition. Formally this is captured in equation (5.5).

Lemma 5.2: Let *N* be a player set, κ:$2^N\backslash\emptyset\rightarrow[0,1]$ be a balanced map, $\boldsymbol{S}\in\mathbb{N}_0^N$ and let $\rho>0$.
Then: $$\sum_{M\in 2^N\backslash\phi}\kappa(M)\cdot|M|\cdot\pi_0\left(\sum_{i\in M}S_i\ ,\ |M|\cdot\rho\right)\geq|N|\cdot\pi_0\left(\sum_{i\in N}S_i\ ,\ |N|\cdot\rho\right)\tag{5.5}$$
Proof:
First we define five variables. *a* is the right-hand side of equation (5.5) and *e* is the left-hand side of equation (5.5). We compare *a* and *e* to variables *b*, *c* and *d*, using properties of the Erlang loss function π_0, in order to eventually show that $e \geq a$.

$$a = |N|\cdot\pi_0\left(\sum_{i\in N}S_i\ ,\ |N|\cdot\rho\right)$$

$$b = |N|\cdot\pi_0\left(\sum_{M\in 2^N\backslash\phi}\kappa(M)\cdot\sum_{i\in M}S_i\ ,\ |N|\cdot\rho\right)$$

$$c = |N|\cdot\pi_0\left(\sum_{M\in 2^N\backslash\phi}\kappa(M)\cdot\frac{|M|}{|N|}\cdot\frac{|N|}{|M|}\cdot\sum_{i\in M}S_i\ ,\ |N|\cdot\rho\right)$$

$$d = |N|\cdot\sum_{M\in 2^N\backslash\phi}\kappa(M)\cdot\frac{|M|}{|N|}\cdot\pi_0\left(\frac{|N|}{|M|}\cdot\sum_{i\in M}S_i\ ,\ |N|\cdot\rho\right)$$

$$e = \sum_{M\in 2^N\backslash\phi}\kappa(M)\cdot|M|\cdot\pi_0\left(\sum_{i\in M}S_i\ ,\ |M|\cdot\rho\right)$$

By Lemma 5.1, *a*=*b*. Furthermore, clearly *b*=*c*. By convexity of $\pi_0(S,\rho)$ in *S* (by Lemma 4.3) and because $\sum_{M\in 2^N\backslash\phi}\kappa(M)\cdot\frac{|M|}{|N|}=1$ by Lemma 5.1, $d \geq c$. Because $\pi_0(tS,t\rho)$ is strictly decreasing in $t \geq 0$ (by Lemma 4.2), and because both |*N*| terms cancel out, $e \geq d$. Hence, we conclude $e \geq a$.
■

Theorem 5.2: Let $\varphi_{simple,id:\lambda,c^{emer}} \in \Gamma_{simple,id:\lambda,c^{emer}}$ and let $\boldsymbol{S}\in\mathbb{N}_0^N$. The associated spare parts pooling game (*N*,*c*) has a non-empty core.
Proof:
Let κ:$2^N\rightarrow[0,1]$ be a balanced map. By Lemma 5.2 we have:
$$\sum_{M\in 2^N\backslash\phi}\kappa(M)\cdot|M|\cdot\pi_0\left(\sum_{i\in M}S_i\ ,\ |M|\cdot\rho\right)\geq|N|\cdot\pi_0\left(\sum_{i\in N}S_i\ ,\ |N|\cdot\rho\right)\tag{5.6}$$
Multiply both sides of (5.6) by $p = \lambda\cdot c^{emer}$, then add $\sum_{i\in N}h_i\cdot S_i$ to both sides to obtain:

$$\sum_{i\in N} h_i \cdot S_i + \sum_{M\in 2^N\setminus\phi} \kappa(M)\cdot |M| \cdot \pi_0\left(\sum_{i\in M} S_i, |M|\cdot\rho\right)\cdot p \geq \sum_{i\in N} h_i \cdot S_i + |N|\cdot \pi_0\left(\sum_{i\in N} S_i, |N|\cdot\rho\right)\cdot p \quad (5.7)$$

By Lemma 5.1, we can rewrite the holding cost terms on the left side of (5.7) to obtain:

$$\sum_{M\in 2^N\setminus\phi} \kappa(M)\cdot\left[\sum_{i\in M} h_i \cdot S_i + |M|\cdot \pi_0\left(\sum_{i\in M} S_i, |M|\cdot\rho\right)\cdot p\right] \geq \sum_{i\in N} h_i \cdot S_i + |N|\cdot \pi_0\left(\sum_{i\in N} S_i, |N|\cdot\rho\right)\cdot p \quad (5.8)$$

Note that (5.8) is identical to:

$$\sum_{M\in 2^N\setminus\phi} \kappa(M)\cdot c(M) \geq c(N) \quad (5.9)$$

As κ was arbitrarily chosen, equation (5.9) holds for any balanced map. Hence, equation (5.9) states that the game is balanced and therefore we conclude it has a non-empty core.
■

5.2: Simple pooling games with to-be-optimized stock levels

In this section, we give proofs that games associated with certain types of simple spare parts inventory situations have non-empty cores. Let $\varphi_{simple,id:all}$ be a simple fully identical spare parts pooling game. The associated spare parts pooling game has a non-empty imputation set, as stated in the following lemma. We remark that this holds for any number of companies.

Lemma 5.3: Let $\varphi_{simple,id:all} \in \Gamma_{simple,id:all}$. The associated spare parts pooling game (*N*,*c*) has a non-empty imputation set.

Proof:

Recall that $S_M^{tot}*$ denotes the optimal base-stock level sum for coalition $M \subseteq N$.

Let $\lambda_N = \lambda\cdot|N|$. Let *i*∈*N*. We start with:

$$\pi_0\left(\frac{\lambda_N}{\mu}, |N|\cdot S_{\{i\}}^{tot}*\right) \leq \pi_0\left(\frac{\lambda}{\mu}, S_{\{i\}}^{tot}*\right) \quad (5.10)$$

(5.10) holds by Lemma 4.2. Multiply (5.10) by $\lambda\cdot c^{emer}$ and then add $h\cdot S_{\{i\}}^{tot}*$ to obtain:

$$h\cdot S_{\{i\}}^{tot}* + \pi_0\left(\frac{\lambda_N}{\mu}, |N|\cdot S_{\{i\}}^{tot}*\right)\cdot\lambda\cdot c^{emer} \leq h\cdot S_{\{i\}}^{tot}* + \pi_0\left(\frac{\lambda}{\mu}, S_{\{i\}}^{tot}*\right)\cdot\lambda\cdot c^{emer} \quad (5.11)$$

Multiply (5.11) by |*N*| and rewrite to obtain:

$$h\cdot S_{\{i\}}^{tot}*\cdot|N| + \pi_0\left(\frac{\lambda_N}{\mu}, |N|\cdot S_{\{i\}}^{tot}*\right)\cdot\sum_{i\in N}\lambda\cdot c^{emer} \leq |N|\cdot\left(h\cdot S_{\{i\}}^{tot}* + \pi_0\left(\frac{\lambda}{\mu}, S_{\{i\}}^{tot}*\right)\cdot\lambda\cdot c^{emer}\right) \quad (5.12)$$

We remark that the since all companies are identical, $S_{\{i\}}^{tot}* = S_{\{j\}}^{tot}*$ for all $i, j \in N$. Hence, (5.12) is equivalent to:

$$\mathrm{K}_{tot}^{\varphi_{simple,id:all};S_{\{i\}}^{tot}*\cdot|N|}(N) \leq \sum_{j\in N}\mathrm{K}_{tot}^{\varphi_{simple,id:all};S_{\{j\}}^{tot}*}(\{j\}) = \sum_{j\in N} c(\{j\})$$

Observing $c(N) = \mathrm{K}_{tot}^{\varphi_{simple,id:all};S_N^{tot}*}(N) \le \mathrm{K}_{tot}^{\varphi_{simple,id:all};S_{\{i\}}^{tot}*,|N|}(N)$ and thus $c(N) \le \sum_{j\in N} c(\{j\})$ completes the proof.

■

In Lemma 5.3, the optimal $S_M^{tot}*$ could take on any value. We will now limit us to games associated with more restrictive rules on base-stock levels, so that we can make a definite statement about the core of a less complex game. Particularly, we set the following limit: $S_M^{tot}* \le 20\cdot|M|$ for all $M \subseteq N$. Furthermore, we limit the number of companies to three.

This maximum on the base-stock level is not constraining for practical situations, since the spare parts, for which pooling via lateral transshipments is interesting, are characterized by low demand rates, high holding costs, and low optimal base-stock levels (see e.g., Wong (2006) for representative sample numerical values of base-stock levels). The reason for setting this maximum lies in the proof methodology[13].

Let $\varphi_{simple,id:all,|N|=3}$ be a simple fully identical spare parts pooling game with N={1,2,3}. Lemma 5.4 states that if $S_{\{1,2\}}^{tot}*$ is odd, then the stock-out probability for three companies using a randomized base-stock vector of $1.5\cdot S_{\{1,2\}}^{tot}*$ is at most as large as the stock-out probability for companies 1 and 2. This is subsequently used in Theorem 5.3, which states that the spare parts pooling game, associated with $\varphi_{simple,id:all,|N|=3}$ and the limit of $S_M^{tot}* \le 20\cdot|M|$ for all $M \subseteq N$, has a non-empty core.

Lemma 5.4: Let ρ>0 and let $S_{\{1,2\}}^{tot}* \in \{1,3,5,...,37,39\}$. Then:

$$\frac{1}{2}\pi_0\left(3\rho, 1.5\cdot S_{\{1,2\}}^{tot}*-0.5\right)+\frac{1}{2}\pi_0\left(3\rho, 1.5\cdot S_{\{1,2\}}^{tot}*+0.5\right) \le \pi_0\left(2\rho, S_{\{1,2\}}^{tot}*\right). \qquad (5.13)$$

Proof: See Appendix 7[14].

Theorem 5.3: Let $\varphi_{simple,id:all,|N|=3} \in \Gamma_{simple,id:all,|N|=3}$ such that $S_M^{tot}* \le 20\cdot|M|$ for all $M \subseteq N$. The associated spare parts pooling game (N,c) has a non-empty core.

[13] We wish to take $S_{\{1,2,3\}}^{tot} = 1.5 S_{\{1,2\}}^{tot}*$ because then when you compare the two coalitions, the holding costs per company are the same. This approach does not work nicely when $S_{\{1,2\}}^{tot}*$ is an odd number, since you can then only choose $1.5\cdot S_{\{1,2\}}^{tot}*-0.5$ and $S_{\{1,2\}}^{tot}*+0.5$ as integer values. We can still make a useful statement regarding stock-out probabilities for this situation in Lemma 5.4, but can only numerically verify this for some $S_{\{1,2\}}^{tot}*$.

[14] Note that it will very likely also hold for odd values of $S_{\{1,2\}}^{tot}*$>39, but this has not been verified yet. The proof provides a methodology to numerically check the validity of the statement for any odd value of $S_{\{1,2\}}^{tot}*$, and this has been done up to 39. This was deemed a large enough value for practical purposes (furthermore, larger numbers result in factorial overflows that computers / calculators can't handle).

Proof:
We will show balancedness by proving each balancedness condition individually, i.e.:
(1) $c(\{1,2,3\}) \leq c(\{1\}) + c(\{2) + c(\{3\})$.
(2) $c(\{1,2,3\}) \leq 0.5 \cdot c(\{1,2\}) + 0.5 \cdot c(\{1,3\}) + 0.5 \cdot c(\{2,3\})$
(3) $c(\{1,2,3\}) \leq c(\{1,2\}) + c(\{3\}) = c(\{1,3\}) + c(\{2\}) = c(\{2,3\}) + c(\{1\})$

(1) follows directly from Lemma 5.3.

For **(2)** we use the property that all companies are identical, so it suffices to show $c(\{1,2,3\}) \leq 1.5 \cdot c(\{1,2\})$. We then discern between (2a) $S_{\{1,2\}}^{tot}*$ is even (2b) $S_{\{1,2\}}^{tot}*$ is odd.
In case **(2a)** $S_{\{1,2\}}^{tot}*$ is even, we start with:

$$\pi_0\left(\frac{3\lambda}{\mu}, 1.5 \cdot S_{\{1,2\}}^{tot}*\right) \leq \pi_0\left(\frac{2\lambda}{\mu}, S_{\{1,2\}}^{tot}*\right) \tag{5.14}$$

(5.14) holds by Lemma 4.2. Multiply (5.14) by $\sum_{i\in\{1,2\}} \lambda \cdot c^{emer}$, then add $h \cdot S_{\{1,2\}}^{tot}*$, then multiply by 1.5 to obtain:

$$h \cdot 1.5 \cdot S_{\{1,2\}}^{tot}* + \pi_0\left(\frac{3\lambda}{\mu}, 1.5 \cdot S_{\{1,2\}}^{tot}*\right) \cdot \sum_{i\in N} \lambda \cdot c^{emer} \leq 1.5 \cdot \left(h \cdot S_{\{1,2\}}^{tot}* + \pi_0\left(\frac{2\lambda}{\mu}, S_{\{1,2\}}^{tot}*\right) \cdot \sum_{i\in\{1,2\}} \lambda \cdot c^{emer}\right) \tag{5.15}$$

(5.15) is equivalent to $\mathrm{K}_{tot}^{\varphi_{simple,id:all,|N|=3;S_{\{1,2\}}^{tot}*\cdot 1.5}}(N) \leq 1.5 \cdot c(\{1,2\})$. Finally, since $c(N) \leq \mathrm{K}_{tot}^{\varphi_{simple,id:all,|N|=3;S_{\{1,2\}}^{tot}*\cdot 1.5}}(N)$, we have $c(N) \leq 1.5 \cdot c(\{1,2\})$.

In case **(2b)** $S_{\{1,2\}}^{tot}*$ is odd, let $S^- = 1.5 \cdot S_{\{1,2\}}^{tot}* - 0.5$ and $S^+ = 1.5 \cdot S_{\{1,2\}}^{tot}* + 0.5$. Then:

$$\frac{1}{2}\pi_0\left(\frac{3\lambda}{\mu}, S^-\right) + \frac{1}{2}\pi_0\left(\frac{3\lambda}{\mu}, S^+\right) \leq \pi_0\left(\frac{2\lambda}{\mu}, S_{\{1,2\}}^{tot}*\right) \tag{5.16}$$

(5.16) holds by Lemma 5.4. Multiply (5.16) by $\sum_{i\in\{1,2\}} \lambda \cdot c^{emer}$, then add $h \cdot S_{\{1,2\}}^{tot}*$, then multiply by 1.5 and re-arrange terms to obtain:

$$\begin{aligned} &\frac{1}{2} \cdot \left(h \cdot S^- + \pi_0\left(\frac{3\lambda}{\mu}, S^-\right) \cdot \sum_{i\in N} \lambda \cdot c^{emer}\right) + \frac{1}{2} \cdot \left(h \cdot S^+ + \pi_0\left(\frac{3\lambda}{\mu}, S^+\right) \cdot \sum_{i\in N} \lambda \cdot c^{emer}\right) \\ &\leq 1.5 \cdot \left(h \cdot S_{\{1,2\}}^{tot}* + \pi_0\left(\frac{2\lambda}{\mu}, S_{\{1,2\}}^{tot}*\right) \cdot \sum_{i\in\{1,2\}} \lambda \cdot c^{emer}\right) \end{aligned} \tag{5.17}$$

(5.17) is equivalent to $Q = \frac{1}{2} \cdot \mathrm{K}_{tot}^{\varphi_{simple,id:all,|N|=3;S^-}}(N) + \frac{1}{2} \cdot \mathrm{K}_{tot}^{\varphi_{simple,id:all,|N|=3;S^+}}(N) \leq 1.5 \cdot c(\{1,2\})$

If $\frac{1}{2} \cdot \mathrm{K}_{tot}^{\varphi_{simple,id:all,|N|=3;S^-}}(N) \leq \frac{1}{2} \cdot \mathrm{K}_{tot}^{\varphi_{simple,id:all,|N|=3;S^+}}(N)$, then $\mathrm{K}_{tot}^{\varphi_{simple,id:all,|N|=3;S^-}}(N) \leq Q$.

If $\frac{1}{2} \cdot \mathrm{K}_{tot}^{\varphi_{simple,id:all,|N|=3;S^-}}(N) \geq \frac{1}{2} \cdot \mathrm{K}_{tot}^{\varphi_{simple,id:all,|N|=3;S^+}}(N)$, then $\mathrm{K}_{tot}^{\varphi_{simple,id:all,|N|=3;S^+}}(N) \leq Q$.

$c(N) \leq \mathrm{K}_{tot}^{\varphi_{simple,id:all,|N|=3;S^-}}(N)$ and $c(N) \leq \mathrm{K}_{tot}^{\varphi_{simple,id:all,|N|=3;S^+}}(N)$. Thus, $c(N) \leq Q \leq 1.5 \cdot c(\{1,2\})$.

We show **(3)** by using $c(\{1,2\}) \leq c(\{1\}) + c(\{2\}) = 2 \cdot c(\{3\})$ (by Lemma 5.3 and due to the property that all companies are identical). Therefore, $1.5 \cdot c(\{1,2\}) \leq c(\{1,2\}) + c(\{3\})$. We just showed for **(2)** that $c(\{1,2,3\}) \leq 1.5 \cdot c(\{1,2\})$. Hence, $c(\{1,2,3\}) \leq c(\{1,2\}) + c(\{3\})$. This completes the proof.
■

5.3: Examples of games with empty cores

Now, by Theorems 5.1, 5.2, and 5.3 we have shown that games associated with certain classes of simple spare parts pooling situations and certain rules on base-stock levels always have a non-empty core. However, this does not hold for all simple spare parts pooling games. In this section, we provide three counter-examples that show that games associated with certain classes of simple spare parts pooling situations and certain rules on base-stock levels do not always have a non-empty core. After these examples are given, an explanation on why their cores are empty is provided.

Example 5.1: A game associated with a spare parts inventory situation in $\Gamma_{\text{simple,id:h}}$ and fixed identical base-stock levels can have an empty core.
Consider the 2-player simple spare parts inventory situation $\varphi_1 \in \Gamma_{simple,id:h}$ with N={1,2}, μ=0.03, h=0; c_1^{emer}=100, c_2^{emer}=0; λ_1=0.01, λ_2=1. So company 1 has very high emergency costs while company 2 has very low emergency costs, while the demand rate of company 2 is much higher. Suppose that we have situation FIX with base-stock vector $\boldsymbol{S}$ given by S_1=5, S_2=5. The associated spare parts pooling game is described by (values rounded):
c({1})=$2.46 \cdot 10^{-05}$; c({2})=0; c({1,2})=0.71.
Clearly, the core of this game is empty.

Example 5.2: A game associated with a spare parts inventory situation in $\Gamma_{\text{simple,id:}\lambda\text{,h}}$ and fixed different base-stock levels can have an empty core.
Consider the 2-player simple spare parts inventory situation $\varphi_2 \in \Gamma_{simple,id:\lambda,h}$ with N={1,2}, μ=0.03, h=0; c_1^{emer}=100, c_2^{emer}=0; λ=1. So company 1 has very high emergency costs while company 2 has very low emergency costs. Suppose that we have situation FIX with base-stock vector $\boldsymbol{S}$ given by S_1=25, S_2=1. The associated simple spare parts pooling game is described by (values are rounded to two decimals):
c({1})=30.63; c({2})=0; c({1,2})=61.89.
Clearly, the core of this game is empty.

Example 5.3: A game associated with a spare parts inventory situation in $\Gamma_{\text{simple,id:}\lambda\text{,h}}$ and to-be-optimized base-stock levels can have an empty core.
Consider the 2-player simple spare parts inventory situation $\varphi_3 \in \Gamma_{simple,id:\lambda,h}$ with N={1,2}, μ=5; h=25000; c_1^{emer}=120000, c_2^{emer}=3000; λ=2.5. So company 1 has very high emergency costs while company 2 has very low emergency costs. Suppose that we have situation OPT, so base-stock vectors are to-be-optimized. The associated simple spare parts pooling game is described by (values are rounded to two decimals):

c({1}) =73,076.92 ($S_{\{1\}}^{tot}$ *=2);

c({2}) =7,500.00 ($S_{\{2\}}^{tot}$ *=0);

c({1,2}) =94,218.75 ($S_{\{1,2\}}^{tot}$ *=3).

Clearly, the core of this game is empty.

An intuitive explanation behind these counter-examples would be as follows. Company 2 has very low emergency costs and hence very low total costs by itself. But its demand rate is at least as high as the demand rate of company 1. If we combine both companies in a coalition, then company 2 adds a lot of demand strain to the pooling stock, while it does not face high emergency costs. Every time a demand for company 2 comes in and it is fulfilled from the pooling stock, this takes away stock which company 1 – with very high downtime costs – could put to better use. It would actually be better to not use lateral transshipments at all. So, the main problem in these examples lies in the full pooling approach that is assumed.

5.4: Numerical experiment

We now know that for certain classes of simple spare parts pooling games it is possible that the core of the associated game is empty. But how often does this happen? Do we only encounter empty cores when we choose extreme and unrealistic parameter values? Or will games with non-empty cores also exist for cases with parameters that are representative of real-life cases? In this section, we present a numerical experiment that aims to answer these questions. The calculations are carried out by a Java program, which is described in Appendix 8.

5.4.1 Setup of the numerical experiment: spare parts inventory situations

Recall that a simple spare parts inventory situation is uniquely characterized by the set of companies, the repair rate, the demand rates of each company, the holding costs and the emergency (shipment and downtime) costs of each company. Table 5.1 shows all attainable parameter values in the first part of the experiment. For the number of companies, we use two values: three companies and four companies. A three-company situation is big enough to study the game theoretical nuances, and a four-company situation can provide insight into a larger cooperation. We remark that we will study two-company situations later in Section 5.4.5, but we focus on 3 and 4 players first. For the other input parameters, Table 5.1 only shows an index. Each index (like All-Min and DIFF1) refers to a “rule” on how the values of that parameter are set for each company. Table 5.2 shows for each index the corresponding “rule” on the parameter values of each company. This “rule” sets the parameter value of each company to either a minimum, low, standard, high, or maximum value. For each input parameter, the actual values corresponding to minimum, low, standard, high, and maximum are given in Table 5.3.

Table 5.1: All attainable parameter values in the experiment.

Parameter name and unit	Number of values
|N|	2 (three, four)
λ (demands per year)	8 (All-Min, All-Low, All-Standard, All-High, All-Max, DIFF1, DIFF2, DIFF3)
μ (repairs per year)	5 (All-Min, All-Low, All-Standard, All-High, All-Max)
h ($ per unit per year)	8 (All-Min, All-Low, All-Standard, All-High, All-Max, DIFF1, DIFF2, DIFF3)
c^{emer} ($ per emergency shipment)	8 (All-Min, All-Low, All-Standard, All-High, All-Max, DIFF1, DIFF2, DIFF3)

Table 5.2: For each index, the "rules" on how the values of a parameter are set for each company. When *N*={1,2,3}, only the first three company values are used.

Index	Value for company 1	Value for company 2	Value for company 3	Value for company 4 (if needed)
All-Min	Minimum value	Minimum value	Minimum value	Minimum value
All-Low	Low value	Low value	Low value	Low value
All-Standard	Standard value	Standard value	Standard value	Standard value
All-High	High value	High value	High value	High value
All-Max	Maximum value	Maximum value	Maximum value	Maximum value
DIFF1	Minimum value	Standard value	Maximum value	High value
DIFF2	High value	Low value	Standard value	Standard value
DIFF3	Low value	Maximum value	Low value	Low value

Table 5.3: For each input parameter, the actual values corresponding to minimum, low, standard, high, and maximum.

Parameter name and unit	Minimum value	Low value	Standard value	High value	Maximum value
λ (demands per year)	0.5	2.5	5	10	50
μ (repairs per year)	1.67	12.5	25	50	500
h ($ per unit per year)	400	2000	4000	8000	28000
c^{emer} ($ per emergency shipment)	2600	6500	13000	26000	78000

For each of the input parameters covered in Table 5.3, the minimum and maximum values are based on the literature study that was presented in section 3.4. We selected the "standard" values such that they are approximately the same orders of magnitude away from the minimum and maximum values.[15] We selected a "low" value to be half the standard value, and a "high" value to be twice the standard value. As such, DIFF2 should give realistic differences of a factor 4 amongst companies (realistic according to the papers covered in Appendix 5). DIFF1 gives extremely large differences amongst companies (although parameter values are still within the reasonable bounds of Table 3.1), and DIFF3 has the special property that at least two companies still have the same parameter value (with just one company being vastly different from the others). With all the combinations of the parameters described in Table 5.1, we have in total 2x8x5x8x8=5120 spare part inventory situations.

[15] That is, (the standard value) / (the minimum value) should be approximately the same as (the maximum value) / (the standard value).

One of these will serve as an example to illustrate the naming conventions: a situation where $|N|$=3, λ has value All-Low (hence λ_1=2.5, λ_2=2.5, λ_3=2.5), μ has value All-Min (hence μ_1=1.67, μ_2=1.67, μ_3=1.67), h has value DIFF2 (hence h_1=8000, h_2=2000, h_3=4000), and c^{emer} has value DIFF3 (hence c_1^{emer} =6500, c_2^{emer} =78000, c_3^{emer} =6500).[16]

5.4.2 Setup of the numerical experiment: rule on base-stock levels

Recall that with one simple spare parts inventory situation we can associate one simple spare parts pooling game (with to-be-optimized base-stock levels). Furthermore, with the combination of this simple spare parts inventory situation and some base-stock vector $\boldsymbol{S}$ we can associate another simple spare parts pooling game (with fixed base-stock levels). In this numerical experiment we use five types of base-stock vectors for the situation with fixed base-stock levels. In their definition, we use $S_{SUM} = \sum_{i\in N}\left(S_N^*\right)_i$.[17]

- $\boldsymbol{S}_N^*$: The base-stock level of each company is equal to the optimal cost-minimizing base-stock level when it is in the grand coalition.
- $\boldsymbol{S}^{\text{indiv}}$: The base-stock level of each company is equal to the optimal base-stock level when it would be acting alone; $S_i^{indiv} = \left(S_{\{i\}}^*\right)_i$ for all $i\in N$.
- $\boldsymbol{S}^{\text{high}}$: The base-stock level of each company is equal to $S_{SUM}/|N|$, rounded down, then plus two; $S_i^{high} = \lfloor S_{SUM}/|N|\rfloor + 2$ for all $i\in N$.
- $\boldsymbol{S}^{\text{low}}$: The base-stock level of each company is equal to $S_{SUM}/|N|$, rounded down, then minus one if possible; $S_i^{low} = \max\left(0, \lfloor S_{SUM}/|N|\rfloor - 1\right)$ for all $i\in N$.
- $\boldsymbol{S}^{\text{mix}}$: $S_1^{mix} = S_1^{indiv}$; $S_2^{mix} = S_2^{high}$; $S_3^{mix} = S_3^{low}$; and if $|N|$=4 then $S_4^{mix} = S_4^{indiv}$.

Note that $\boldsymbol{S}^{\text{indiv}}$ and $\boldsymbol{S}_N^*$ and $\boldsymbol{S}^{\text{mix}}$ can yield different base-stock levels amongst companies, whereas in the other two types of base-stock vectors all companies have identical base-stock levels. We remark that each of the 5120 spare parts inventory situations can be associated with one game with to-be-optimized base-stock levels and five games with fixed base-stock levels.

5.4.3 Results of the numerical experiment: Simple pooling games with to-be-optimized base-stock levels

Out of 5120 cases, 5018 had non-empty cores (98.0%), with 97.9% for 3-player games and 98.1% for 4-player games. Every game associated with a simple spare parts inventory situation for which all companies had identical emergency costs (i.e., one in $\Gamma_{simple,id:c^{emer}}$) had a non-empty core. This is in line with Theorem 5.3.

[16] We remark that the company having the highest value is a different one for DIFF1, DIFF2, and DIFF3, i.e. in this example company 1 has the highest holding cost rate while company 2 has the highest emergency costs. This hopefully allows more interaction and different results between situations.

[17] See Section 4.5 and/or Appendix 8 for more detail on how the unique optimal base-stock vector for coalition $M\subseteq N$, $\boldsymbol{S}_M^*$, is found.

Furthermore, interestingly, every game associated with a simple spare parts inventory situation for which all companies were not only allowed to have different emergency costs, different holding costs, and different demand rates but forced to be non-identical in all these respects (i.e., one where the index of λ, h, and c^{emer} was DIFF1, DIFF2, or DIFF3) had a non-empty core as well[18]. Only when we set companies to be identical on at least either λ or h and force their c^{emer} to be different, do we encounter associated games with empty cores. Results for these types of spare parts inventory situations are shown in Table 5.4. Figure 5.1 through Figure 5.4 show the percentage of associated games with empty cores for various input parameter values. A discussion and explanation of the pivotal observations is given at the end of Section 5.4.4.

Table 5.4: Percentage of associated games with empty cores differentiated for classes of simple spare parts inventory situations

Parameters that are set to be different (others are identical)	Corresponding parameter indexes	Percentage of associated games with empty cores	Percentage of associated games with empty imputation set
c^{emer}	$c^{emer}\in${DIFF1,DIFF2,DIFF3}; $\lambda,\mu,h\in${All-Min, All-Low, All-Standard, All-High, All-Max }	8.93%	4.67%
c^{emer} and λ	$c^{emer},\lambda\in${DIFF1,DIFF2,DIFF3}; $\mu,h\in${All-Min, All-Low, All-Standard, All-High, All-Max }	7.33%	3.33%
c^{emer} and h	$c^{emer},h\in${DIFF1,DIFF2,DIFF3}; $\mu,\lambda\in${All-Min, All-Low, All-Standard, All-High, All-Max }	0.44%	0.00%

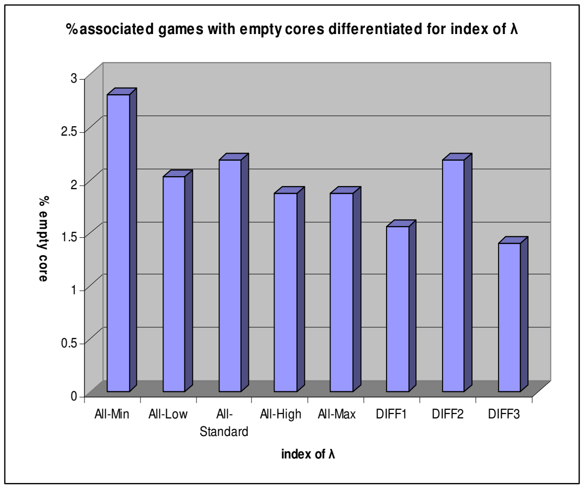

Figure 5.1: Results for each value index of λ.

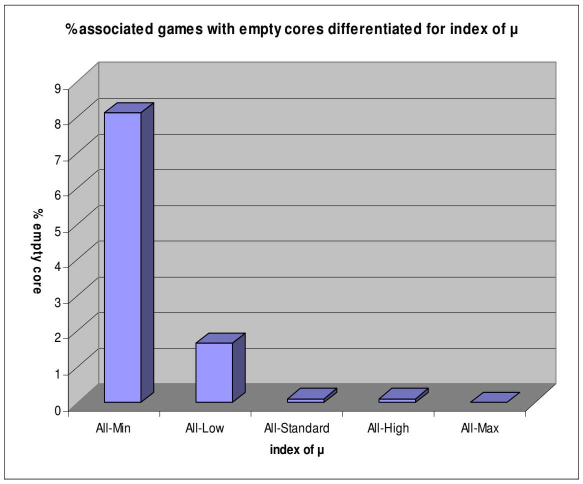

Figure 5.2: Results for each value index of μ.

[18] Note that there were only 3^3 games for which $c^{emer},h,\lambda\in${DIFF1,DIFF2,DIFF3}, so the sample size on which this statement is based was fairly small. As such, not finding any empty cores for this subset may be just due to the specific parameter values chosen. No conjecture on some generic property is implied.

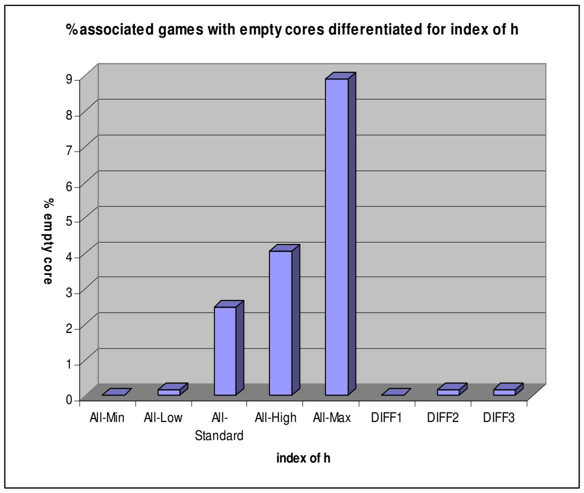

Figure 5.3: Results for each value index of *h*.

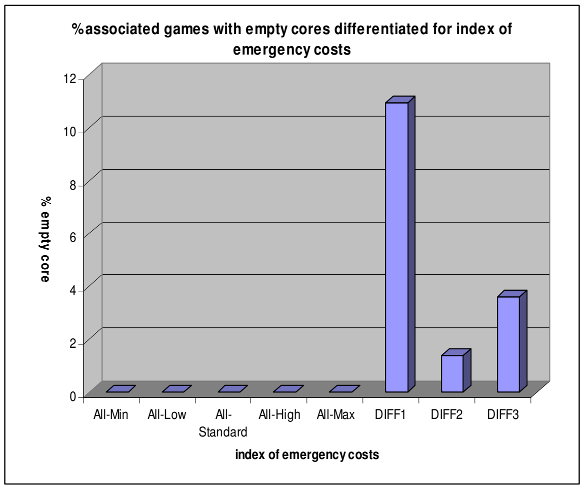

Figure 5.4: Results for each value index of c^{emer}.

5.4.4 Results of the numerical experiment: Simple pooling games with fixed base-stock levels

Out of 25600 cases, 25272 had non-empty cores (98.7%), with 98.6% for 3-player games and 98.8% for 4-player games. Every game associated with a simple spare parts inventory situation for which all companies had identical emergency costs (i.e., one in $\Gamma_{simple,id:c^{emer}}$) and any fixed base-stock vector had a non-empty core. Furthermore, every game associated with simple spare parts inventory situations for which companies had different emergency costs, identical λ *and* identical base-stock levels, had a non-empty core. Only when we set companies to be different on c^{emer} and at least different on λ or base-stock levels, do we encounter associated games with empty cores. This is in line with Theorems 5.1 and 5.2. Results for these types of spare parts inventory situations are shown in Table 5.5. Figure 5.5 through Figure 5.9 show the percentage of associated games with empty cores for various input parameter values. Afterwards, a discussion and explanation of the pivotal observations is provided.

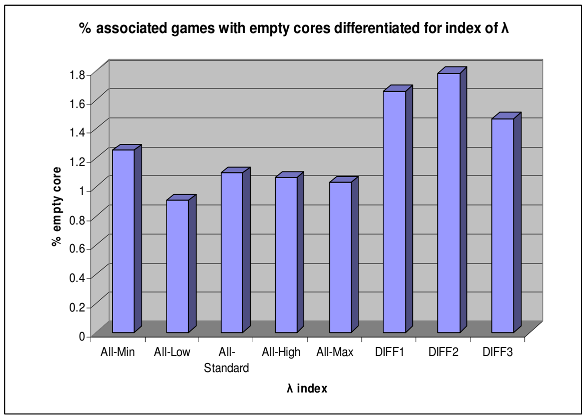

Figure 5.5: Results for each value index of λ.

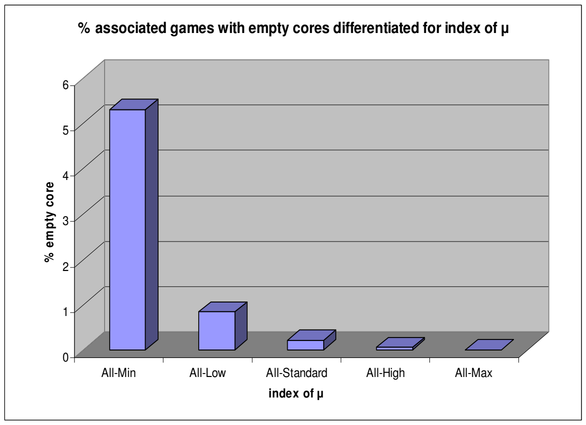

Figure 5.6: Results for each value index of μ.

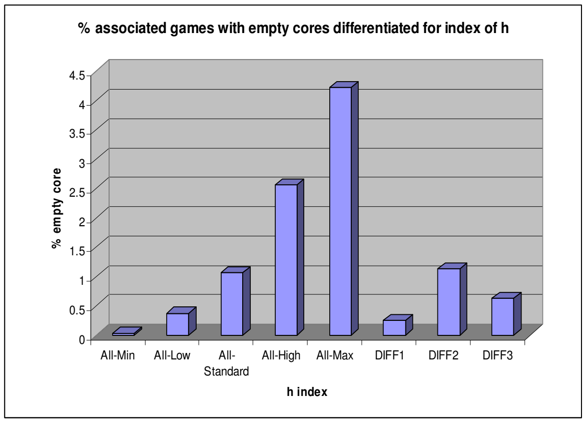

Figure 5.7: Results for each value index of *h*.

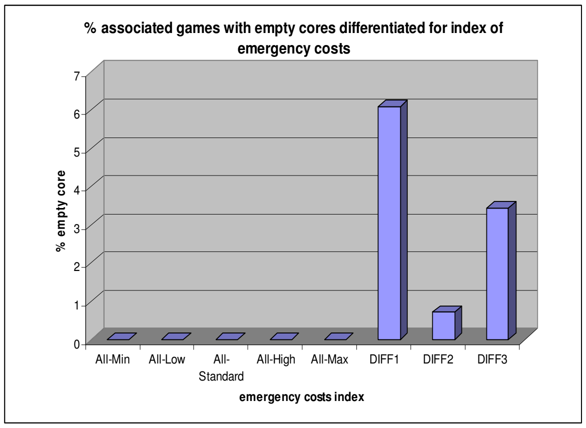

Figure 5.8: Results for each value index of c^{emer}.

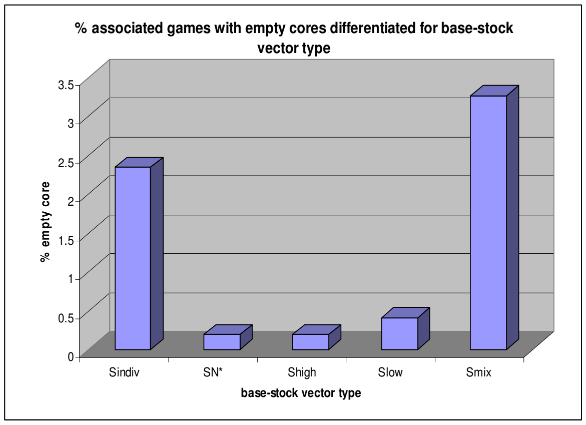

Figure 5.9: Results for each base-stock vector type

Table 5.5: Percentage of associated games with empty cores differentiated for classes of spare parts inventory situations and type of base-stock vectors

Parameters that are set to be different (others are identical)	Identical base-stock levels for all companies or possibly different?	Corresponding parameter indexes	Percentage of associated games with empty cores	Percentage of associated games with empty imputation set
c^{emer} and λ	Identical	$c^{emer},\lambda\in${DIFF1,DIFF2,DIFF3}; $\mu,h\in${All-Min, All-Low, All-Standard, All-High, All-Max }; $\boldsymbol{S}=\boldsymbol{S}^{\text{high}}$ or $\boldsymbol{S}=\boldsymbol{S}^{\text{low}}$	2.89 %	1.00 %
c^{emer} and λ and h	Identical	$c^{emer},\lambda,h\in${DIFF1,DIFF2,DIFF3}; $\mu\in${All-Min, All-Low, All-Standard, All-High, All-Max }; $\boldsymbol{S}=\boldsymbol{S}^{\text{high}}$ or $\boldsymbol{S}=\boldsymbol{S}^{\text{low}}$	0.93 %	0 %
c^{emer}	Can be different	$c^{emer}\in${DIFF1,DIFF2,DIFF3}; $\mu,h,\lambda\in${All-Min, All-Low, All-Standard, All-High, All-Max }; $\boldsymbol{S}=\boldsymbol{S}^{\text{indiv}}$ or $\boldsymbol{S}=\boldsymbol{S}_N$* or $\boldsymbol{S}=\boldsymbol{S}^{\text{mix}}$	6.22 %	3.64 %
c^{emer} and λ	Can be different	$c^{emer},\lambda\in${DIFF1,DIFF2,DIFF3}; $\mu,h\in${All-Min, All-Low, All-Standard, All-High, All-Max }; $\boldsymbol{S}=\boldsymbol{S}^{\text{indiv}}$ or $\boldsymbol{S}=\boldsymbol{S}_N$* or $\boldsymbol{S}=\boldsymbol{S}^{\text{mix}}$	7.26 %	2.59 %
c^{emer} and h	Can be different	$c^{emer},h\in${DIFF1,DIFF2,DIFF3}; $\mu,\lambda\in${All-Min, All-Low, All-Standard, All-High, All-Max }; $\boldsymbol{S}=\boldsymbol{S}^{\text{indiv}}$ or $\boldsymbol{S}=\boldsymbol{S}_N$* or $\boldsymbol{S}=\boldsymbol{S}^{\text{mix}}$	2.30 %	0.74 %
c^{emer} and λ and h	Can be different	$c^{emer},\lambda,h\in${DIFF1,DIFF2,DIFF3}; $\mu\in${All-Min, All-Low, All-Standard, All-High, All-Max }; $\boldsymbol{S}=\boldsymbol{S}^{\text{indiv}}$ or $\boldsymbol{S}=\boldsymbol{S}_N$* or $\boldsymbol{S}=\boldsymbol{S}^{\text{mix}}$	3.46 %	0.49 %

We can make several interesting observations in this numerical experiment that hold for both situations OPT and FIX:

- c^{emer} has a marked influence on whether the core of an associated game is empty or not (see Figure 5.4 and 5.8). If companies all have the exact same c^{emer}, then this numerical experiment suggests that the core of an associated game will be non-empty. This will later be captured formally in a conjecture. Empty cores are most often found for games associated with spare parts inventory situations in which the emergency costs differed largely between companies. The logic behind this has already been discussed in the counter-examples of Section 5.3; when demand for a company with a very low c^{emer} comes in, it is fulfilled from the pooling stock, but when subsequently demand for a company with a very high c^{emer} comes in, it may be that all available stock have been consumed by the company with the low c^{emer}. Once pooling stock gets low, fulfillling every demand of the company with the low c^{emer} via emergency shipment may give lower costs than full pooling. So, the main problem seems to lie in the full pooling approach that is assumed.
- The demand rate does not appear to be an important factor in whether we get an empty or non-empty core; Figure 5.1 and Figure 5.5 do not show a large effect.
- An empty core does not always imply an empty imputation set (see Table 5.4 and 5.5). Therefore, the game theoretical view does add something interesting.
- The games associated with simple spare parts inventory situations for which companies have identical, but very low repair rates and/or identical, but very high holding cost rates surprisingly often have empty cores. An intuitive explanation for this is that in these situations, the marginal added value of putting another spare part on stock is low. You have to pay lots of holding costs and/or the part is in repair for most of the time. This means that emergency costs will likely be dominant in the cost function, which amplifies the effect that c^{emer} has (as described in the first point in this list). This concept is illustrated in more detail in example games in Appendix 9. There, a game is given with an empty core and it is shown that setting the repair rate to be very high or setting the holding cost rate to be very low instead leads to a game with a non-empty core (and the holding costs are dominant in the latter two).

The following observation only applies to situation OPT:

- Consider the classes $\Gamma_{simple,id:h}$ and Γ_{simple}. This numerical experiment indicates that the former has more associated games with an empty core than the latter. An intuitive explanation of why setting holding cost rates to be different can lead to a balanced game is that this adds an additional way of obtaining cost savings. For example, if $h_1 > h_2$ then putting companies 1 and 2 together in a coalition implies that the spare parts needed for company 1 can now be stored more cheaply at company 2. If h_1 would have been the same as h_2, there would be no such benefit of cooperation.

The following observation only applies to situation FIX:

- Games for which companies had *non-identical* base-stock levels had more empty cores than games for which companies had *identical* base-stock levels. Particularly, $\boldsymbol{S}^{\mathrm{indiv}}$ and $\boldsymbol{S}^{\mathrm{mix}}$ relatively often led to empty cores. Finding a clear explanation for this is left as a future research direction.

5.4.5: Results of the numerical experiment: 2-player games

Recall that in section 5.3 we gave counter-examples of 2-player games. We now focus on sub-games of the 3-player games considered in the numerical experiment.
With each 3-player game, we can limit the characteristic cost function to player sets {1,2}, {1,3}, and {2,3} to construct 2-player sub-games. Two interesting questions arise:

- If the core of the three-player game is empty, then will we always find a two-player sub-game whose core is also empty? In other words, if things go wrong for three players, then is this always due to a problem that exists for two players?
- If the core of the three-player game is non-empty, then is it possible to find a two-player sub-game whose core is empty? If this happens, then the three-player game is not totally balanced.

The results of the numerical experiment with respect to these 2-player games are given for situation OPT in Table 5.6 and for situation FIX in Table 5.7. Interestingly, no game was found for which all three sub-games had an empty core. Furthermore, for both situations OPT and FIX, it was most likely that if the core of the 3-player game was empty that 2 sub-games had an empty core, and that if the core of the 3-player game was non-empty that 0 sub-games had an empty core.

Table 5.6: Results for 2-player games in situation OPT. The value in the table is the percentage of 3-player games with empty core (column 1) or non-empty core (column 2) with the number of sub-games with an empty core depending on the row.

	Core of the 3-player game is empty	Core of the 3-player game is non-empty
3 sub-games had an empty core	0%	0%
2 sub-games had an empty core	77.77%	0.04%
1 sub-games had an empty core	20.37%	0.60%
0 sub-games had an empty core	1.85%	99.36%

Table 5.7: Results for 2-player games in situation FIX. The value in the table is the percentage of 3-player games with empty core (column 1) or non-empty core (column 2) with the number of sub-games with an empty core depending on the row.

	Core of the 3-player game is empty	Core of the 3-player game is non-empty
3 sub-games had an empty core	0%	0%
2 sub-games had an empty core	44.44%	0.05%
1 sub-games had an empty core	39.44%	0.55%
0 sub-games had an empty core	16.11%	99.40%

In Appendix 9, three examples can be found: one where the core of the 3-player game was empty but none of the sub-games had empty cores; one where the core of the 3-player game was non-empty but two sub-games had empty cores; and one where the core of the 3-player game was empty but two sub-games also had empty cores.

This 2-player sub-game research showed that if the core of the three-player game is empty, then we will *not* always find a two-player sub-game whose core is also empty, i.e. if things go wrong for three players, then this is *not* always already due to a problem that exists for two players (although this is often the case). Furthermore, even if the core of the three-player game is non-empty, the three-player game is *not* always totally balanced.

5.5 Chapter summary

In this chapter, we have seen proofs that games associated with certain classes of spare parts inventory situations will always have non-empty cores. We have also seen counterexamples showing that games associated with certain classes of simple spare parts inventory situations will *not* always have non-empty cores.

A numerical experiment has led to interesting observations. One of those will now be formally captured in a conjecture. Based on the observations in this numerical experiment, we conjecture that any simple spare parts pooling game (situation FIX or situation OPT), associated with any simple spare parts inventory situation in which companies have identical emergency costs, has a non-empty core.

Conjecture 5.1: Let $\varphi \in \Gamma_{simple,id:c^{emer}}$. Let $\boldsymbol{S} \in \mathbb{N}_0^{\,N}$ be fixed. Then:

(i) The spare parts pooling game (N,c) associated with φ and $\boldsymbol{S}$ has a non-empty core.

(ii) The spare parts pooling game (N,c) associated with φ also has a non-empty core.

We now provide a structured overview of what we know for games associated with each class of spare parts inventory situation in Tables 5.8 (for situations with to-be-optimized base stock levels) and 5.9 (for situations with fixed base-stock vectors).

Table 5.8: Summary of what we know on the cores of simple spare parts pooling games for situation OPT

Class of spare parts inventory situation	Parameters that may be different between companies	What do we know about the core of the associated game?
$\Gamma_{simple,id:\lambda,h,c^{emer}}$	-	If $\lvert N\rvert=3$ and $S_M^{tot} \leq 20 \cdot \lvert M \rvert$, core is non-empty (Theorem 5.3). Else, conjecture 5.1 applies
$\Gamma_{simple,id:h,c^{emer}}$	Demand rates	Conjectured to be non-empty (Conjecture 5.1)
$\Gamma_{simple,id:\lambda,c^{emer}}$	Holding cost rates	Conjectured to be non-empty (Conjecture 5.1)
$\Gamma_{simple,id:c^{emer}}$	Demand rates, holding cost rates	Conjectured to be non-empty (Conjecture 5.1)
$\Gamma_{simple,id:\lambda,h}$	Emergency costs	Counter-example of empty core 5.3
$\Gamma_{simple,id:h}$	Emergency costs, demand rates	Counter-example of empty core 5.3
$\Gamma_{simple,id:\lambda}$	Emergency costs, holding cost rates	Counter-example of empty core 5.3
Γ_{simple}	Emergency costs, demand rates, holding cost rates	Counter-example of empty core 5.3

Table 5.9: Summary of what we know on the cores of spare parts pooling games for situation FIX[19]

Class of spare parts pooling game	Identical base-stock levels for all companies or possibly different?	What do we know about the core of the associated game?
$\Gamma_{simple,id:\lambda,h,c^{emer}}$	Identical	Non-empty (Theorem 5.1 and 5.2)
$\Gamma_{simple,id:h,c^{emer}}$	Identical	Conjectured to be non-empty (Conjecture 5.1)
$\Gamma_{simple,id:\lambda,h,c^{emer}}$	Different	Non-empty (Theorem 5.2)
$\Gamma_{simple,id:h,c^{emer}}$	Different	Conjectured to be non-empty (Conjecture 5.1)
$\Gamma_{simple,id:\lambda,h}$	Identical	Non-empty (Theorem 5.1)
$\Gamma_{simple,id:h}$	Identical	Counter-example of empty core 5.1
$\Gamma_{simple,id:\lambda,h}$	Different	Counter-example of empty core 5.2
$\Gamma_{simple,id:h}$	Different	Counter-example of empty core 5.1 or 5.2

[19] Note that the results for any spare parts inventory situation in which *h* is indentical will be the same as any spare parts inventory situation in which *h* is different between companies. It can be easily verified that holding cost terms will always cancel out against each other in all balancedness equations and hence do not affect whether a core is empty or not. The reason why e.g. figure 5.7 shows different results for different indexes of *h* is that the base-stock vectors chosen are based on *h* and we compare S(*h*), *h* combinations with S(*h*'),*h*' combinations rather than *S*(*h*),*h* combinations with *S*(*h*),*h*' combinations.

Chapter 6: Cost allocation in simple spare parts pooling games

In this chapter we attempt to answer research question 2 ("What is a proper cost allocation policy for a simple spare parts pooling game?"). In section 6.1, we first propose four cost allocations. In section 6.2, we define useful properties and see whether these cost allocations adhere to it. In section 6.3, we perform a numerical experiment on cost allocations and draw some conclusions regarding stability of allocation rules.

6.1. Proposed cost allocations

In this section, we define four cost allocations that will form the basis of this chapter. These cost allocations are taken from a much larger list of potential cost allocations that originated from the literature or from a brainstorm session. All of these cost allocations can be found in Appendix 10. The four cost allocations that will be presented now were selected mainly because they performed well in a numerical experiment, i.e. in a relatively large percentage of games these allocations were in the core. Some results of this numerical experiment will be shown in Section 6.3. While the idea behind all four cost allocations can be applied both to games with to-be-optimized base-stock levels and to games with fixed base-stock levels, the formulas and behavior can differ. Therefore, in definitions in this chapter we make a distinction between situations OPT and FIX.
For situation OPT, we have an allocation rule defined as a function $f^{OPT} : \Gamma \to \mathbb{R}^N$.
For situation FIX, we have an allocation rule defined as a function $f^{FIX} : \Gamma * \mathbb{N}^N \to \mathbb{R}^N$.
Finally, we remark that cost allocations allocate *expected* yearly costs, not *realized* yearly costs (as the characteristic cost function is also defined as *expected* yearly costs).

Allocation rule AL: An allocation of total costs based on the demand rate of each company.
<u>Formula for situation FIX</u>: Let $\varphi \in \Gamma_{simple}$ and $\boldsymbol{S} \in \mathbb{N}_0^N$. Then, for all $i \in N$:

$$AL_i^{FIX}(\varphi, \boldsymbol{S}) = K^{\varphi,S}(N) \cdot \lambda_i / \sum_{j \in N} \lambda_j \quad .$$

<u>Formula for situation OPT</u>: Let $\varphi \in \Gamma_{simple}$. Then, for all $i \in N$:

$$AL_i^{OPT}(\varphi) = AL_i^{FIX}(\varphi, \boldsymbol{S}^*_N(\varphi)).$$

This is cost allocation policy 3 in Wong et al. (2007) and is quite easy to understand and administer. Simplicity of a cost allocation rule is somewhat subjective, but we suggest that this allocation rule is the simplest of all rules presented here. The demand rate turned out to be the most important information for cost allocations; in the numerical experiment, allocating based on demand rates gave more core elements than allocating based on holding cost rates or emergency costs.

Allocation rule SPLIT: An allocation where holding costs are allocated based on demand rates and emergency costs based on the demand rate times emergency costs.
<u>Formula for situation FIX</u>: Let $\varphi \in \Gamma_{simple}$ and $S \in \mathbb{N}_0^N$. Then, for all $i \in N$:

$$SPLIT_i^{FIX}(\varphi, S) = \frac{\lambda_i}{\sum_{j \in N} \lambda_j} \cdot \left(\sum_{j \in N} h_j \cdot S_j \right) + \pi_0\left(\frac{\sum_{j \in N} \lambda_j}{\mu}, \sum_{j \in N} S_j\right) \cdot \lambda_i \cdot c_i^{emer}.$$

<u>Formula for situation OPT</u>: Let $\varphi \in \Gamma_{simple}$. Then, for all $i \in N$:

$$SPLIT_i^{OPT}(\varphi) = SPLIT_i^{FIX}(\varphi, S_N^*(\varphi)).$$

This allocation is the result of a brainstorm experiment (Appendix 10) in which many similar allocations were tried, but this particular allocation turned out to be in the core most often. The amount of spare parts that has to be held on stock for a company is based on his demand rate, so allocating holding costs based on demand rate makes sense intuitively (and allocating costs this way gave more core elements than allocating holding costs based on holding cost rates in the numerical experiment). The total emergency costs is dependent on $\lambda_i \cdot c_i^{emer}$ and therefore this term is used.

Allocation rule BL: An allocation of total benefits based on the demand rate of each company:
<u>Formula for situation FIX</u>: Let $\varphi \in \Gamma_{simple}$ and $S \in \mathbb{N}_0^N$. Then, for all $i \in N$:

$$BL_i^{FIX}(\varphi, S) = \mathrm{K}^{\varphi;S}(\{i\}) - \frac{\lambda_i}{\sum_{j \in N} \lambda_j} \cdot \left(\sum_{j \in N} \mathrm{K}^{\varphi;S}(\{j\}) - \mathrm{K}^{\varphi;S}(N) \right).$$

<u>Formula for situation OPT</u>: Let $\varphi \in \Gamma_{simple}$. Then, for all $i \in N$:

$$BL_i^{OPT}(\varphi) = \mathrm{K}^{\varphi;S_{\{i\}}^*(\varphi)}(\{i\}) - \frac{\lambda_i}{\sum_{j \in N} \lambda_j} \cdot \left(\sum_{j \in N} \mathrm{K}^{\varphi;S_{\{j\}}^*(\varphi)}(\{j\}) - \mathrm{K}^{\varphi;S_N^*(\varphi)}(N) \right).$$

Based on Kilpi et al. (2008), allocating benefits ensures that (if the imputation set is non-empty) no individual company will have to pay more costs than when working alone.

The Shapley value:
<u>Formula for situation FIX</u>: Let $\varphi \in \Gamma_{simple}$ and $S \in \mathbb{N}_0^N$. Then, for all $i \in N$:

$$\Phi_i(\varphi, S) = \sum_{M \subseteq N \setminus \{i\}} \frac{|M|! \cdot (|N - M| - 1)!}{|N|!} \cdot \left(\mathrm{K}^{\varphi;S}(M \cup \{i\}) - \mathrm{K}^{\varphi;S}(M) \right).$$

<u>Formula for situation OPT</u>: Let $\varphi \in \Gamma_{simple}$. Then, for all $i \in N$:

$$\Phi_i(\varphi) = \sum_{M \subseteq N \setminus \{i\}} \frac{|M|! \cdot (|N - M| - 1)!}{|N|!} \cdot \left(\mathrm{K}^{\varphi;S_{M \cup \{i\}}^*(\varphi)}(M \cup \{i\}) - \mathrm{K}^{\varphi;S_M^*(\varphi)}(M) \right).$$

This allocation is in "game terms" rather than in "spare parts inventory situation terms". The Shapley value is a well-established allocation in game theory literature, however.

In this chapter our focus is on trying to find a cost allocation rule that:
(i) is easy to grasp, easy to calculate, and applicable to spare parts pooling games
(ii) adheres to various fairness properties
(iii) is always in the core (if non-empty) of a simple spare parts pooling game.

We remark that we will not cover the nucleolus (see Appendix 2) here, although the nucleolus is always in the core (if the core is non-empty). However, it is relatively hard to calculate (Hartman&Dror, 1996; Sankaran, 1991). We posit that the four allocation rules presented in this section, in our opinion, do adhere to the above property (i), as opposed to the nucleolus. The nucleolus can act as a fall-back option if none of the four allocation rules proposed here can be shown to be in the core of a simple spare parts pooling game.

6.2. Properties of cost allocations

We now provide a list of properties that may be desirable for cost allocations to have. Efficiency and stability are clearly important and well-established in game theory literature; the set of all stable and efficient allocations is called the core. The other properties are fairness properties that are applicable to spare parts pooling games and that may help to select a fair cost allocation.

6.2.1 Efficiency

An allocation rule is efficient if all costs incurred are fully split. Formally:
<u>Definition for situation OPT:</u> An allocation rule f^{OPT} is efficient if for all $\varphi \in \Gamma_{simple}$:
$\sum_{i \in N} f_i^{OPT}(\varphi) = K^{\varphi, S_N^*(\varphi)}(N)$.
<u>Definition for situation FIX:</u> An allocation rule f^{FIX} is efficient if for all $\varphi \in \Gamma_{simple}$ and all $\boldsymbol{S} \in \mathbb{N}_0^N$: $\sum_{i \in N} f_i^{FIX}(\varphi, \boldsymbol{S}) = K^{\varphi, S}(N)$.

Lemma 6.1: Allocation rules AL^{FIX}, SPLIT^{FIX}, BL^{FIX}, Φ^{FIX}, AL^{OPT}, SPLIT^{OPT}, BL^{OPT}, and Φ^{OPT} are efficient.
Proof:
Let $\varphi \in \Gamma_{simple}$ and let $\boldsymbol{S} \in \mathbb{N}_0^N$.
We will first show efficiency for the Shapley value. We will use a result from the literature, but in order to be able to use it we need to transform the Shapley value defined as a function of a simple spare parts inventory situation, Φ_i, to the Shapley value defined as a function of a game, Φ_i^{game} (as defined in Section 1.4).
Let the game associated with φ be (N, c^{OPT}) and the game associated with φ and $\boldsymbol{S}$ be (N, c^{FIX}). Now, for $i \in N$: $\Phi_i^{game}(N, c^{OPT}) = \Phi_i^{OPT}(\varphi)$ and $\Phi_i^{game}(N, c^{FIX}) = \Phi_i^{FIX}(\varphi, \boldsymbol{S})$.

Since Φ_i^{game} satisfies efficiency in terms of a game (see Appendix 2), it follows that allocation rules Φ^{FIX} and Φ^{OPT} are efficient.
As for AL^{FIX}, it is easy to show directly in one step that $\sum_{i\in N} AL_i^{FIX}(\varphi,\boldsymbol{S}) = K^{\varphi,S}(N)$.
Furthermore: $\sum_{i\in N} SPLIT_i^{FIX}(\varphi,\boldsymbol{S}) = \sum_{i\in N} h_i \cdot S_i + \pi_0(\sum_{i\in N}\lambda_i/\mu, \sum_{i\in N} S_i)\cdot \sum_{i\in N}\lambda_i \cdot c_i^{emer} = K^{\varphi,S}(N)$.
Furthermore: $\sum_{i\in N} BL_i^{FIX}(\varphi,\boldsymbol{S}) = \sum_{i\in N} \mathrm{K}^{\varphi;S}(\{i\}) - \left(\sum_{i\in N} \mathrm{K}^{\varphi;S}(\{i\}) - \mathrm{K}^{\varphi;S}(N)\right) = \mathrm{K}^{\varphi;S}(N)$.
Using the above, we also have $AL_i^{OPT}(\varphi) = AL_i^{FIX}(\varphi, \boldsymbol{S}^*_N(\varphi)) = K^{\varphi,S_N^*(\varphi)}(N)$ and
$SPLIT_i^{OPT}(\varphi) = SPLIT_i^{FIX}(\varphi,\ \boldsymbol{S}^*_N(\varphi)) = K^{\varphi,S_N^*(\varphi)}(N)$.
Finally: $\sum_{i\in N} BL_i^{OPT}(\varphi) = \sum_{i\in N} \mathrm{K}^{\varphi;S^*_{\{i\}}(\varphi)}(\{i\}) - \left(\sum_{i\in N} \mathrm{K}^{\varphi;S^*_{\{i\}}(\varphi)}(\{i\}) - \mathrm{K}^{\varphi;S_N^*(\varphi)}(N)\right) = \mathrm{K}^{\varphi;S_N^*(\varphi)}(N)$
■

6.2.2 Stability

Definition for situation OPT: An allocation rule f^{OPT} is stable if for all $\varphi \in \Gamma_{simple}$:
$\sum_{i\in M} f_i^{OPT}(\varphi) \le K^{\varphi,S_M^*(\varphi)}(M)$ for all $M \subseteq N$.
Definition for situation FIX: An allocation rule f^{FIX} is stable if for all $\varphi \in \Gamma_{simple}$ and all $\boldsymbol{S} \in \mathbb{N}_0^N$: $\sum_{i\in M} f_i^{FIX}(\varphi,\boldsymbol{S}) \le K^{\varphi,S}(M)$ for all $M \subseteq N$.

We will investigate stability for games associated with certain classes of spare parts inventory situations in section 6.3.

6.2.3 Monotonicity (in *λ*, in *h*, and in *c^emer^*)

We shall only define Monotonicity in λ in order to be brief. Definitions for Monotonicity in h and Monotonicity in c^{emer} are obtained by replacing all instances of λ with h or c^{emer}, respectively in the following definition. Intuitively, an allocation rule is monotone in λ if, keeping everything else equal, an increase in the demand rate of a company does not result in a decrease of the costs allocated to him. For a single company, his cost function when acting alone is non-decreasing in λ, h, and c^{emer} and it seems reasonable to require the same for cost allocations in bigger coalitions. Formally:

Definition for situation OPT: Suppose $\varphi, \varphi' \in \Gamma_{simple}$ such that φ' is identical to φ in all respects, except that for some $i \in N$: λ_i is higher in φ' than λ_i in φ. Then, an allocation rule f^{OPT} is monotonic in λ if: $f_i^{OPT}(\varphi') \ge f_i^{OPT}(\varphi)$.

Definition for situation FIX: Suppose $\varphi, \varphi' \in \Gamma_{simple}$ such that φ' is identical to φ in all respects, except that for some $i \in N$: λ_i is higher in φ' than λ_i in φ. Let $\boldsymbol{S} \in \mathbb{N}_0^N$. Then, an allocation rule f^{FIX} is monotonic in λ if: $f_i^{FIX}(\varphi', \boldsymbol{S}) \geq f_i^{FIX}(\varphi, \boldsymbol{S})$.

Lemma 6.2: Allocation rules $\mathrm{AL}^{\mathrm{FIX}}$ and $\mathrm{AL}^{\mathrm{OPT}}$ are monotonic in λ, in h, and in c^{emer}.
Proof:
Let $\varphi \in \Gamma_{simple}$, let $M \subseteq N$, let $\boldsymbol{S} \in \mathbb{N}_0^M$, and let $i \in N$. Suppose $\varphi' \in \Gamma_{simple}$ such that φ' is identical to φ in all respects, except that either λ_i is higher in φ' than λ_i in φ. In the remainder of this proof, we will first show some properties about the characteristic cost functions and subsequently combine these to complete the proof.

Part 1: We will first show that $\mathrm{K}^{\varphi';S}(N) \geq \mathrm{K}^{\varphi;S}(N)$. $B = \pi_0\left(\sum_{i \in M} \lambda_i / \mu, \sum_{i \in M} S_i\right)$ is non-decreasing in λ_i by Lemma 4.1. Therefore, $\mathrm{K}^{\varphi;S}(M) = \sum_{i \in M} h_i \cdot S_i + B \cdot \sum_{i \in M} \lambda_i \cdot c_i^{emer}$ is easily seen to be non-decreasing in λ_i, hence $\mathrm{K}^{\varphi';S}(N) \geq \mathrm{K}^{\varphi;S}(N)$.

Part 2: We show $\mathrm{K}^{\varphi';S_N^*(\varphi')}(N) \geq \mathrm{K}^{\varphi;S_N^*(\varphi)}(N)$. We start with $\mathrm{K}^{\varphi';S_N^*(\varphi')}(N) \geq \mathrm{K}^{\varphi;S_N^*(\varphi')}(N)$, which holds by Part 1. Observe $\mathrm{K}^{\varphi;S_N^*(\varphi')}(N) \geq \mathrm{K}^{\varphi;S_N^*(\varphi)}(N)$, by definition of $S_N^*(\varphi)$ as a cost-minimizing base-stock vector. Hence, $\mathrm{K}^{\varphi';S_N^*(\varphi')}(N) \geq \mathrm{K}^{\varphi;S_N^*(\varphi)}(N)$.

Part 3: Obviously, $\lambda_i / \sum_{j \in N} \lambda_j$ is non-decreasing in λ_i.

Using the definitions of $AL_i^{FIX}(\varphi, \boldsymbol{S})$ and $AL_i^{OPT}(\varphi)$, we complete the proof with:
By part 1 and 3: $AL_i^{FIX}(\varphi', \boldsymbol{S}) \leq AL_i^{FIX}(\varphi, \boldsymbol{S})$. By part 2 and 3: $AL_i^{OPT}(\varphi') \geq AL_i^{OPT}(\varphi)$.
The proof for monotonicity in h and in c^{emer} goes analogously to the above.
■

We will now posit two conjectures on whether certain allocation rules adhere to monotonicity properties. These conjectures are based on a check added to all games of the numerical experiment of section 6.3, where for each spare parts inventory situations three additional spare parts inventory situations were created, in which either λ_1, h_1, or c_1^{emer}, respectively, were doubled and the cost allocations for these were compared to the cost allocations for the original ones. For allocations that adhered to monotonicity properties in all of these cases, the following conjectures are stated.

Conjecture 6.1: Allocation rules Φ^{OPT}, Φ^{FIX} and $\mathrm{SPLIT}^{\mathrm{FIX}}$ are monotonic in λ, in h and in c^{emer}.

Conjecture 6.2: Allocation rules $\mathrm{BL}^{\mathrm{OPT}}$ and $\mathrm{BL}^{\mathrm{FIX}}$ are monotonic in h and in c^{emer}.

We will now present an example (6.1) showing that allocation rule $\mathrm{SPLIT}^{\mathrm{OPT}}$ is not always monotonic in h.

The idea behind this example is that an increase in the holding costs of a company can result in a lower optimal base-stock level and hence both lower total holding costs and higher total emergency costs (as the probability of having a stock-out has increased). Since allocation rule SPLIT^{OPT} allocates holding costs and emergency costs in different fashion, it may be that one part decreases greatly while the other part increases only slightly. Subsequently, we present an example (6.2) showing that allocation rule SPLIT^{OPT} is not always monotonic in λ and an example (6.3) showing that allocation rule SPLIT^{OPT} is not always monotonic in c^{emer}. The idea behind these examples is very similar to the first one.

Example 6.1: Allocation rule SPLIT^{OPT} is not always monotonic in h

Consider the 2-player simple spare parts inventory situation $\varphi_1 \in \Gamma_{simple}$ with N={1,2}, μ=500, h_1=400; h_2=28000; c_1^{emer}=10, c_2^{emer}=50,000; λ_1=5, λ_2=0.05. Suppose that we have situation OPT. The associated simple spare parts pooling game is described by:

c({1}) = 50.0; c({2})= 2500.0; c({1,2}) = 425.5;

We remark that for $S_{\{1\}}^{tot}* = S_{\{2\}}^{tot}* = 0$ and $S_{\{1,2\}}^{tot}* = 1$. Allocation SPLIT^{OPT} (φ_1) results in

SPLIT$_1^{OPT}$ (φ_1)=396.54 and SPLIT$_2^{OPT}$ (φ_1)=28.96.

Now, consider the 2-player simple spare parts inventory situation $\varphi_2 \in \Gamma_{simple}$ which is identical to φ_1 except that the holding cost rate of company 1 has increased: h_1 is now 28000 instead. Suppose that we have situation OPT. The associated game is described by:

c({1}) = 50.0; c({2})= 2500.0; c({1,2}) = 2550.0;

We remark that $S_{\{1\}}^{tot}* = S_{\{2\}}^{tot}* = S_{\{1,2\}}^{tot}* = 0$. Allocation SPLIT^{OPT} (φ_2) results in

SPLIT$_1^{OPT}$ (φ_2)=50.0 and SPLIT$_2^{OPT}$ (φ_2)=2500.0.

As $SPLIT_1^{OPT}(\varphi_2) < SPLIT_1^{OPT}(\varphi_1)$, SPLIT^{OPT} does not always satisfy monotonicity in h.

Example 6.2: Allocation rule SPLIT^{OPT} is not always monotonic in λ

Consider the 2-player simple spare parts inventory situation $\varphi_3 \in \Gamma_{simple,id:h}$ with N={1,2}, μ=1.67, h=28000; c_1^{emer}=2600, c_2^{emer}=78000; λ_1=10, λ_2=5. Suppose that we have situation OPT. The associated simple spare parts pooling game is described by:

c({1}) = 26,000; c({2}) = 182,721.67; c({1,3}) = 343,904.73.

We remark that for $S_{\{1\}}^{tot}* = 0$, $S_{\{2\}}^{tot}* = 5$, $S_{\{1,2\}}^{tot}* = 8$. Allocation SPLIT^{OPT}(φ_3) results in

SPLIT$_1^{OPT}$ (φ_3)=156,827.28 and SPLIT$_2^{OPT}$ (φ_3)=187,077.35.

Now, consider the 2-player simple spare parts inventory situation $\varphi_4 \in \Gamma_{simple,id,h}$ which is identical to φ_3 except that the demand rate of company 1 has increased: λ_1 is now 20 instead. Suppose that we have situation OPT. The associated game is described by:

c({1}) =52,000; c({2}) = 182,721.67; c({1,3}) = 442,000.

We remark that for $S_{\{1\}}^{tot}* = S_{\{1,2\}}^{tot}* = 0$, $S_{\{2\}}^{tot}* = 5$. Allocation SPLIT^{OPT}(φ_4) results in

SPLIT$_1^{OPT}$ (φ_4)=52,000, SPLIT$_2^{OPT}$ (φ_4)=390,000.

As $SPLIT_1^{OPT}(\varphi_4) < SPLIT_1^{OPT}(\varphi_3)$, SPLIT^{OPT} does not always satisfy monotonicity in λ.

Example 6.3: Allocation rule SPLIT$^{\mathrm{OPT}}$ is not always monotonic in c^{emer}

Consider the 3-player simple spare parts inventory situation $\varphi_5 \in \Gamma_{simple,id:\lambda,h}$ with N={1,2,3}, μ=25, h=400; c_1^{emer}=26000, c_2^{emer}=6500; c_3^{emer}=13000; λ=2.5. Suppose that we have situation OPT. The associated simple spare parts pooling game is described by:

c({1}) = 1094.12;	c({2})= 873.53;	c({3}) = 947.06;
c({1,2}) = 1288.7;	c({1,3}) = 1306.44;	c({2,3}) = 1253.22;

c({1,2,3}) = 1579.31 (with $S_{\{1,2,3\}}^{tot}* = 3$).

We have SPLIT$_1^{OPT}$ (φ_5)=616.8; SPLIT$_2^{OPT}$ (φ_5)=454.2; SPLIT$_3^{OPT}$ (φ_5)=508.4.

Now, consider the 3-player simple spare parts inventory situation $\varphi_6 \in \Gamma_{simple,id:\lambda,h}$ which is identical to φ_5 except that the emergency costs of company 1 has increased: c_1^{emer} is now 52000 instead. Suppose that we have situation OPT. The associated simple spare parts pooling game is described by:

c({1}) = 1219.6;	c({2})= 873.53;	c({3}) = 947.06;
c({1,2}) = 1359.66;	c({1,3}) = 1377.4;	c({2,3}) = 1253.22;

c({1,2,3}) = 1644.69 (with $S_{\{1,2,3\}}^{tot}* = 4$).

We have SPLIT$_1^{OPT}$ (φ_6)=565.84; SPLIT$_2^{OPT}$ (φ_6)=537.40; SPLIT$_3^{OPT}$ (φ_6)=541.46.

As $SPLIT_1^{OPT}(\varphi_6) < SPLIT_1^{OPT}(\varphi_5)$, SPLIT$^{\mathrm{OPT}}$ does not satisfy monotonicity in c^{emer}.

We will now present an example (6.4) showing that allocation rule BL$^{\mathrm{OPT}}$ is not always monotonic in λ. The idea behind this example is that an increase in the demand rate of a company will result in higher costs of the grand coalition, but more benefits allocated to the company with the higher demand rate (hence it has to pay less costs). Subsequently, we present an example (6.5) showing that allocation rule BL$^{\mathrm{FIX}}$ is not always monotonic in λ. The idea behind this example is very similar to the previous one.
We remark that Allocation BL$^{\mathrm{OPT}}$ is not monotonic in λ while Allocation AL$^{\mathrm{OPT}}$ is monotonic in λ by Lemma 6.2. Therefore, these two rules can give different allocations. Hence, the method of allocating costs based on demand rates are different for a cost game and a benefit game and therefore allocations AL(φ) and BL(φ) do not adhere to the justifiability criterion of Hartman and Dror (1996).

Example 6.4: Allocation rule BL$^{\mathrm{OPT}}$ is not always monotonic in λ

Consider once again the 3-player simple spare parts inventory situation φ_5 (see Example 6.3). We have BL$_1^{OPT}$ (φ_5)=649.0, BL$_2^{OPT}$ (φ_5)=428.4, BL$_3^{OPT}$ (φ_5)=501.9.

Now, consider the 3-player simple spare parts inventory situation $\varphi_7 \in \Gamma_{simple,id:\lambda,h}$ which is identical to φ_5 except that the demand rate of company 1 has increased: λ_1 is now 5 instead. Suppose that we have situation OPT. The associated game is described by:

c({1}) = 1341.92;	c({2})= 873.53;	c({3}) = 947.06;
c({1,2}) = 1636.57;	c({1,3}) = 1640.63;	c({2,3}) = 1253.22;

c({1,2,3}) = 1727.82.

Allocation BL$^{\mathrm{OPT}}$(φ_7) results in BL$_1^{OPT}$ (φ_7)=624.6, BL$_2^{OPT}$ (φ_7)=514.9, BL$_3^{OPT}$ (φ_7)=588.4.

As $BL_1^{OPT}(\varphi_7) < BL_1^{OPT}(\varphi_5)$, BL$^{\mathrm{OPT}}$ does not always satisfy monotonicity in λ.

Example 6.5: Allocation rule $\mathrm{BL}^{\mathrm{FIX}}$ is not always monotonic in λ

Consider the 3-player simple spare parts inventory situation $\varphi_8 \in \Gamma_{simple,id:\lambda,h,c^{emer}}$ with $N=\{1,2,3\}$, $\mu=25$, $h=4000$; $c^{emer}=13000$; $\lambda=5$. Suppose that we have situation FIX with base-stock vector $\boldsymbol{S}$ given by $S_1=2$, $S_2=3$, $S_3=0$. The associated game is described by:

c({1}) = 9,065.57;	c({2})= 12,070.96;	c({3}) = 65,000.0;
c({1,2}) = 20,007.44;	c({1,3}) = 15,027.03;	c({2,3}) = 12,930.23;
c({1,2,3}) = 20,069.35		

We have $\mathrm{BL}_1^{FIX}(\varphi_8,\boldsymbol{S})=-12{,}956.82$, $\mathrm{BL}_2^{FIX}(\varphi_8,\boldsymbol{S})=-9{,}951.43$, $\mathrm{BL}_3^{FIX}(\varphi_8,\boldsymbol{S})=42{,}977.61$.

Now, consider the 3-player simple spare parts inventory situation $\varphi_9 \in \Gamma_{simple,id:h,c^{emer}}$ which is identical to φ_8 except that the demand rate of company 1 has increased: λ_1 is now 10 instead. Suppose that we have situation FIX. The simple spare parts pooling game associated with φ_9 and $\boldsymbol{S}$ is described by:

c({1}) = 15,027.03;	c({2})= 12,070.96;	c({3}) = 65,000.0;
c({1,2}) = 20,069.35;	c({1,3}) = 27,719.1;	c({2,3}) = 12,930.23;
c({1,2,3}) = 20,319.07		

We have $\mathrm{BL}_1^{FIX}(\varphi_9,\boldsymbol{S})=-20{,}862.43$, $\mathrm{BL}_2^{FIX}(\varphi_9,\boldsymbol{S})=-5{,}873.77$, $\mathrm{BL}_3^{FIX}(\varphi_9,\boldsymbol{S})=47{,}055.27$.

As $BL_1^{FIX}(\varphi_9,\boldsymbol{S}) < BL_1^{FIX}(\varphi_8,\boldsymbol{S})$, $\mathrm{BL}^{\mathrm{FIX}}$ does not always satisfy monotonicity in λ.

6.2.4 Symmetry

An allocation rule is symmetric if the costs allocated to companies with identical parameters are identical. Formally:

Definition for situation OPT: Let $\varphi \in \Gamma_{simple}$ such that $\lambda_i=\lambda_j$, $h_i=h_j$, and $c_i^{emer}=c_j^{emer}$ for some $i,j \in N$. Then, an allocation rule f^{OPT} is symmetrical if $f_i^{OPT}(\varphi)=f_j^{OPT}(\varphi)$.

Definition for situation FIX: Let $\varphi \in \Gamma_{simple}$ such that $\lambda_i=\lambda_j$, $h_i=h_j$, and $c_i^{emer}=c_j^{emer}$ for some $i,j \in N$. Let $\boldsymbol{S} \in \mathbb{N}_0^N$ such that $S_i=S_j$. Then, an allocation rule f^{FIX} is symmetrical if $f_i^{FIX}(\varphi,\boldsymbol{S})=f_j^{FIX}(\varphi,\boldsymbol{S})$.

It is obvious why this is a fair criterion; if two companies are exactly the same then they should be treated equally. We will show in Lemma 6.3 that all allocations proposed in section 6.1 are symmetric. However, interestingly, an important allocation does *not* always adhere to it. This particular allocation rule, A-local$^{\mathrm{OPT}}$, does not require any transfer payments, i.e. each company pays its own local holding and local emergency costs (included in Appendix 9, and the same as allocation policy 1 in Wong et al., 2007):

$$\text{A-local}_i^{OPT}(\varphi)=h_i\cdot\left(S_N^*\right)_i+\pi_0\Big(\sum_{j\in N}\lambda_j/\mu,\sum_{j\in N}\left(S_N^*\right)_j\Big)\cdot\lambda_i\cdot c_i^{emer}.$$

In the following example (6.6), we see that even if companies have the same input parameters, their optimal base-stock level need not be the same for both companies.

Particularly, if the optimal sum of base-stock levels for the grand coalition is not dividable by the number of players, then at least one company will get a higher base-stock level than another. In that case, allocation rule A-local$^{\mathrm{OPT}}$ will allocate more local (holding) costs to that company than to another. In the numerical experiment, this cost allocation was often not in the core; not always being symmetric could account for this.

Example 6.6: An allocation without any transfer payments is not always symmetrical.
Consider the 2-player simple spare parts inventory situation $\varphi_{10} \in \Gamma_{simple,id:\lambda,h,c^{emer}}$ with $N=\{1,2\}$, $\mu=25$, $h=400$; $c^{emer}=6500$; $\lambda=2.5$. Suppose that we have situation OPT.
The associated spare parts pooling game is described by:
c({1}) = 873.53; c({2})= 873.53; c({1,2}) = 1235.48.
We remark that for the grand coalition the optimal base-stock vector $\boldsymbol{S}^*_N$ is $\left(S^*_N\right)_1=2$ and $\left(S^*_N\right)_2=1$. A-local$^{\mathrm{OPT}}(\varphi_{10})$ yields A-local$_1^{OPT}(\varphi_{10})=817.74$; A-local$_2^{OPT}=417.74$.
These values are not the same and hence A-local$^{\mathrm{OPT}}$ does not always satisfy symmetry.

Lemma 6.3: Allocation rules AL$^{\mathrm{FIX}}$, SPLIT$^{\mathrm{FIX}}$, BL$^{\mathrm{FIX}}$, Φ^{FIX}, AL$^{\mathrm{OPT}}$, SPLIT$^{\mathrm{OPT}}$, BL$^{\mathrm{OPT}}$, and Φ^{OPT} are symmetric.
Proof:
The proof is fairly straightforward and follows easily from the cost allocation formulas. The full proof can be found in Appendix 11.
■

6.2.5 Demand dummy property

An allocation rule adheres to the demand dummy property if a player with no demand rate is not allocated positive costs. Formally:
Definition for situation OPT: An allocation rule f^{OPT} adheres to the demand dummy property if for all $\varphi \in \Gamma_{simple}$ with $\lambda_i = 0$ for some $i \in N$: $f_i^{OPT}(\varphi) \leq 0$.
Definition for situation FIX: An allocation rule f^{FIX} adheres to the demand dummy property if for all $\varphi \in \Gamma_{simple}$ with $\lambda_i = 0$ for some $i \in N$ and all $\boldsymbol{S} \in \mathbb{N}_0^N$: $f_i^{FIX}(\varphi, \boldsymbol{S}) \leq 0$.

The fairness of this property might also seem obvious; a player without any demand strain does not face any emergency costs and does not need to hold any stock for himself. However, suppose that we have situation FIX and there is a storage company that does not face demand for spare parts, but that can contribute to a cost reduction for other players because it can store spare parts very cheaply. Then if this storage company chooses a base-stock level of more than zero, it will face holding costs if acting alone (note that this will be different for situation OPT, as then the storage company will always choose a base-stock level of zero when acting alone). In such a case, an allocation rule that allocates positive costs to the storage company can still be in the core. We will now present an example (6.7) where this could happen.

Example 6.7: Allocation rules $\mathrm{BL}^{\mathrm{FIX}}$ and Φ^{FIX} do not always adhere to the demand dummy property.

In this example, company 2 is an airline that faces demand for spare parts and company 1 is a storage company that does not face demand for spare parts, but that can store those parts more cheaply than the airline company. Suppose that we have situation FIX with base-stock vector $\boldsymbol{S}$ given by S_1=3, S_2=3. Consider the 2-player simple spare parts inventory situation $\varphi_{11} \in \Gamma_{simple,id:c^{emer}}$ with N={1,2}, μ=25, h_1=400, h_2=2000; c^{emer}=6500; λ_1=0, λ_2=2.5. The associated spare parts pooling game is described by:

c({1})=1200.0; c({2})=6002.45; c({1,2})=7200.00002.

The Shapley value gives $\Phi_1^{FIX}(\varphi_{11},\boldsymbol{S})$=1198.77, $\Phi_2^{FIX}(\varphi_{11},\boldsymbol{S})$=6001.23.

Allocation $\mathrm{BL}^{\mathrm{FIX}}$ gives $\mathrm{BL}_1^{FIX}(\varphi_{11},\boldsymbol{S})$=1200, $\mathrm{BL}_1^{FIX}(\varphi_{11},\boldsymbol{S})$=6000.

Lemma 6.3: Allocations rules $\mathbf{AL}^{\mathbf{OPT}}$, $\mathbf{AL}^{\mathbf{FIX}}$, $\mathbf{SPLIT}^{\mathbf{OPT}}$, and $\mathbf{SPLIT}^{\mathbf{FIX}}$ adhere to the demand dummy property.
Proof:
It is readily seen from the cost allocation formulas that if λ_i=0, then the costs allocated to player i is zero.
■

Lemma 6.4: Allocations rules $\Phi^{\mathbf{OPT}}$ and $\mathbf{BL}^{\mathbf{OPT}}$ adhere to the demand dummy property.
Proof:
Let $\varphi \in \Gamma_{simple}$. Let $i \in N$. Suppose λ_i=0.
First we show that Φ^{OPT} adheres to the demand dummy property. Let $M \subseteq N/\{i\}$.
Observe that $c(M \cup \{i\}) \leq c(M)$, as adding a player without any demand strain to a coalition can not increase costs. Hence, $c(M \cup \{i\}) - c(M) \leq 0$.

Furthermore $\frac{|M|! \cdot (|N-M|-1)!}{|N|!} \geq 0$.

Hence, $\sum_{M \subseteq N \backslash \{i\}} \frac{|M|! \cdot (|N-M|-1)!}{|N|!} \cdot \left(c(M \cup \{i\}) - c(M)\right) \leq 0$ and therefore $\Phi_i^{OPT}(\varphi) \leq 0$.

Now we show that $\mathrm{BL}^{\mathrm{OPT}}$ adheres to the demand dummy property. Observe that $c(\{i\})$=0, as choosing a base-stock level of zero is optimal. Hence, $BL_i^{OPT}(\varphi) = c(\{i\}) - 0 = 0$.
■

6.2.6 Final remarks on cost allocation fairness

In Table 6.1 an overview is given of the four cost allocations and their fairness properties. It is readily seen that allocation rules $\mathrm{AL}^{\mathrm{OPT}}$ and $\mathrm{AL}^{\mathrm{FIX}}$ adhere to all fairness properties. For the other allocation rules, no such definite statement can be made. In fact, for some allocation rules we have shown counter-examples indicating that they do not always adhere to some fairness properties.

Table 6.1: Cost allocations and whether they adhere to properties or not

Allocation rule	Monotonic in λ	Monotonic in h	Monotonic in c^{emer}	Symmetric	Demand dummy property
AL^{OPT}	Yes	Yes	Yes	Yes	Yes
$SPLIT^{OPT}$	No	No	No	Yes	Yes
BL^{OPT}	No	Conjectured	Conjectured	Yes	Yes
Φ^{OPT})	Conjectured	Conjectured	Conjectured	Yes	Yes
AL^{FIX}	Yes	Yes	Yes	Yes	Yes
$SPLIT^{FIX}$	Conjectured	Conjectured	Conjectured	Yes	Yes
BL^{FIX}	No	Conjectured	Conjectured	Yes	No
Φ^{FIX}	Conjectured	Conjectured	Conjectured	Yes	No

6.3 Numerical experiment on cost allocations

In this section, we study the stability property of all allocations proposed in section 6.1, by means of a numerical experiment. All attainable parameter values in this experiment are very similar to the ones used in the experiment of the previous chapter. Table 5.1 shows all attainable parameter indexes, Table 5.2 shows all attainable parameter rules, and Table 5.3 shows all actual values, with one addition: we also look at $|N|=2$ and in that case, only the first two columns of Table 5.2 are used. As such, we generated 46080 games and subsequently selected only those games with a non-empty core. For each of these games, we determined (a) what type of simple spare parts inventory situation and what type of rule on base-stock levels it was associated with and (b) for each of the four cost allocations rules, whether it was in the core or not. The results are shown in Table 6.2 (separated for the number of players and type of game) and in Figures 6.1 - 6.4 (separated for class of spare parts inventory situation and rule on base-stock levels).

Table 6.2: Percentage of games for which a cost allocation is in the core, differentiated for *N* and type of game. Note that there were more games of situation FIX than OPT; hence values in the first three columns are skewed towards that.

Allocation rule	% in core for $|N|=2$	% in core for $|N|=3$	% in core for $|N|=4$	% in core for games of situation OPT	% in core for games of situation FIX	Average
AL	62.02%	58.73%	54.31%	86.74%	44.21%	58.35%
BL	100%	52.09%	44.22%	58.43%	63.18%	65.44%
The Shapley value	100%	63.22%	56.59%	82.69%	68.55%	73.27%
SPLIT	68.28%	65.32%	61.34%	94.01%	53.74%	64.98%

Observations that we can draw from the results of the numerical experiment are:

- None of the cost allocations considered were stable for all of the test cases.
- All four allocations considered here are less often in the core for larger games (i.e. more players) than for smaller games. This indicates that the allocation rules have trouble handling the intricacies of large-scale cooperations.
- For games with 3 or 4 companies and/or games with to-be-optimized base-stock levels, allocation SPLIT was most often in the core (compared to the other allocation rules).
- For games with 2 companies and/or games with fixed base-stock levels, allocation rules BL and the Shapley value were most often in the core (compared to the other rules).

- While allocation rules AL^{FIX} and AL^{OPT} satisfy all fairness criteria put forward in section 6.2 and are quite simple as well, they give less often a core element than the other allocation rules. A problem with allocating costs based on demand rates is that this is short-sighted; you only take into account differences between demand rates amongst companies. When demand rates are identical but companies differ highly on other parameters, you get an equal split, which is not smart when companies are actually highly different.

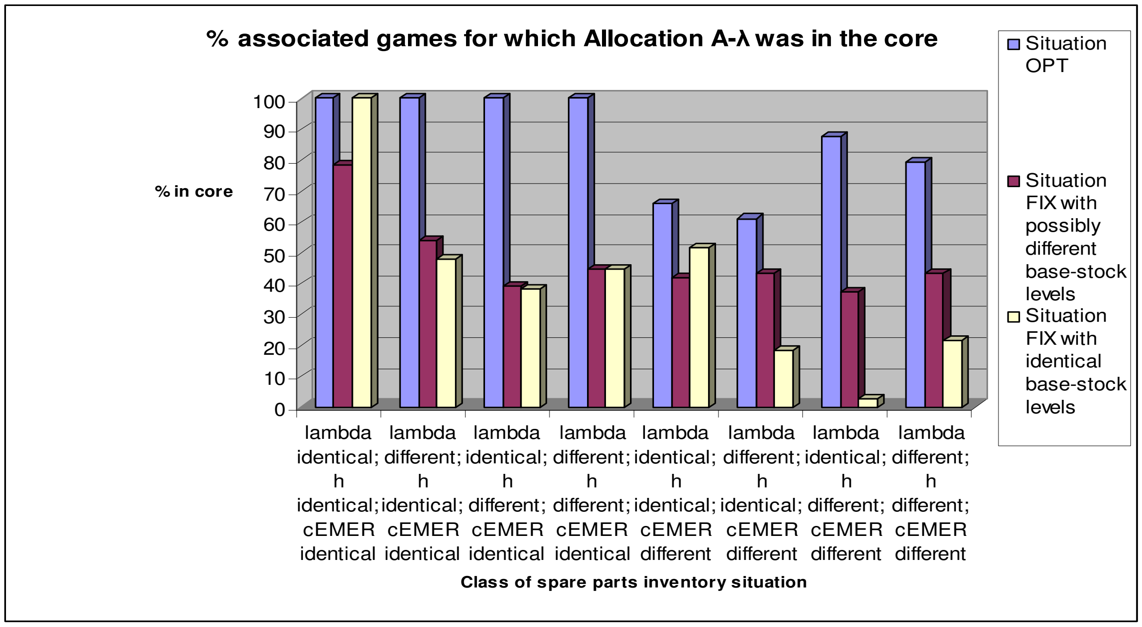

Figure 6.1: Results for Allocation AL^{OPT} (in blue) and AL^{FIX} (in purple and yellow).

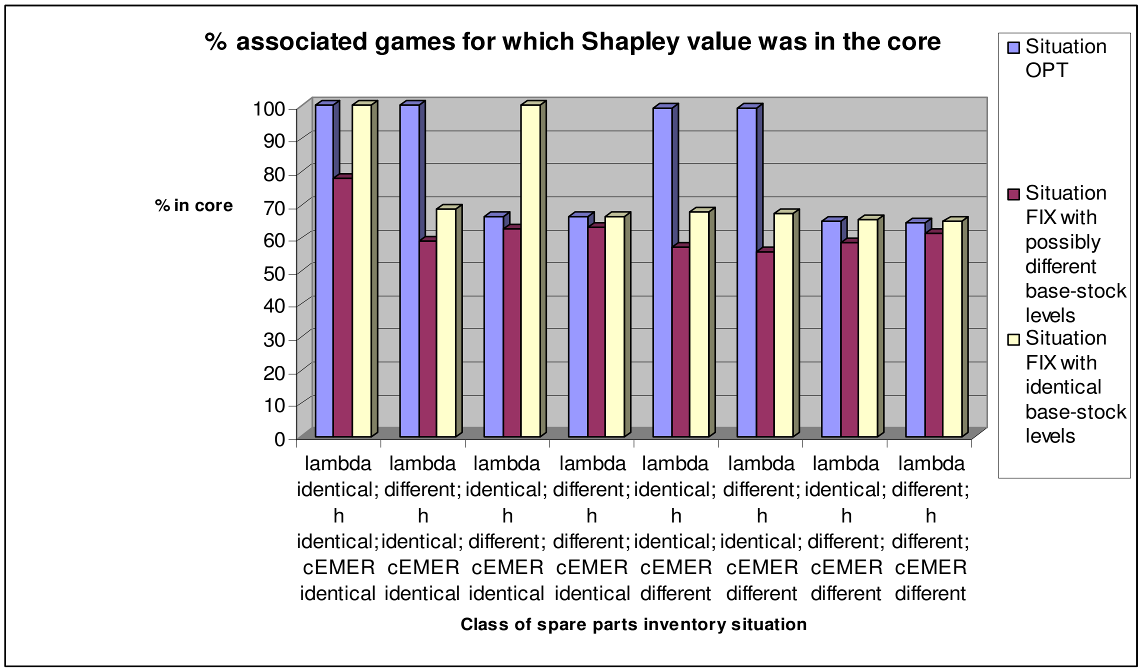

Figure 6.2: Results for the Shapley value.

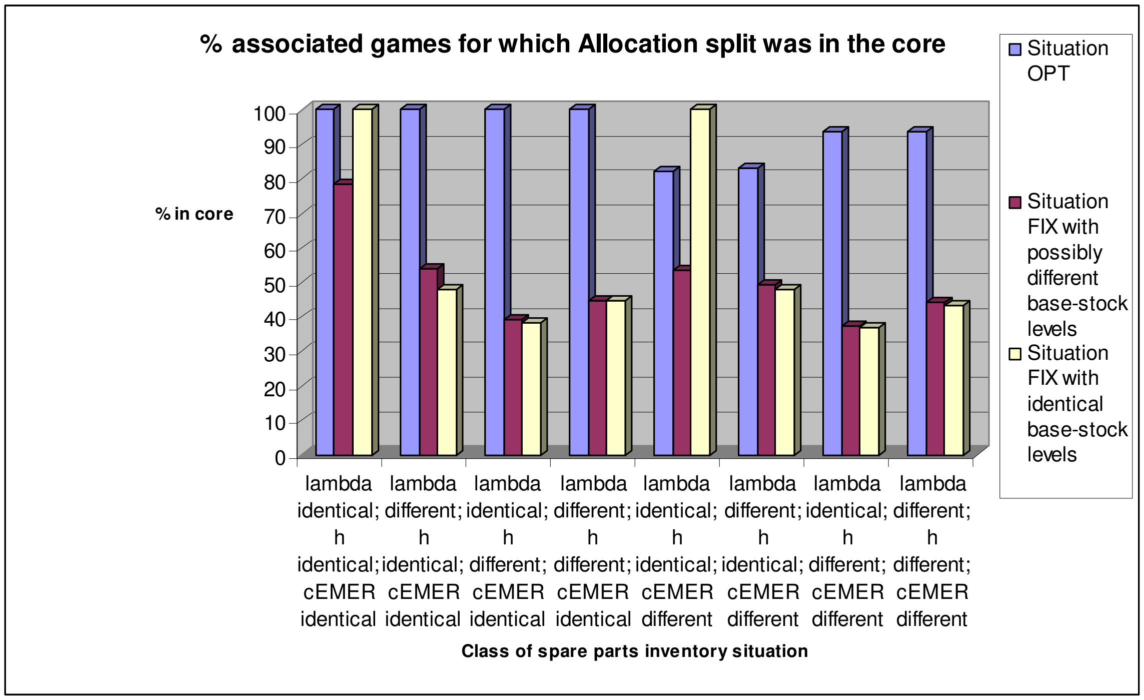

Figure 6.3: Results for Allocation SPLIT^{OPT} (in blue; on the left of each group) and SPLIT^{FIX} (in purple and yellow; middle and right of each group).

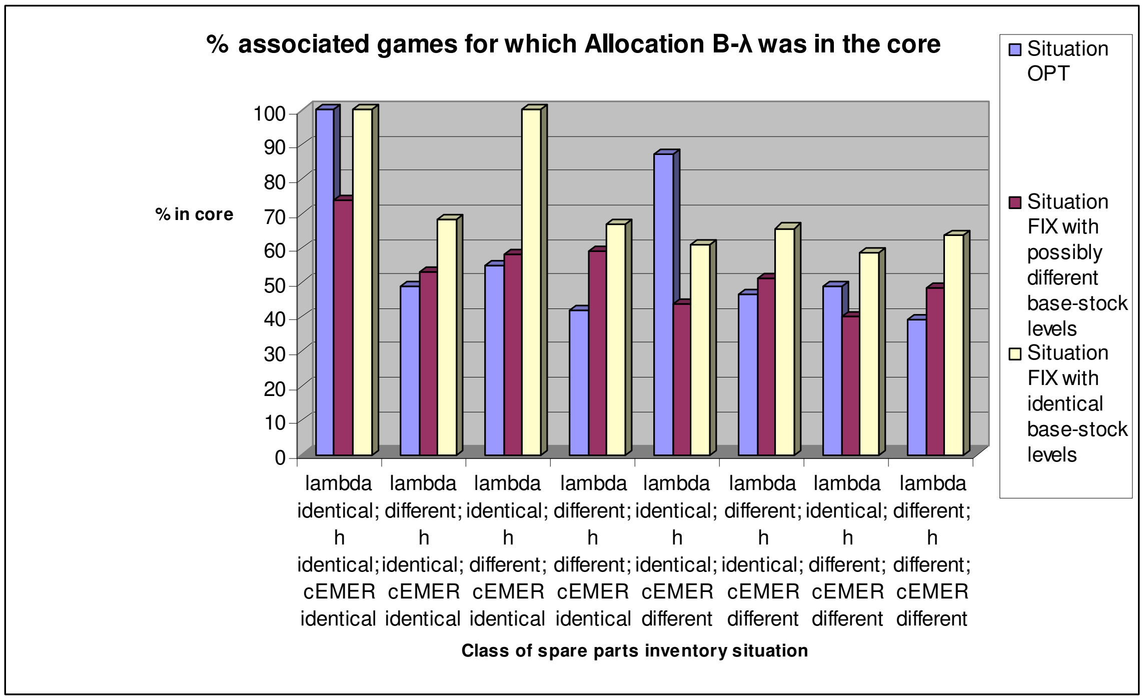

Figure 6.4: Results for Allocation BL^{OPT} (in blue; on the left of each group) and BL^{FIX} (in purple and yellow; middle and right of each group).

We will now posit three lemma's showing that for games associated with certain classes of simple spare parts inventory situations and/or rules on base-stock levels, certain cost allocation rules give core elements.

Lemma 6.5: Let $\varphi \in \Gamma_{simple,id:all}$. Let $\boldsymbol{S} \in \mathbb{N}_0^N$ with $S_i = S_j$ for all $i,j \in N$.
(i) If the game associated with φ has a non-empty core, then allocations AL$^{\text{OPT}}(\varphi)$, BL$^{\text{OPT}}(\varphi)$, $\Phi^{\text{OPT}}(\varphi)$, and SPLIT$^{\text{OPT}}(\varphi)$ are elements of its core.
(ii) If the game associated with $\boldsymbol{S}$ and φ has a non-empty core, then AL$^{\text{FIX}}(\varphi,\boldsymbol{S})$, BL$^{\text{FIX}}(\varphi,\boldsymbol{S})$, $\Phi^{\text{FIX}}(\varphi,\boldsymbol{S})$, and SPLIT$^{\text{FIX}}(\varphi,\boldsymbol{S})$ are elements of its core.
Proof:
Since all companies are fully identical, $c(M)$ is only dependent on $|M|$. Then, since the core is non-empty, an equal cost split ($x_i = c(N)/|N|$ for all $i \in N$) will be a core element. Since by Lemma 6.3 allocation rules AL$^{\text{FIX}}$, SPLIT$^{\text{FIX}}$, BL$^{\text{FIX}}$, Φ^{FIX}, AL$^{\text{OPT}}$, SPLIT$^{\text{OPT}}$, BL$^{\text{OPT}}$, and Φ^{OPT} are all symmetrical, they result in core elements.
■

Lemma 6.6: Let $\varphi \in \Gamma_{simple,id:\lambda,h}$ and $\boldsymbol{S} \in \mathbb{N}_0^N$ with $S = \boldsymbol{S}_i = \boldsymbol{S}_j$ for all $i,j \in N$ and let the associated simple spare parts pooling game be (N,c). Then SPLIT$^{\text{FIX}}(\varphi,\boldsymbol{S}) \in \text{Core}(N,c)$.
Proof:
Efficiency follows from Lemma 6.1, so it suffices to show stability:
$\sum_{i \in M} SPLIT_i^{FIX}(\varphi, S) \le c(M)$ for all $M \subseteq N$, i.e (since λ and h are identical for all players):

$$\sum_{i \in M}\left[\frac{1}{|N|}\left(h \cdot S \cdot |N|\right) + \pi_0\left(\frac{|N| \cdot \lambda}{\mu}, S \cdot |N|\right) \cdot \lambda c_i^{emer}\right] \le h \cdot S \cdot |M| + \pi_0\left(\frac{|M| \cdot \lambda}{\mu}, S \cdot |M|\right) \cdot \lambda \sum_{i \in M} c_i^{emer} \quad (6.1)$$

In order to show that (6.1) holds for all $M \subseteq N$, we let $M \subseteq N$ and start with:

$$\pi_0\left(\frac{|N| \cdot \lambda}{\mu}, |N| \cdot S\right) \le \pi_0\left(\frac{|M| \cdot \lambda}{\mu}, |M| \cdot S\right),$$ which holds by Lemma 4.2.

Multiplying by $\lambda \cdot \sum_{i \in M} c_i^{emer}$ and subsequently adding $h \cdot S \cdot |M|$ gives:

$$h \cdot S \cdot |M| + \pi_0\left(\frac{|N| \cdot \lambda}{\mu}, |N| \cdot S\right) \cdot \lambda \cdot \sum_{i \in M} c_i^{emer} \le h \cdot S \cdot |M| + \pi_0\left(\frac{|M| \cdot \lambda}{\mu}, |M| \cdot S\right) \cdot \lambda \cdot \sum_{i \in M} c_i^{emer} \quad (6.2)$$

Equation (6.2) is equivalent to equation (6.1). This completes the proof.
■

Lemma 6.7: Let $\varphi \in \Gamma_{simple,id:\lambda,c^{emer}}$ and $\boldsymbol{S} \in \mathbb{N}_0^N$ with $S = \boldsymbol{S}_i = \boldsymbol{S}_j$ for all $i,j \in N$ and let the associated simple spare parts pooling game be (N,c). Then BL$^{\text{FIX}}(\varphi,\boldsymbol{S}) \in \text{Core}(N,c)$.
Proof:
Efficiency follows from Lemma 6.1, so it suffices to show stability:

$$\sum_{i \in M} BL_i(\varphi, S) \le c(M) \text{ for all } M \subseteq N. \quad (6.3)$$

Let $M \subseteq N$. We can rewrite the left part of inequality (6.3) as:

$$\sum_{i\in M} BL_i(\varphi,S) = \sum_{i\in M}\left[c(\{i\}) - \frac{\lambda}{\sum_{j\in N}\lambda}\cdot\left(\sum_{j\in N}c(\{j\}) - c(N)\right)\right]$$

$$= \sum_{i\in M}c(\{i\}) - \frac{|M|}{|N|}\cdot\left(\sum_{j\in N}c(\{j\}) - c(N)\right)$$

$$= \sum_{i\in M}h_i\cdot S + |M|\cdot\pi_0(\frac{\lambda}{\mu},S)\cdot\lambda\cdot c^{emer} - \frac{|M|}{|N|}\cdot\left(\sum_{i\in N}h_i\cdot S + |N|\cdot\pi_0(\frac{\lambda}{\mu},S)\cdot\lambda\cdot c^{emer} - \sum_{i\in N}h_i\cdot S - \pi_0(\frac{|N|\cdot\lambda}{\mu},|N|\cdot S)\cdot|N|\cdot\lambda\cdot c^{emer}\right)$$

$$= \sum_{i\in M}h_i\cdot S + |M|\cdot\pi_0(\frac{\lambda}{\mu},S)\cdot\lambda\cdot c^{emer} - |M|\cdot\pi_0(\frac{\lambda}{\mu},S)\cdot\lambda\cdot c^{emer} + |M|\cdot\pi_0(\frac{|N|\cdot\lambda}{\mu},|N|\cdot S)\cdot\lambda\cdot c^{emer}$$

$$= \sum_{i\in M}h_i\cdot S + \pi_0(\frac{|N|\cdot\lambda}{\mu},|N|\cdot S)\cdot|M|\cdot\lambda\cdot c^{emer}$$

Furthermore, the right part of (6.3) is: $c(M) = \sum_{i\in M}h_i\cdot S + \pi_0(\frac{|M|\cdot\lambda}{\mu},|M|\cdot S)\cdot|M|\cdot\lambda\cdot c^{emer}$

Hence, the following inequality (6.4) is identical to (6.1):

$$\sum_{i\in M}h_i\cdot S + \pi_0(\frac{|N|\cdot\lambda}{\mu},|N|\cdot S)\cdot|M|\cdot\lambda\cdot c^{emer} \le \sum_{i\in M}h_i\cdot S + \pi_0(\frac{|M|\cdot\lambda}{\mu},|M|\cdot S)\cdot|M|\cdot\lambda\cdot c^{emer} \quad (6.4)$$

In order to show that (6.4) holds, we start with: $\pi_0\left(\frac{|N|\cdot\lambda}{\mu},|N|\cdot S\right) \le \pi_0\left(\frac{|M|\cdot\lambda}{\mu},|M|\cdot S\right)$,

which holds by Lemma 4.2. We then multiply by $\lambda\cdot c^{emer}\cdot|M|$ and subsequently add $\sum_{i\in M}h\cdot S$ to complete the proof.

■

The following conjectures are based on the results of the experiment (Figures 6.1 - 6.4):

Conjecture 6.3: Let $\varphi\in\Gamma_{simple,id:c^{emer}}$. If the associated simple spare parts pooling game (*N*,*c*) has a non-empty core, then $\mathrm{AL}^{\mathrm{OPT}}(\varphi)$ and $\mathrm{SPLIT}^{\mathrm{OPT}}(\varphi)$ are core elements.

Conjecture 6.4: Let $\varphi\in\Gamma_{simple,id:h,c^{emer}}$. If the associated simple spare parts pooling game (*N*,*c*) has a non-empty core, then $\Phi^{\mathrm{OPT}}(\varphi)$ is a core element.

Conjecture 6.5: Let $\varphi\in\Gamma_{simple,id:\lambda,c^{emer}}$ and $\boldsymbol{S}\in\mathbb{N}_0^N$ with $S_i=S_j$ for all $i,j\in N$. Then $\Phi^{\mathrm{FIX}}(\varphi,\boldsymbol{S})$ is a core element.

In conclusion, unfortunately no cost allocation rule has been found so far that is always in the core (if non-empty) of a simple spare parts pooling game. No clear-cut practical recommendation can be made at this point, as there is a trade-off between stability, simplicity, and fairness of cost allocation rules. The stability property can be viewed as the most important (see, e.g., Hartman&Dror, 1996). Hence, companies may have to resort to using the nucleolus, particularly for situations with only a small number of companies, as then the computational complexity of the nucleolus is manageable.

Chapter 7: More complex settings

In this chapter we attempt to answer research question 3 ("Can we generalize results to a more complex setting?"). First, we look at the class of general spare parts inventory situations Γ, for which we derive an expression of steady-state probabilities and describe the characteristic cost functions of spare parts pooling games that can be associated with it (Sections 7.1 and 7.2). Then we present a numerical experiment on these spare parts pooling games (Section 7.3). Subsequently, we look at a partial pooling approach, for which we derive an expression of steady-state probabilities and describe the characteristic cost functions of simple partial parts pooling (Sections 7.4 and 7.5). Then we present a numerical experiment on these simple partial pooling games (Section 7.6).

7.1 Steady state probabilities for a general spare parts inventory situation

Consider a general spare parts inventory situation $\varphi \in \Gamma$. The system behavior of a coalition $M \subseteq N$ may be described by an $|M|$-dimensional Markov process with state $\boldsymbol{x}=\{x_1,x_2,\ldots,x_{|M|}\}$, where x_i represents the on-hand inventory at company i, $0 \le x_i \le S_i$.

We define for all $i\in M$: $\boldsymbol{x}_{i-}(\boldsymbol{x})= \begin{cases} x_j - 1 & if\ j = i \\ x_j & if\ j \in M \setminus i \end{cases}$ and $\boldsymbol{x}_{i+}(\boldsymbol{x})= \begin{cases} x_j + 1 & if\ j = i \\ x_j & if\ j \in M \setminus i \end{cases}$

All possible transitions of the Markov process are as follows (similar to Wong, 2007a):
Transition 1 (regular demand): A failure of a part occurs at location i while x_i>0; the state transition is $\boldsymbol{x} \rightarrow \boldsymbol{x}_{i-}(\boldsymbol{x})$ and the transition rate is λ_i.
Transition 2 (emergency supply): A failure of a part occurs at company i while x_j=0 for all $j\in M$; the state transition is $\boldsymbol{x} \rightarrow \boldsymbol{x}$ and the transition rate is λ_i.
Transition 3 (repair complete): The repair of a part belonging to company i is completed; the state transition is $\boldsymbol{x} \rightarrow \boldsymbol{x}_{i+}(\boldsymbol{x})$ and the transition rate is $(S_i-x_i)\cdot\mu_i$.

All that remains is lateral transshipments. Transshipments to company i are sourced from the company j with the lowest transshipments costs to company i, c_{ji}^{trans}. Ties are broken by sourcing from the company with the highest current on-hand inventory, x_j (note that the type of transshipment is probably not very important; see Tagaras, 1999). Formally, the function that determines which company in $M\setminus\{i\}$ will source a transshipment to company i in Markov state $\boldsymbol{x}$, source(M,$\boldsymbol{x}$,i), is defined as follows, using the set of feasible Companies with Lowest transshipment Costs, CLC, and the set of feasible Companies with Highest Inventory, CHI:

$$CLC(M,\boldsymbol{x},i)=\left\{j \mid j\in M\setminus\{i\}, x_j>0, (c_{ji}^{trans} \le c_{ki}^{trans})\forall(k\in M\setminus\{i,j\}, x_k>0)\right\}.$$

$$CHI(M,\boldsymbol{x},i)=\left\{j \mid j\in CLC(M,x,i), (x_j \ge x_k)\forall(k\in CLC(M,x,i)\setminus\{j\})\right\}.$$

$$source(M,\boldsymbol{x},i) = \underset{j\in CHI(M,x,i)}{Min}\ j\ .$$

Transition 4 (lateral transshipment): A failure of a part occurs at location i while x_i=0 and at least one other company $j \in M \backslash i$ has x_j>0. Select company j=*source*(M,$\boldsymbol{x}$,i). The state transition is $\boldsymbol{x} \to \boldsymbol{x}_{j-}(\boldsymbol{x})$ and the transition rate is λ_i.

An example Markov chain for M={1,2,3}, with c^{trans} identical for all transshipment routes and $\boldsymbol{S}$ given by S_1=1, S_2=2, S_3=1, is pictured in Figure 7.1.

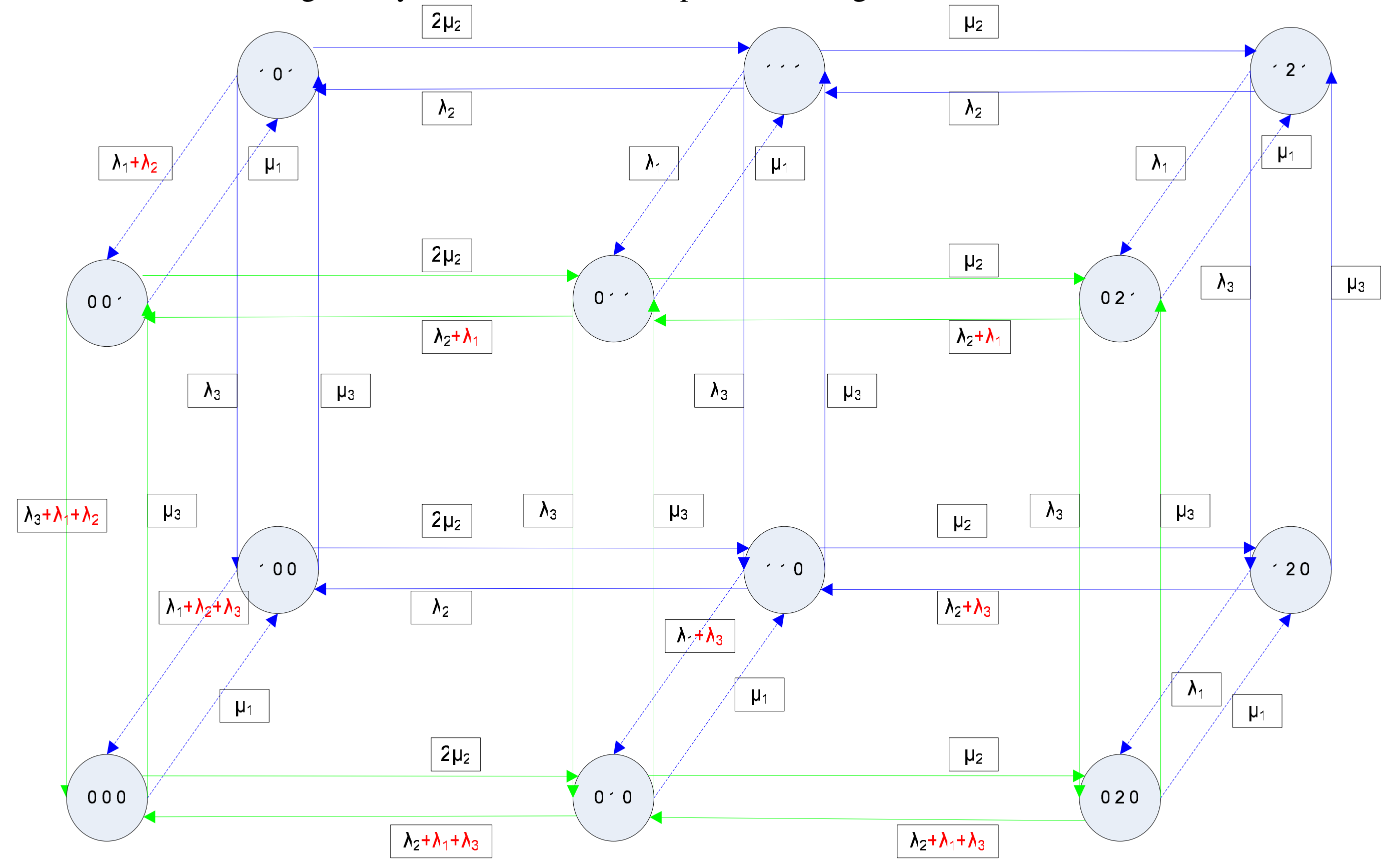

Figure 7.1: The example Markov chain. Red text implies lateral transshipments.

From a Markov chain we can obtain the steady-state probabilities of being in state $\boldsymbol{x}$, π_x. According to Wong (2006), "since the number of states in our problem is not large, a direct method based on Gaussian elimination can be applied to determine $\boldsymbol{\pi}$".

7.2. Characteristic cost functions of general pooling games

Let $\varphi \in \Gamma$ and consider coalition $M \subseteq N$. Let the base-stock vector for this coalition be $\boldsymbol{S} \in \mathbb{N}_0^{\ M}$. The total expected costs per unit of time that coalition M has to pay is:

$$K^{\varphi;S}(M) = \sum_{i\in M} h_i \cdot S_i + \pi_{0_M} \cdot \sum_{i\in M} \lambda_i \cdot c_i^{emer} + \sum_{i\in M} \sum_{x\in X0_i^M} \pi_s \cdot \lambda_i \cdot c^{trans}_{source(M,x,i),i} \quad \text{, where:} \qquad (7.1)$$

- 0_M refers to the state in which x_i=0 for all $i \in M$.
- $X0_i^M$ refers to the set consisting of all states for coalition M in which x_i=0.
- *source*(M,s,i) as defined in Section 7.1.

We remark that equation (7.1) can be seen as an extension of cost function (4.2) to $\varphi \in \Gamma$.

Characteristic cost function for a general spare parts pooling game with situation FIX
Let $\varphi \in \Gamma$ and let the chosen fixed base-stock level vector be $\boldsymbol{S} \in \mathbb{N}_0^N$. With the combination of φ and $\boldsymbol{S}$ we can associate a spare parts pooling game (N, c) that is defined by $c(M) = \mathrm{K}^{\varphi;S}(M)$ and K given by equation (7.1).

Characteristic cost function for a general spare parts pooling game with situation OPT
Let $\varphi \in \Gamma$. Consider a coalition $M \subseteq N$, which can choose a base-stock vector $\boldsymbol{S} \in \mathbb{N}_0^M$ with associated expected costs per time unit $\mathrm{K}^{\varphi;S}(M)$, given by equation (7.1). The optimal base-stock vector for coalition M is $\boldsymbol{S}^*_M$ with minimal cost $K^{\varphi;S^*_M}(M)$. So, with φ we can associate a spare parts pooling game (with to-be-optimized stock levels) (N, c) that is defined by $c(M) = \underset{S \in N_0^M}{Min}\left(\mathrm{K}^{\varphi;S}(M)\right)$ and K given by equation (7.1).

Calculation of these cost functions involves creating a Markov chain and calculating steady-state probabilities. Furthermore, calculating optimal costs when base-stock levels are to-be-optimized involves finding an optimal solution in an infinite space (which can be bounded and enumerated efficiently). The algorithms that do this are described in Appendix 12 (they may be of independent interest) and are implemented in Java.

7.3: Numerical experiment - cores and cost allocations in general spare parts pooling games

In this section, we do a numerical experiment on 3-company spare parts pooling games with parameters selected such that they are realistic for real-life situations. The methodology resembles the one used in the numerical experiment of Chapter 5. However, because adding lateral transshipment costs greatly increases computational complexity, we generated less games and we chose parameter values such that the computational time needed (particularly for games with to-be-optimized base-stock levels) was reduced. For When we have relatively low demand rates, low emergency costs, and low transshipment costs together with fast repair rates and high holding costs, it will be optimal to have low base-stock levels and the algorithm will not take a lot of time finding these. Therefore we take values in these ranges. We allow λ to take on multiple values, as λ is a crucial parameter for some cost allocation rules. We allow c^{emer} to take on multiple values, as c^{emer} appeared to be the most important parameter in game balancedness. We will of course have games with to-be-optimized and with fixed base-stock levels (in the latter case, we will take the same, identical base-stock levels for all test cases). Finally, we have three different values for the transshipment costs:

- All-Zero: this case actually corresponds to what is used in the previous chapters and acts as a reference.
- All-Standard: transshipment costs are a non-zero identical fraction of emergency costs (we did this in order to avoid cases where $c_{ij}^{trans} > c_j^{emer}$ for some $i,j \in N$, as that would imply that transshipments are too expensive to consider).
- DIFF: this situation represents a case where companies 1 and 2 lie close together, while transportation time to company 3 is relatively long.

All values are shown in Table 7.1; this amounts to 24 games total. All numeric values except the ones for c^{trans} have also been used in Chapter 5.

Table 7.1: Selected values for the company parameters

|N|	3
λ	All-Low (2.5) or DIFFLow (λ_1=0.5, λ_2=2.5, λ_3=5)
μ	All-Standard (25)
h	All-High (8000)
c^{emer}	All-Low (6500) or DIFFLow ($c_1^{emer} = 13000, c_2^{emer} = 2600, c_3^{emer} = 6500$)
c^{trans}	All-Zero ($c_{i,j}^{trans} = 0$) or All-Standard ($c_{i,j}^{trans} = 0.2 \cdot c_j^{emer}$) or DIFF ($c_{1,2}^{trans} = 0.01 \cdot c_2^{emer}$, $c_{2,1}^{trans} = 0.01 \cdot c_1^{emer}$, $c_{1,3}^{trans} = 0.8 \cdot c_3^{emer}$, $c_{3,1}^{trans} = 0.8 \cdot c_1^{emer}$, $c_{2,3}^{trans} = 0.8 \cdot c_3^{emer}$, $c_{3,2}^{trans} = 0.8 \cdot c_2^{emer}$)
Rule on ***S***	To-be-optimized or Fixed (S_1=2, S_2=2, S_3=2)

Our focus of inquiry will be on the effect of the addition of transshipment costs, i.e.:

- Is there a difference between how often games have empty cores for situations with negligible transshipment costs, non-zero transshipment costs, and highly different transshipment costs?
- Is there a difference between how often the four cost allocations of Chapter 6 are in the core for situations with negligible transshipment costs, non-zero transshipment costs, and highly different transshipment costs? Note that allocation rules AL, BL, and the Shapley value can be applied right away to situations with non-zero transshipment costs - as they are stated in Section 6.1 in terms of $K^{\varphi,S}(N)$ - by simply extending them to $\varphi \in \Gamma$ (equation 7.1). However, Allocation SPLIT doesn't take into account transshipment costs. Therefore, we make a small adjustment to get an Allocation SPT (name based on SPlit-Trans), in which total transshipment costs are allocated in similar fashion as total emergency costs:

<u>Formula for situation FIX</u>: Let $\varphi \in \Gamma$ and $\boldsymbol{S} \in \mathbb{N}_0^N$. Then, for all $i \in N$:

$$SPT_i^{FIX}(\varphi, \boldsymbol{S}) = \frac{\lambda_i}{\sum_{j \in N} \lambda_j} \cdot \left(\sum_{j \in N} h_j \cdot S_j \right) + \pi_{0_N} \cdot \lambda_i \cdot c_i^{emer} + \left(\sum_{j \in M} \sum_{x \in X0_j^N} \pi_s \cdot \lambda_j \cdot c_{source(M,x,j),j}^{trans} \right) \cdot \frac{\lambda_i \cdot \sum_{j \in N} c_{j,i}^{trans}}{\sum_{k \in N} \lambda_k \cdot \sum_{j \in N} c_{j,k}^{trans}}$$

<u>Formula for situation OPT</u>: Let $\varphi \in \Gamma$. For all $i \in N$: $SPT_i^{OPT}(\varphi) = SPT_i^{FIX}(\varphi, \boldsymbol{S}_N^*(\varphi))$.[20]

The results of the numerical experiment are shown in Table 7.2. For all 24 games the core was non-empty. This would indicate that adding transshipment costs does not have a large effect on the balancedness of a game. Furthermore, allocation rule SPT performed best when transshipment costs were "All-Standard", but was less often in the core when transshipment costs differed largely between companies.

[20] Note that $\boldsymbol{S}_N^*(\varphi)$ has not been formally defined for $\varphi \in \Gamma \setminus \Gamma_{simple}$ yet. For, the method to obtain $\boldsymbol{S}_N^*(\varphi)$ is by using the algorithm described in Appendix 12.

Allocation rules AL and BL, which only take into account the demand rate parameter, often did not provide core elements when highly different transshipment costs are added as another complicating factor to the spare parts inventory situation. However, do note that this has been a very limited testbed that merely serves to give some first indications.

Table 7.2: Results of the numerical experiment for general spare parts pooling games.

c^{trans}	Core	AL in core	SPT in core	Shapley value in core	BL in core
All-Zero	100% non-empty	5/8 games	6/8 games	5/8 games	4/8 games
All-Standard	100% non-empty	3/8 games	6/8 games	5/8 games	4/8 games
DIFF	100% non-empty	1/8 games	3/8 games	5/8 games	1/8 games

7.4 Steady state probabilities for a simple spare parts inventory situation with partial pooling

Consider a simple spare parts inventory situation $\varphi \in \Gamma_{simple}$ and a coalition $M \subseteq N$. Let the base-stock vector for this coalition be $\boldsymbol{S} \in \mathbb{N}_0^M$. For the remainder of this chapter, we relax the assumption that companies employ full pooling. Instead, companies will use a partial pooling approach, will which now be explained. Recall that in a simple spare parts inventory situation, transshipments are free and companies have an identical exponential repair process. For the partial pooling approach that we will consider, only the total on-hand inventory at all companies together (denoted with x) has to be known. Hence, the system behavior may be described by a one-dimensional Markov process with state x, $0 \le x \le S_M$, with $S_M = \sum_{i \in M} S_i$.

The partial pooling process employs for each company $i \in M$ a trigger level T_i, $0 \le T_i \le S_M$. If a demand comes in for company i, if $x>T_i$ then it is fulfilled from the pooling stock, else it is fulfilled by emergency shipment. When T_i=0 for all $i \in M$ then this is equivalent to full pooling. When T_i=S_M for all $i \in M$ then this is equivalent to fulfilling every incoming demand by emergency shipment (and "wasted" pooling stock).

An example is shown in Figure 7.2 for M={1,2,3}, S_M=4, T_1=4, T_2=2, T_3=0. Note that when $c_3^{emer} >> c_1^{emer}$ such an arrangement could make sense intuitively.

$$\begin{tikzcd}
0 \arrow[r, shift left, "4\cdot\mu"] & 1 \arrow[l, shift left, "\lambda_3"] \arrow[r, shift left, "3\cdot\mu"] & 2 \arrow[l, shift left, "\lambda_3"] \arrow[r, shift left, "2\cdot\mu"] & 3 \arrow[l, shift left, "\lambda_2+\lambda_3"] \arrow[r, shift left, "\mu"] & 4 \arrow[l, shift left, "\lambda_2+\lambda_3"]
\end{tikzcd}$$

Figure 7.2: The Markov process of an example situation with partial pooling.

All possible transitions of the Markov process are as follows:
Transition 1 (demand): A failure of a part occurs at location i while $x>T_i$; the state transition is $x \rightarrow x-1$ and the transition rate is λ_i.
Transition 2 (emergency supply): A failure of a part occurs at company i while $x \leq T_i$; the state transition is $x \rightarrow x$ and the transition rate is λ_i.
Transition 3 (repair complete): The repair of a part is completed; the state transition is $x \rightarrow x+1$ and the transition rate is $(S_M-x)\cdot\mu$.

From a Markov chain we can obtain the steady-state probabilities of being in state x, π_x. We remark that this partial pooling approach is different from the one used in Wong et al (2007a). They set a reserved stock level for a company, which will only supply a lateral transshipment to another company if its current inventory level is above its reserved stock level. However, we consider situations with negligible transshipment costs, for which it is optimal to put all stock at the company with the lowest holding cost rate. Using the partial pooling process of Wong et al (2007a), this company still cannot discern between a lateral transshipment request coming in from a neighbor company with high emergency costs (call this one A) and one with low emergency costs (call this one B). In our partial pooling process A may be given a spare part from stock whereas B is told that the spare parts stock are reserved for A instead, which appears to be a smarter approach.

7.5. Characteristic cost functions of simple partial pooling games

Let $\varphi \in \Gamma$ and consider coalition $M \subseteq N$. Let the base-stock vector for this coalition be $\boldsymbol{S} \in \mathbb{N}_0^{\,M}$. Let the trigger levels for this coalition be $\boldsymbol{T} \in \mathbb{N}_0^{\,M}$, $0 \leq T_i \leq S_M$ for all $i \in M$.
The total expected costs per unit of time that coalition M has to pay is:

$$K_{pp}^{\varphi;S;T}(M) = \sum_{i \in M} h_i \cdot S_i + \sum_{i \in M} \sum_{p \in P(S_M, T_i)} \pi_p \cdot \lambda_i \cdot c_i^{emer} \quad \text{, where:} \tag{7.2}$$

- $P(S_M,T_i)$ refers to the set consisting of all states in $0 \leq x \leq S_M$ for which $x \leq T_i$.

An optimal trigger level vector for M with this $\boldsymbol{S}$ is $\boldsymbol{T}_M^{opt}$ with minimal cost:

$$K_{pp,opt}^{\varphi;S}(M) \equiv K_{pp}^{\varphi;S;T_M^{opt}}(M) = \underset{T \in \{0,1,\ldots,S_M\}^M}{Min} K_{pp}^{\varphi;S;T}(M). \tag{7.3}$$

Characteristic cost function for a simple partial pooling game with situation FIX
Let $\varphi \in \Gamma$ and let the chosen fixed base-stock level vector be $\boldsymbol{S} \in \mathbb{N}_0^{\,N}$. With the combination of φ and $\boldsymbol{S}$ we can associate a simple partial pooling game (N,c) that is defined by $c(M) = K_{pp,opt}^{\varphi;S}(M)$ and $K_{pp,opt}$ given by equation (7.3).

Characteristic cost function for a simple partial pooling game with situation OPT
Let $\varphi \in \Gamma$. Consider a coalition $M \subseteq N$, which can choose a base-stock vector $\boldsymbol{S} \in \mathbb{N}_0^{\,M}$ with associated expected costs per time unit $K_{pp,opt}^{\varphi;S}(M)$, given by equation (7.3). An optimal base-stock vector for coalition M is $\boldsymbol{S}_M^{opt}$ with minimal cost $K_{pp,opt}^{\varphi;S_M^{opt}}(M)$.

So, with φ we can associate a simple partial pooling game (with to-be-optimized stock levels) (N,c), defined by $c(M) = \underset{S\in N_0{}^M}{Min}\left(K^{\varphi;S}_{pp,opt}(M)\right)$ and $K_{pp,opt}$ given by equation (7.3).

Algorithms that calculate these cost functions are described in Appendix 13 and are implemented in a Java application.

7.6: Numerical experiment - cores and cost allocations in simple partial pooling games

In this section, we do a numerical experiment on 3-company simple partial pooling games with parameters selected such that they are realistic for real-life situations. The methodology resembles the one used in the numerical experiment of Chapter 5. However, because adding the partial pooling approach increases computational complexity, we generated fewer games and selected value ranges in order to limit the necessary computational time, in similar fashion as Section 7.3. Furthermore, in Chapter 5 it seemed that due to the full pooling assumption, games associated with simple spare parts inventory situations in which companies had different emergency costs could have empty cores. We speculate that a smarter partial pooling approach alleviates this effect and hence leads to fewer games with empty cores. Therefore, we limit our investigation to simple spare parts inventory situations with different emergency costs. We will also limit our investigation to games with to-be-optimized base-stock levels, as this allows us to fairly compare partial and full pooling approaches on the same spare parts inventory situations, without having to worry about the effect that fixed base-stock levels may have. All parameter indices used are shown in Table 7.3, where Table 5.2 shows for each index the corresponding "rule" on the parameter values of each company and Table 5.3 shows the actual corresponding values. This amounts to 240 spare parts inventory situations total. With each spare parts inventory situation we can associate one simple spare parts pooling game and one simple partial pooling game.

Our focus of inquiry will be on the effect of the addition of partial pooling, i.e.:

- Is there a difference between how often games have empty cores for situations with partial pooling and full pooling?
- Is there a difference between how often the four cost allocations of Chapter 6 are in the core for situations with partial pooling and full pooling? For simple spare parts pooling games, we look at allocation rules AL^{OPT}, BL^{OPT}, $SPLIT^{OPT}$ and Φ^{OPT} (as defined in Section 6.1). For simple partial pooling games, we construct allocation rules $AL\text{-}PP^{OPT}$, $BL\text{-}PP^{OPT}$, and $\Phi\text{-}PP^{OPT}$, by replacing K with $K_{pp;opt}$ in the definitions of AL^{OPT}, BL^{OPT}, and Φ^{OPT} , respectively. However, once again allocation rule SPLIT is not as easily transformed. Therefore, let $\varphi\in\Gamma_{simple}$ and define for all $i\in N$:

$$SP-PP^{OPT}_i(\varphi) = \frac{\lambda_i}{\sum_{j\in N}\lambda_j}\cdot\left(\sum_{j\in N}h_j\cdot S_j\right) + \left(\underset{S\in N_0{}^M}{Min}\left(K^{\varphi;S}_{pp,opt}(N)\right) - \sum_{j\in N}h_j\cdot S_j\right)\cdot\frac{\lambda_i\cdot c_i^{emer}}{\sum_{j\in N}\lambda_j\cdot c_j^{emer}}$$

Table 7.3: Selected parameter indices for the company parameters

$\|N\|$	3
λ	4 (All-Low, All-Standard, All-High, DIFF2)
μ	3 (All-Min, All-Low, All-Standard)
h	4 (All-Standard, All-High, All-Max, DIFF2)
c^{emer}	5 (All-Low, All-Standard, All-High, DIFF1, DIFF2)
Rule on $\boldsymbol{S}$	To-be-optimized

The results of the numerical experiment are shown in Table 7.4. 20 out of 240 games with full pooling had empty cores. But no games with partial pooling had empty cores! This would indicate that using a partial pooling approach has a large effect on the balancedness of a game. Furthermore, it supports the idea that the full pooling approach was the main reason why we got games with empty cores in Chapter 5. Finally, allocation rule SP-PP^{OPT} performed best for games with partial pooling, while all other allocation rules performed worse for games with partial pooling than for games with full pooling.

Table 7.4: Results of the numerical experiment for partial pooling games. The third through seventh column consider only the games with non-empty cores and determine how often a certain allocation rule was in the core for that subset.

Pooling process	Core	Rule AL^{OPT} or AL-PP^{OPT}	Rule Φ^{OPT} or Φ-PP^{OPT}	Rule SPLIT^{OPT} or SP-PP^{OPT}	Rule BL^{OPT} or BL-PP^{OPT}
Full pooling	92% non-empty	85.9% in Core	99.5% in Core	93.6% in Core	78.6% in Core
Partial pooling	100% non-empty	80.4% in Core	88.3% in Core	99.1% in Core	74.6% in Core

Chapter 8: Conclusions

In Section 8.1, conclusions regarding the research questions are provided. In Section 8.2, suggestions for future research are made.

8.1: Conclusions regarding the research questions

The problem statement in this paper was:
The scientific literature currently gives no insight into the non-emptiness of the core in a spare parts pooling game and there is insufficient knowledge about proper cost allocation policies that are proven to be in the core of the cooperative game. This lack of managerial insights may impede profitable collaboration on spare parts pooling.

In order to provide a solution to this problem, three main research questions were formulated. We will now provide conclusions regarding each of these research questions.

Research question 1a: Does a simple spare parts pooling game, with the base-stock levels already pre-determined at arbitrary values, have a non-empty core?

We proved that:
- For a game associated with a spare parts inventory situation where all companies have the same demand rate and the same fixed base-stock levels (other parameters may be asymmetrical), the core is non-empty.
- For a game associated with a spare parts inventory situation where all companies have the same emergency costs and demand rates (and possibly different fixed base-stock levels), the core is non-empty.

Based on the results of a numerical experiment, we conjectured that any game associated with a spare parts inventory situation where all companies have the same emergency costs has a non-empty core. Games with empty cores have also been found. Empty cores are most often found for games associated with spare parts inventory situations in which the emergency costs differed largely between companies, and where companies had very low repair rates and/or very high holding cost rates. In these games, emergency costs dominate holding costs and companies with low emergency costs take spare parts that would have better been saved for companies with high emergency costs.

Research question 1b: Does a simple spare parts pooling game, where the base-stock levels are not yet determined and can still be jointly optimized, have a non-empty core?

We proved that:
- For a game associated with a spare parts inventory situation where companies are fully identical, the imputation set is non-empty. But a non-empty imputation set does not always imply a non-empty core.

- For a three-player game associated with a spare parts inventory situation where companies are fully identical with realistic base-stock levels, the core is non-empty.

Based on the results of a numerical experiment, we conjectured that any game associated with a spare parts inventory situation where all companies have the same emergency costs has a non-empty core. Games with empty cores have also been found, and the statements made for research question 1a also apply here.

Research question 2: What is a proper cost allocation policy for a spare parts pooling game?

There is a trade-off between (i) simplicity, (ii) always being in the core, (iii) fairness (various fairness properties appropriate for the spare parts setting were defined, e.g., if a company gets a higher demand rate, it should not be allocated less costs). Many allocation methods were considered and tested in a large numerical experiment, but so far no policy was found that satisfies all three requirements. Currently, no allocation rule is available that that easily handles the intricacies of large-scale cooperations of asymmetrical companies well. Two allocations rules that performed reasonably well and that can specifically be applied to spare parts pooling games are:
Allocation SPLIT: Total holding costs are allocated based on the demand rate of each company. Each company pays their own local emergency costs.
The Shapley value: A well-established allocation rule in game theory literature.

Observations that we can draw from the results of the numerical experiment are:

- For games with 2 companies and/or games with fixed base-stock levels, the Shapley value was often in the core (compared to other allocation rules).
- For games with 3 or 4 companies and/or games with to-be-optimized base-stock levels, allocation rule SPLIT relatively often gave core elements.

Lastly, the nucleolus, while (computationally) difficult, deserves consideration, since it is guaranteed to be in the core of the game.

Research question 3: Can we generalize results to a more complex setting?

We investigated a setting where transshipment costs were non-negligible with a small numerical experiment. Initial findings indicate that adding transshipment costs does not have a large effect on non-emptiness of the core. However, the cost allocation methods considered above less often produced core elements. However, we stress that this was a very limited study. We also investigated a setting with a smart partial pooling rather than full pooling. Initial findings indicate that using this partial pooling approach leads to significantly fewer games with empty cores. Finally, allocation rules SPLIT and the Shapley value performed best for these more complex settings.

8.2: Directions for future research

We posit the following directions for further scientific research:

- We conjectured that any game associated with a spare parts inventory situation where all companies have the same emergency costs has a non-empty core. A proof for this should be found.[21] Furthermore, it is also reasonable to assume that the core will be non-empty if the emergency costs of the companies are very close to each other, so trying to find such conditions could also be interesting.
- Similarly, we made several conjectures on properties of cost allocation rules in Chapter 6. A proof for those should be found as well.
- More effort is needed to find better allocation rules, preferably one that is always in the core of a simple spare parts pooling game.
- An interesting extension is to extend the single-echelon structure (this is identified as a future research direction in Wong et al., 2007a) to a structure with main and local warehouses. This can yield significant benefits for cooperating companies if setting up warehouses to allow for lateral transshipments is expensive, according to Kranenburg & Van Houtum (2008).
- We have looked at expected costs rather than realized costs so far. Realization games may offer new insights. For example, for allocation rule AL, does it make sense for a given yearly realization of costs, to allocate these based on expected demand rates or on realized demand rates?
- We have looked at a single-item situation so far. A multi-item approach could make the model and cooperation process richer. Investigating this could bring interesting new insights.

[21] The work of Sandra van Wijk, doctoral candidate at the TU/e, who investigates for (what we call) situation FIX for which conditions full pooling is optimal, is related to this.

References

- Alfredsson, P., Verrijdt, J. (1999), ‘Modeling emergency supply flexibility in a two-echelon inventory system’, Management Science, 45, 1416–1431.
- Axsäter, S. (1990), ‘Modelling emergency lateral transshipments in inventory systems’, Management Science 36, 1329–1338.
- Enders, P. (2004), ‘Spare parts inventory control in a multi-item, two-echelon network with lateral and emergency shipments’, M.Sc. Thesis, Eindhoven University of Technology.
- Grahovac, J., Chakravarty, A. (2001), ’Sharing and lateral transshipment of inventory in a supply chain with expensive low-demand items’, Management Science 47, 579–594.
- Hartman, B.C., Dror, M. (1996), ‘Cost allocation in continuous-review inventory models’, Naval Research Logistics 43, 549–561.
- Jagers, A.A. & van Doorn, E.A. (1986), “On the continued Erlang loss function”, Operations Research Letters, 5-1.
- Karsten, F. (2006), ‘Exact optimization for two-echelon inventory systems with an aggregate mean waiting time constraint per local warehouse’, B.Sc. Thesis, Eindhoven University of Technology
- Karsten (2008), “Master Thesis Preparation II – Research Proposal”, TU/e.
- Kilpi, J., Töyli, J., and Vepsäläinen, A. (2009), Cooperative strategies for the availability service of repairable aircraft components, International Journal of Production Economics 117, 360-370.
- Kranenburg A., van Houtum G.J. (2009), ‘A new partial pooling structure for spare parts networks’, European Journal of Operations Research, To appear
- Kukreja, A., Schmidt, C.P., Miller, D.M. (2001), ‘Stocking decisions for low-usage items in a multilocation inventory system’, Management Science 47, 1371–1383.
- Kulkarni V. (1999), “Modeling, Analysis, Design, and Control of Stochastic Systems”, Springer, p. 264
- Lee, H.L. (1987), ‘A multi-echelon inventory model for repairable items with emergency lateral transshipments’, Management Science 33, 1302–1316.
- Messerli E.J. (1972), Bell System Technical Journal, 51, p. 951-953.
- Rustenburg, J.W., Van Houtum, G.J., Zijm, W.H.M. (2003), ‘Exact and approximate analysis of multi-echelon, multi-indenture spare parts systems with commonality’, In: Shantikumar, J.G., Yao, D.D., Zijm, W.H.M. (Eds.), ‘Stochastic Modeling and Optimization of Manufacturing Systems and Supply Chains’, Kluwer, Boston, Ch. 7.
- Sankaran, J. K. (1991), On Finding the Nucleolus of an N-Person Cooperative Game, International Journal of Game Theory, 19:329-338.
- Sherbrooke, C.C. (1968), ‘METRIC: a multi-echelon technique for recoverable item control’, Operations Research, 16, 122–141.
- Sherbrooke, C.C. (2004), ‘Optimal Inventory Modeling of Systems’, Kluwer, Boston.
- Slikker, M. (2007), ‘Game theory with application to supply chain management’, Lecture notes Eindhoven University of Technology.

- Smith, D.R. & Whitt, W. (1981), “Resource sharing for efficiency in traffic systems”, The Bell System Technical Journal, 60-1
- Tagaras, G. (1999), Pooling in multi-location periodic inventory distribution systems, Omega International Journal of Management Science 27, 39–59.
- Wong, H., Cattrysse, D., Van Oudheusden, D. (2005), ‘Stocking decisions for repairable spare parts pooling in a multi-hub system’, International Journal of Production Economics 93–94, 309–317.
- Wong, H., Van Houtum, G.J., Cattrysse, D. and Van Oudheusden, D. (2006), ‘Multi-item spare parts systems with lateral transshipments and waiting time constraints’, European Journal of Operational Research, 171, 1071–1093.
- Wong H., Van Oudheusden D., Cattrysse D. (2007a), ‘Cost allocation in spare parts inventory pooling’, Transportation Research Part E, 43, 370–386
- Wong H., Van Oudheusden, D. and Cattrysse, D. (2007b), 'Two-echelon multi-item spare parts systems with, emergency supply flexibility and waiting time constraints', IIE Transactions, 39:11, 1045-1057
- Young, H.P. (1994), ‘Cost Allocation’, In: Aumann, R.J., Hart, S. (Eds.), ‘Handbook of Game Theory with Economic Applications’, Elsevier, Amsterdam.
- Whitt (2002), “IEOR 6707: Lecture Notes Advanced Topics in Queuing Theory”
- Zeng G. (2003), ”Two Common Properties of the Erlang-B Function, Erlang-C Function, and Engset Blocking Function”, Mathematical and Computer Modelling, 37, 1287-1296.

Appendix 1: Literature overview of lateral transshipment models

This appendix gives a more in-depth summary of all models shown in Table 1.1. It is taken from the literature study preceding this thesis (Karsten, 2008).

Lee (1987) considers emergency lateral transshipments in a two-echelon inventory system with one central warehouse and N local warehouses. The local warehouses are supplied from a central warehouse, which in turn is supplied from an infinite source. The local warehouses are grouped into a number of disjoint demand pooling groups. Lateral transshipments are made only within each group. The local warehouses in each group are identical, i.e. they face identical failure rates (this is the only paper considered in this section that makes this assumption). When a local warehouse is out of stock and faces demand, (emergency) lateral transshipments are used to obtain the item from another local warehouse in the same pooling group that has stock on hand. The local warehouse that sources the unit issues a replenishment order to the depot to restore its inventory level up to S. If all the local warehouses in the group are out of stock, then the demand is backordered. Transshipment times are assumed to be negligible. Lee provides approximations for the fraction of demand satisfied from the stock on hand, the fraction of demand that is satisfied by lateral transshipments, and the fraction of demand that is backordered. Lee also recognizes that it is important to establish a sourcing rule for lateral transshipments when there are more than two local warehouses in a pooling group. Three sourcing rules are suggested and Lee's simulation results show only small differences between the sourcing rules:

(i) The random sourcing rule, which chooses randomly from among the local warehouses that have stock.

(ii) Choose the local warehouse with the maximum stock on hand (with ties broken by the random sourcing rule).

(iii) Choose the local warehouse with the maximum stock on hand, with ties broken by selecting the local warehouse with the smallest number of outstanding orders waiting (with remaining ties broken randomly).

Axsäter (1990) considers the exact same system as Lee. While Lee focused on modeling the outstanding orders, Axsäter emphasizes *modeling the effective demand rate* at a local warehouse in more detail. He distinguishes between two situations: when the on-hand inventory at a local warehouse is positive and when it is not positive. When the on-hand inventory is positive, the demand faced by the local warehouse equals the regular demand plus the demand from other warehouses in the same pooling group due to lateral transshipments. When the on-hand inventory is not positive, the only demand faced by the local warehouse is the backordered demand that cannot be filled by lateral transshipment. In Axsäter's model, the local warehouses in each pooling group do not have to be identical, relaxing the assumption of identical warehouses made in Lee (1987). The random sourcing rule is used. The replenishment lead times are exponentially distributed, which allows the derivation of steady-state probabilities for the on-hand inventory. Simulation results indicate that Axsäter's model gives smaller errors than Lee's model.

Alfredsson & Verrijdt (1999) consider a two-echelon inventory system that allows *(direct) emergency shipments*. If demand at a local warehouse cannot be met by either stock at that local warehouse or stock at another local warehouse, an emergency shipment from the central warehouse is made (which is a faster transportation mode than regular replenishment). If the central warehouse is out of stock as well, an emergency shipment from an infinite source is made. With these emergency shipments, no customer demand is backordered. Furthermore, they extend the models of Axsäter and Lee by relaxing the assumption of negligible lateral transshipment times. They assume identical *lateral transshipment times* between local warehouses and assume that the customer initiating a lateral transshipment will continue to wait for this item, although the local warehouse could receive items earlier through normal replenishments while the customer is waiting. They allow non-identical local warehouses and assume that all local warehouses form *one pooling group* (the assumption of one pooling group is also made by all subsequent papers). They assume exponential lead times and show simulation results that indicate that the performance of the inventory system is insensitive to the lead time distribution. Using Markov analysis, they use a two-step procedure to first find estimates of the fraction of demand satisfied through emergency shipment from the central warehouse, the fraction of demand satisfied through emergency shipment from the infinite source, and the average delay at the central warehouse due to stock-outs. In the second step, they find estimates for the fraction of demand for each local warehouse satisfied from stock on hand and through lateral transshipment.

Kukreja et al. (2001) relax the assumption of an exponential repair time distribution made in Axsäter (1990). They show that the service performance is not very sensitive to the type of *repair time distribution*. They also introduce a new sourcing rule: from the locations that have inventory on hand, transship from that location with the lowest transshipment cost to the location needing the unit. If transshipment costs are linearly related to the distance between two locations, this corresponds to a *closest neighbor rule*.

Grahovac and Chakravarty (2001) investigate a two-echelon spare parts system with *pro-active lateral transshipments*. Retailer orders can be either regular or emergency. Retailers place regular orders as long as their net inventory on hand is above a trigger level *K*. They place emergency orders when it becomes equal to or smaller than *K*. Emergency orders will be delivered from a randomly selected location at the lower echelon if the distributor is out of stock.

Wong et al. (2005a) present a model for a multi-hub, multi-company system. The *multi-hub setting* is applicable for machines that are not standing on a fixed location but rather move from one location (or hub) to another, like airplanes. This implies that part failures can happen randomly at any location. Furthermore, they extend the existing lateral transshipment models by allowing *delayed lateral transshipments*. When a location having no backorders receives a repaired part and at the same time at least one location in the pooling group has backorders, their model allows sending the repaired part to the location having backorders. To find the optimal stocking levels, a two-stage solution is proposed. In the first stage, the demands at all hubs are aggregated and treated as if occurring at a single location. The optimal number of total spare parts is determined by minimizing the sum of inventory holding cost and downtime cost. In the second stage, a heuristic procedure is developed to find the optimal allocation of the total spare parts to minimize the total transshipment cost.

Wong et al. (2006) consider a single-echelon, multi-item, two-location spare parts inventory system in which emergency shipments are allowed as response to stock-outs. They focus on minimizing the expected total system cost subject to a target level for the aggregate waiting time at each location. By using a *multi-item perspective*, they can use a system approach. Their solution procedure is based on Lagrangian relaxation. A greedy heuristic may also be used to find near-optimal base stock levels (see Karsten, 2006). It is acknowledged that companies can cooperate via lateral transshipments both by (a) using the lateral transshipments in daily practice and (b) by incorporating it in the initial stocking decision.

Wong et al. (2007a) consider a single-echelon, single-item system with transshipments between local warehouses. If all the local warehouses are out of stock, then the demand is backordered. *Partial pooling* is used. Each company sets its reserved stock level and will only supply a lateral transshipment if its current on-hand inventory level is above its reserved stock level. Complete pooling and no pooling can be seen as the case with a reserved stock level of zero and a reserved stock level of S+1, respectively. Furthermore, the assumption of infinite sources of failures (which has been made in all previously mentioned papers) is violated. Instead, a *finite number of machines that can fail* are assumed, which is more reasonable for companies with a small number of machines. Moreover, they also use game theoretic models to analyze the cost allocation problem between individual companies.

Wong et al. (2007b) tackle a *two-echelon*, multi-item setting in which emergency shipments are allowed as response to stock-outs. They focus on minimizing the expected total system cost subject to a target level for the aggregate waiting time at each location. They compared the performance of a two-echelon system to a single-echelon system in terms of total cost. A main finding is that a two-echelon system is only worth implementing when lateral transshipments between local warehouses are not possible. In systems with lateral transshipments, the policy of implementing a two-echelon system should be questioned, since its total cost is merely a little lower than the cost of a single-echelon system, and the saving may eventually be offset by all additional management costs needed for the central warehouse.

Kranenburg and Van Houtum (2009) introduce a distinction between *main and regular local warehouses*. Lateral transshipment is allowed from main local warehouses only. A practical advantage of this structure is that only a limited number of local warehouses has to be equipped to provide lateral transshipment, and it is a network structure that matches with observations in practice. They show that only a small number of main locals is sufficient to obtain most of the full pooling benefits.

Appendix 2: Game Theory

In this Appendix, we give all balancedness conditions for games with 3 or 4 players. Furthermore, the nucleolus and Shapley value will be explained in more detail.

<u>Balancedness</u>

Balancedness was defined already in Section 1.3. We will now give all balancedness conditions for games with 3 or 4 players.

For a game with $|N|=3$, if and only if all of the following conditions are satisfied, the core of the game is non-empty:
$c(\{1,2,3\})\leq c(\{1\})+c(\{2\})+c(\{3\})$
$c(\{1,2,3\})\leq 0.5c(\{1,2\})+0.5c(\{1,3\})+0.5c(\{2,3\})$
$c(\{1,2,3\})\leq c(\{3\})+c(\{1,2\})$
$c(\{1,2,3\})\leq c(\{2\})+c(\{1,3\})$
$c(\{1,2,3\})\leq c(\{1\})+c(\{2,3\})$

For a game with $|N|=4$, if and only if all of the following conditions are satisfied, the core of the game is non-empty:

$c(\{1,2,3,4\})\leq c(\{1,2\})+c(\{3,4\})$; $c(\{1,2,3,4\})\leq c(\{1,3\})+c(\{2,4\})$
$c(\{1,2,3,4\})\leq c(\{1,4\})+c(\{2,3\})$

$c(\{1,2,3,4\})\leq c(\{1,2,3\})+c(\{4\})$; $c(\{1,2,3,4\})\leq c(\{1,2,4\})+c(\{3\})$
$c(\{1,2,3,4\})\leq c(\{1,4,3\})+c(\{2\})$; $c(\{1,2,3,4\})\leq c(\{4,2,3\})+c(\{1\})$

$c(\{1,2,3,4\})\leq c(\{1,2\})+c(\{3\})+c(\{4\})$; $c(\{1,2,3,4\})\leq c(\{1,3\})+c(\{2\})+c(\{4\})$
$c(\{1,2,3,4\})\leq c(\{1,4\})+c(\{2\})+c(\{3\})$; $c(\{1,2,3,4\})\leq c(\{2,3\})+c(\{1\})+c(\{4\})$
$c(\{1,2,3,4\})\leq c(\{2,4\})+c(\{1\})+c(\{3\})$; $c(\{1,2,3,4\})\leq c(\{3,4\})+c(\{1\})+c(\{2\})$

$c(\{1,2,3,4\})\leq .5c(\{1,2,3\})+.5c(\{1,2,4\})+.5c(\{3,4\})$
$c(\{1,2,3,4\})\leq .5c(\{1,3,4\})+.5c(\{2,3,4\})+.5c(\{1,2\})$
$c(\{1,2,3,4\})\leq .5c(\{2,4,1\})+.5c(\{2,4,3\})+.5c(\{1,3\})$
$c(\{1,2,3,4\})\leq .5c(\{2,3,1\})+.5c(\{2,3,4\})+.5c(\{1,4\})$
$c(\{1,2,3,4\})\leq .5c(\{1,4,2\})+.5c(\{1,4,3\})+.5c(\{2,3\})$
$c(\{1,2,3,4\})\leq .5c(\{1,3,2\})+.5c(\{1,3,4\})+.5c(\{2,4\})$

$c(\{1,2,3,4\})\leq c(\{1\})+c(\{2\})+c(\{3\})+c(\{4\})$
$c(\{1,2,3,4\})\leq 1/3c(\{1,2,3\})+1/3c(\{1,2,4\})+1/3c(\{1,3,4\})+1/3c(\{2,3,4\})$

$c(\{1,2,3,4\})\leq .5c(\{1,2\})+.5c(\{1,3\})+.5c(\{2,3\})+c(\{4\})$
$c(\{1,2,3,4\})\leq .5c(\{1,2\})+.5c(\{1,4\})+.5c(\{2,4\})+c(\{3\})$
$c(\{1,2,3,4\})\leq .5c(\{1,4\})+.5c(\{1,3\})+.5c(\{4,3\})+c(\{2\})$
$c(\{1,2,3,4\})\leq .5c(\{4,2\})+.5c(\{4,3\})+.5c(\{2,3\})+c(\{1\})$

$c(\{1,2,3,4\}) \leq .5c(\{1,2,3\}) + .5c(\{1,4\}) + .5c(\{2,4\}) + .5c(\{3\})$
$c(\{1,2,3,4\}) \leq .5c(\{1,4,3\}) + .5c(\{1,2\}) + .5c(\{2,4\}) + .5c(\{3\})$
$c(\{1,2,3,4\}) \leq .5c(\{2,4,3\}) + .5c(\{1,2\}) + .5c(\{1,4\}) + .5c(\{3\})$
$c(\{1,2,3,4\}) \leq .5c(\{1,2,4\}) + .5c(\{3,1\}) + .5c(\{3,2\}) + .5c(\{4\})$
$c(\{1,2,3,4\}) \leq .5c(\{1,3,4\}) + .5c(\{2,1\}) + .5c(\{2,3\}) + .5c(\{4\})$
$c(\{1,2,3,4\}) \leq .5c(\{2,3,4\}) + .5c(\{1,2\}) + .5c(\{1,3\}) + .5c(\{4\})$
$c(\{1,2,3,4\}) \leq .5c(\{1,2,3\}) + .5c(\{2,4\}) + .5c(\{3,4\}) + .5c(\{1\})$
$c(\{1,2,3,4\}) \leq .5c(\{1,2,4\}) + .5c(\{2,3\}) + .5c(\{3,4\}) + .5c(\{1\})$
$c(\{1,2,3,4\}) \leq .5c(\{1,3,4\}) + .5c(\{2,3\}) + .5c(\{2,4\}) + .5c(\{1\})$
$c(\{1,2,3,4\}) \leq .5c(\{2,1,3\}) + .5c(\{1,4\}) + .5c(\{3,4\}) + .5c(\{2\})$
$c(\{1,2,3,4\}) \leq .5c(\{2,1,4\}) + .5c(\{3,1\}) + .5c(\{3,4\}) + .5c(\{2\})$
$c(\{1,2,3,4\}) \leq .5c(\{2,3,4\}) + .5c(\{1,3\}) + .5c(\{1,4\}) + .5c(\{2\})$

$c(\{1,2,3,4\}) \leq 2/3c(\{1,2,3\}) + 1/3c(\{1,4\}) + 1/3c(\{2,4\}) + 1/3c(\{3,4\})$
$c(\{1,2,3,4\}) \leq 2/3c(\{1,2,4\}) + 1/3c(\{1,3\}) + 1/3c(\{2,3\}) + 1/3c(\{3,4\})$
$c(\{1,2,3,4\}) \leq 2/3c(\{1,3,4\}) + 1/3c(\{1,2\}) + 1/3c(\{2,3\}) + 1/3c(\{2,4\})$
$c(\{1,2,3,4\}) \leq 2/3c(\{2,3,4\}) + 1/3c(\{1,2\}) + 1/3c(\{1,3\}) + 1/3c(\{1,4\})$

Shapley value

The Shapley value $\Phi(N,c)$ is:

$$\Phi_i(N,c) = \sum_{M \subseteq N \setminus \{i\}} \frac{|M|! \cdot (|N-M|-1)!}{|N|!} \cdot \left(c(M \cup \{i\}) - c(M)\right) \text{ for all } i \in N.$$

The idea behind this method is that each member should be allocated a cost equal to the average contribution it makes to each coalition to which it could belong, where all coalitions are regarded as equally likely (Wong et al. 2007). It has the following properties (Slikker, 2007):

Efficiency: For each coalitional game (N,c): $\sum_{i \in N} \Phi_i = c(N)$.
Additive: For two coalitional games (N,c) and (N,d) with the same player set: $\Phi(N, c+d) = \Phi(N,c) + \Phi(N,d)$.
Symmetric: For each coalitional game (N,c) for any two players $i,j \in N$ that are symmetric in (N,c)[22]: $\Phi_i(N,c) = \Phi_j(N,c)$.
Zero-player property: For each coalitional game (N,c): $\Phi_i(N,c) = 0$ for any player $i \in N$ for which $c(M \cup \{i\}) = c(M)$ for all $M \subseteq N$.

[22] Players i and j are symmetric in game (N,c) if $c(M \cup \{i\}) = c(M \cup \{j\})$ for all $M \subseteq N \setminus \{i,j\}$

The nucleolus (by Slikker, 2007):

The nucleolus is defined for games with a nonempty imputation set only. In order to define the nucleolus we first need the concepts of ‘ordering function’ and ‘lexicographic order’. If K is a finite set then the ordering function on $\mathbb{R}^K$ is the function $\eta^K : \mathbb{R}^K \rightarrow \mathbb{R}^{|K|}$, defined by the following subsequent steps: $\eta_1^K(x) = \min\{x_j \mid j \in K\}$. Choosing $j_1 \in K$ such that $\eta_1^K(x) = x_{j_1}$, we have $\eta_2^K(x) = \min\{x_j \mid j \in K \setminus \{j_1\}\}$, etc. For $x, y \in \mathbb{R}^{|K|}$ we say that x is lexicographically larger than y if there exists an s such that x_i=y_i for all i<s and x_s>y.

Now the nucleolus is defined as follows. For a payoff vector $x \in \mathbb{R}^N$ define the satisfaction of coalition $M \subseteq N$ as: $s(M, x) = \sum_{i \in M} c(M) - x$. Let $\theta(x)$ have the satisfactions of payoff vector x ordered increasingly, i.e. $\theta(x) = \eta^{2^N}((s(M, x))_{M \subseteq N})$. Then the nucleolus $v(N,c)$ is defined, if the imputation set is not empty, as the vector in the imputation set whose θ is lexicographically maximal.

Appendix 3: Justification of assumptions

In this appendix, we defend the assumptions made in Section 3.1.

Demand process

(i) Failures occur according to independent Poisson processes with constant rate (i.e. there is an infinite source of failures). If a part fails, it is replaced with a spare part if one is on stock at the corresponding company.

For many real-life complex technical systems, lifetimes of components are (close-to) exponential, so a Poisson failure process is reasonable. A company typically has an amount of expensive technical systems (such as airplanes) that operate at a constant rate, and whose failures are low in general. The assumption of constant Poisson failure rates for the whole set of technical systems at a company is standard in METRIC-type models. In practice this assumption may be violated, since when the number of systems down increases, then the number of items that may fail decreases and hence the demand rate will slow down. However, we do not allow long downtimes and particularly if the number of systems is large, it is reasonable to assume constant failure rates.

(ii) We consider only one type of spare part (i.e. a single-item model).

This assumption is made to simplify the analysis. A multi-item approach could make the model and cooperation process richer, but this is left as a future research direction.

Cooperation process

(iii) If no spare part is available when the part fails, but another cooperating company has a surplus on-hand inventory, a lateral transshipment is used. A lateral transshipment is used from the neighbor that leads to the lowest transshipment costs (with ties broken by sourcing from the neighbor with largest stock on-hand).

This transshipment rule, a combination of rules found in the literature (see appendix 1), makes good use of available state information while not overly complicating the analysis.

(iv) Complete pooling is applied.

We assume that the companies cannot use partial pooling as in Wong et al. (2007a) in order to simplify the analysis. They show in a numerical experiment that in the cooperative optimal solution, all companies agree to use complete pooling anyway.

Repair process

(v) A failed part is immediately sent into repair (therefore, the inventory system at one company can be seen as being controlled by a base stock policy).

An (S_i-1,S_i) policy is reasonable for recoverable items with high cost and low demand, as the fixed ordering costs are small relative to the price of the item.

(vi) Repair lead times are exponential i.i.d..

This assumption facilitates the analysis. It allows us to use a Markov process in the evaluation. Simulations by Alfredsson and Verrijdt (1999) and Kukreja et al. (2001) justify the assumption, as they show that the type of lead time distribution does not have a large effect on steady-state distributions. The repair lead times are i.i.d. variables, as there is assumed to be ample repair capacity and no queuing.

(vii) Repaired parts are returned to the company that fulfilled the demand for the part.
This allows us to keep the inventory positions at constant levels at all companies.
(viii) All parts are perfectly repairable and there is no condemnation.
If not, then one may relax this assumption by assuming that a new part is procured in case repair is not possible for a failed part.

Emergency supply:
(ix) If none of the other warehouses has a part available, an emergency supply is instigated from an outside infinite source and the system goes down until it arrives.
There is no redundancy or cannibalization in our model; the spare part is critical, so the system fails due to precisely one part. If not, then it is still reasonable to assume that a failed part has a bad effect on the performance of complex machine, so there may still be 'partial' downtime costs. The emergency supplier being an infinite source is an assumption often made in related literature (see Table 1.1) and facilitates the analysis.
(x) It is assumed that the emergency costs are smaller than the expected downtime costs during a repair, but larger than the total lateral costs.
If the former would not be the case, emergency shipments would not be economically rational while our analysis "forces" them. If the latter would not be the case, lateral transshipments would not be worthwhile, while our analysis "forces" them.
(xi) Failed part is lost to the emergency supplier; does not return to the inventory system.
If not, total inventory in the system would keep on growing, complicating the analysis.

Cost parameters:
The cost parameters chosen encompass the costs analyzed in related studies (see Appendix 5).

Goal of individual companies:
(xii) Each company aims to minimize expected yearly costs. We have an infinite horizon.
This allows us to use steady-state Markov chains. Spare parts for complex machines typically have long lifetimes, so a very long horizon is not unreasonable.

Appendix 4: Classes of spare parts inventory situations

Class of simple spare parts inv. sit. (with identical λ): $\Gamma_{simple,id:\lambda} \subset \Gamma_{simple}$

For every $\varphi_{simple,id:\lambda} \in \Gamma_{simple,id:\lambda}$, the demand rates are assumed to be identical for all companies. It can be represented a tuple: $\varphi_{simple,id:\lambda} = \left(N, \lambda, \mu, (h_i)_{i\in N}, (c_i^{emer})_{i\in N}\right)$.

Class of simple spare parts inv. sit. (with identical h): $\Gamma_{simple,id:h} \subset \Gamma_{simple}$

For every $\varphi_{simple,id:h} \in \Gamma_{simple,id:h}$, the holding cost rates are assumed to be identical for all companies. It can be represented a tuple: $\varphi_{simple,id:h} = \left(N, (\lambda_i)_{i\in N}, \mu, h, (c_i^{emer})_{i\in N}\right)$.

Class of simple spare parts inv. sit. (with id. λ and c^{emer}): $\Gamma_{simple,id:\lambda,c^{emer}} \subset \Gamma_{simple}$

For every $\varphi_{simple,id:\lambda,c^{emer}} \in \Gamma_{simple,id:\lambda,c^{emer}}$, the demand rates and emergency costs are assumed to be identical for all companies. It can be represented a tuple: $\varphi_{simple,id:\lambda,c^{emer}} = \left(N, \lambda, \mu, (h_i)_{i\in N}, c^{emer}\right)$.

Class of simple spare parts inv. sit. (with id. h and c^{eme}): $\Gamma_{simple,id:h,c^{emer}} \subset \Gamma_{simple}$

For every $\varphi_{simple,id:h,c^{emer}} \in \Gamma_{simple,id:h,c^{emer}}$, the holding cost rates and emergency costs are assumed to be identical for all companies. It can be represented a tuple: $\varphi_{simple,id:h,c^{emer}} = \left(N, (\lambda_i)_{i\in N}, \mu, h, c^{emer}\right)$.

Class of simple spare parts inv. sit. (with ident. λ and h): $\Gamma_{simple,id:\lambda,h} \subset \Gamma_{simple}$

For every $\varphi_{simple,id:\lambda,h} \in \Gamma_{simple,id:\lambda,h}$, the demand rates and holding cost rates are assumed to be identical for all companies. It can be represented a tuple: $\varphi_{simple,id:\lambda,c^{emer}} = \left(N, \lambda, \mu, h, (c_i^{emer})_i)_{i\in N}\right)$.

Class of simple fully identical 3-player spare parts inv. sit.: $\Gamma_{simple,id:all,|N|=3} \subset \Gamma_{simple,id:all}$

For every $\varphi_{simple,id:all,|N|=3} \in \Gamma_{simple,id:all,|N|=3}$, the set of companies is {1,2,3}. It can be represented by a tuple: $\varphi_{simple,id:all,|N|=3} = \left(\lambda, \mu, h, c^{emer}\right)$.

Appendix 5: Realistic parameter values for spare parts inventory situations

Source	λ (per year)	λ /μ	h ($ per unit per year)	downtime costs ($ per unit down per hour)	transport costs ($ per hour)	emer. supply ordering costs ($)	Time needed for emer. ship. (hour)	transship ordering costs ($)	Time needed for lat. trans-ship. (hour)	Notes
Wong et al. (2007a)	Min: 0.002·2·356≈1.4 (Section 5.2) Max: 0.002·15·356≈10.7 (Section 5.2)	Min: 1.4/(0.02·356)≈ 0.20 (Sect 5.2) Max: 10.7/(0.02·356)≈ 1.51 (Sect 5.2)	Min: 10,000 (Section 5.2) Max: 20,000 (Section 4.2)	Min: 1,000/24≈42 (Section 5.2) Max: 10,000/24≈420 (Section 5.2)	Min: 100 (Sect 5.2) Max: 1,000 (Sect 5.2)	N/A	N/A	N/A	Min: 2 (Section 4.2) Max: 12 (Section 4.2)	These parameters were selected such that they are realistic for real-life situations, at least for the airline industry. Demand rates may differ a factor 5 between companies; downtime costs a factor 2 between companies; otherwise companies are identical.
Wong et al. (2006)	Min: 0.0029·356 ≈ 1.03 (Table 4) Max: 0.0457·356=16.27 (Table 4)	Min: 0.0029/0.0217≈0.13 (Table 4) Max: 0.0457/0.0164≈2.79 (Table 4)	Min: 25000·0.2=5000 (Table 4) Max: 143450·0.2≈28690 (Table 4)	N/A	50 (Section 5)	500 (Section 5)	24 (Section 5)	N/A	2 (Section 5)	Section 5 and Table 4: Sample data from an air carrier company located in Brussels. The focus was on systems where the inventory holding cost is dominant in comparison to the lateral transshipment or emergency supply cost. (which should be about 5-10 times higher than lateral transshipment costs).
Kukreja et al. (2001)	Min: 1 (Table 2) Max: 6 (Table 2)	Min: 1/(356/(5·7)) ≈0.098 (Table 2) Max: 6/(356/(15·7)) ≈1.77 (Table 2)	Min: 500·0.29=145 (Section 6 & Table 2) Max: 3000·0.29=870 (Section 6 & Table 2)	N/A	N/A	N/A	N/A	Min: 100 (Table 2) Max: 600 (Table 2)	N/A	Data originates from a large electric utility company. Demand rates may differ a factor 6 between companies; otherwise they are identical.
Enders (2004)	Min: 0.0017·356≈0.61 (Section 4.8.2) Max: 0.27·356≈96 (Section 4.8.2)	Min: 0.61/(356/7)≈ 0.012 (Sect 4.8.2) Max: 96/(356/7) ≈ 1.89 (Sect 4.8.2)	Min: 0.25·1500=375 (Section 4.8.1-2) Max: 0.25·1900=475 (Section 4.8.1-2)	N/A	N/A	Min: 500 (Table 7.4) Max: 1000 (Table 7.4)	24 (Section 4.8.1)	300 (Table 7.4)	Min: 0.25·24=6 (Sect 4.8.1) Max: 0.5·24=12 (Sect 4.8.1)	Data stems from ASML. Demand rates may differ a factor 4 between companies (Section 4.8.2).
Kranenburg & van Houtum (2009)	Min: 0.5 (Table 1) Max: 50 (Table 1)	Min: 0.5/(356/14)=0.002 (Table 1) Max: 50/ (356/14) =2 (Table 1)	Min: 2000·0.25=500 (Section 6) Max: 100,000·0.25=25,000 (Section 6)	N/A	N/A	1000 (Section 7)	2·24=48 (Section 7)	500 (Section 7)	0.5·24=12 (Section 7)	Data obtained from ASML. This data set constitutes data for all 19 local warehouses in the USA.
SELEC-TED VALUE	Min: 0.5 Max: 50	Min: 0.01 Max: 3 (Min and max for μ obtained by fixing λ=5; a "standard" value)	Min: 400 Max: 28000	c^{emer}= Time needed for emer. ship · (transport costs + downtime costs) + emer. supply ordering costs. Minimum for c^{emer}: 24 · (42 + 50) + 500 ≈ 2600. Maximum for c^{emer}: 48 · (420 + 1000) + 1000 ≈ 78000.				c^{trans}= Time needed for lat. trans-ship. · (transport costs + downtime costs) + transship ordering costs. Minimum : 2 · (42 + 50) + 100 ≈ 300. Maximum: 12 · (420 + 1000) + 600 ≈ 18000.		

Appendix 6: The Erlang loss function is decreasing in multiplication.

In this appendix we proof that the Erlang loss function is decreasing in multiplication, i.e. $\pi_0(ax,a) \le \pi_0(bx,b)$, for $a,b \in \mathbb{N}_0$ $a \ge b, x > 0$. This was stated in Lemma 4.2 and had already been proven (Schmidt & Whitt, 1981), but a different proof methodology can also be used.

Lemma A6: $\pi_0(ax,a) \le \pi_0(bx,b)$, for $a,b \in \mathbb{N}_0$ $a \ge b, x > 0$.

Proof:

For *a*=*b*, the proof is trivial and boils down to an equality.
For *b*=0, $\pi_0(bx,b) = 1$. Furthermore, $\pi_0(ax,a) \le 1$ since it is a probability.
Thus, it suffices to show $\pi_0(ax,a) \le \pi_0(bx,b)$ for *b*≥1 and *a*>*b*.

Let *a* and *b* be positive integers with *a*>*b*. Furthermore, let $i \in \{0,1,...,b\}$ be fixed for the moment. Then the following equation holds:

$$(a-b) \ge (a/b - 1) \cdot i \tag{T.1}$$

Add *i* to both sides, then divide both sides by (*a*/*b*) to obtain:

$$\frac{i + (a-b)}{a/b} \ge i \tag{T.2}$$

As (T.2) holds for all *i*∈{0,1,…,*b*}, for all *y*∈{0,1,…,*b*-1} it holds that:

$$\prod_{i=y+1}^{b} \frac{i+(a-b)}{a/b} \ge \prod_{i=y+1}^{b} i \tag{T.3}$$

Relabling the left product range and taking (*a*/*b*) out of the product shows that (T.3) is equivalent to:

$$\frac{\prod_{i=y+1+a-b}^{a} i}{(a/b)^{b-y}} \ge \prod_{i=y+1}^{b} i \tag{T.4}$$

Rewriting the products as a division of factorials shows that (T.3) is also equivalent to, for all *y* in {0,1,…,*b*-1}:

$$\frac{a!}{(a/b)^{b-y} \cdot (y+a-b)!} \ge \frac{b!}{y!} \tag{T.5}$$

Note that for y= *b*, (T.5) corresponds to 1≥1. Hence (T.5) holds not only for all *y* in {0,1,…,*b*-1}, but moreover for all *y* in {0,1,…,*b*}.

Let x>0 and let y∈{0,1,…,b} be fixed for the moment. By multiplying (T.5) with x^y and by dividing both sides by b^{b-y}, it can be seen that it holds that:

$$\frac{a!\cdot x^y}{a^{b-y}\cdot(y+a-b)!}\geq\frac{b!\cdot x^y}{b^{b-y}\cdot y!} \tag{T.6}$$

As (T.6) holds for all y∈{0,1,…,b}, then it also holds that:

$$\sum_{y=0}^{b}\frac{a!\cdot x^y}{a^{b-y}\cdot(y+a-b)!}\geq\sum_{y=0}^{b}\frac{b!\cdot x^y}{b^{b-y}\cdot y!} \tag{T.7}$$

Relabling the left sum range shows that (T.7) is equivalent to:

$$\sum_{y=a-b}^{a}\frac{a!\cdot x^{y-(a-b)}}{a^{a-y}\cdot y!}\geq\sum_{y=0}^{b}\frac{b!\cdot x^y}{b^{b-y}\cdot y!} \tag{T.8}$$

Adding non-negative terms to the left side gives:

$$\sum_{y=0}^{a}\frac{a!\cdot x^{y-(a-b)}}{a^{a-y}\cdot y!}\geq\sum_{y=0}^{b}\frac{b!\cdot x^y}{b^{b-y}\cdot y!} \tag{T.9}$$

Rearrange exponents to see that (T.9) is equivalent to:

$$\sum_{y=0}^{a}\frac{a!\cdot(ax)^y}{a^a\cdot x^{(a-b)}\cdot y!}\geq\sum_{y=0}^{b}\frac{b!\cdot(bx)^y}{b^b\cdot y!} \tag{T.10}$$

Move the exponents and factorials that are independent of y outside the summations:

$$\frac{a!}{a^a\cdot x^{(a-b)}}\cdot\sum_{y=0}^{a}\frac{(ax)^y}{y!}\geq\frac{b!}{b^b}\cdot\sum_{y=0}^{b}\frac{(bx)^y}{y!} \tag{T.11}$$

Raise both sides to the power (-1), then multiply both sides by x^b to obtain:

$$\frac{(ax)^a}{a!\cdot\sum_{y=0}^{a}\frac{(ax)^y}{y!}}\leq\frac{(bx)^b}{b!\cdot\sum_{y=0}^{b}\frac{(bx)^y}{y!}} \tag{T.12}$$

(T.12) is equivalent to $\pi_0(ax,a)\leq\pi_0(bx,b)$. This completes the proof.

■

Appendix 7: Proof to Lemma 5.4

Lemma 5.4: Let x>0 and let $z \in \{1,3,5,...,37,39\}$. Then:

$$\frac{1}{2}\pi_0(3x, 1.5\cdot z - 0.5) + \frac{1}{2}\pi_0(3x, 1.5\cdot z + 0.5) \le \pi_0(2x, z).$$

Proof:

Let:

$$j(y) = 2^z \cdot f(y-z)$$

$$f(y) = \begin{cases} \sum_{v=0}^{y} \frac{3^v}{v!}\cdot\frac{3^{y-v}}{(y-v)!} & 0 \le y \le 1.5z - 0.5 \\ \sum_{v=y-(1.5z+0.5)}^{(1.5z-0.5)} \frac{3^v}{v!}\cdot\frac{3^{y-v}}{(y-v)!} & 1.5z + 0.5 \le y \le 3z \end{cases}$$

$$k(y) = \begin{cases} 0 & , z \le y < 1.5z - 0.5 \\ \frac{z! \cdot 3^{(1.5\cdot z - 0.5)}}{2\cdot(1.5\cdot z - 0.5)!} & , y = 1.5z - 0.5 \\ \frac{z! \cdot 3^{(1.5\cdot z - 0.5)}}{2\cdot(1.5\cdot z - 0.5)!}\cdot g(y-(1.5z-0.5)) + \frac{z! \cdot 3^{(1.5\cdot z + 0.5)}}{2\cdot(1.5\cdot z + 0.5)!}\cdot h(y-(1.5z+0.5)) & , y \ge 1.5z + 0.5 \end{cases}$$

$$g(y) = \begin{cases} \sum_{v=0}^{y} \frac{2^v}{v!}\cdot\frac{3^{y-v}}{(y-v)!} & 0 \le y \le z \\ \sum_{v=0}^{z} \frac{2^v}{v!}\cdot\frac{3^{y-v}}{(y-v)!} & z \le y \le 1.5z + 0.5 \\ \sum_{v=y-(1.5z+0.5)}^{z} \frac{2^v}{v!}\cdot\frac{3^{y-v}}{(y-v)!} & 1.5z + 0.5 \le y \le 2.5z + 0.5 \end{cases}$$

$$h(y) = \begin{cases} \sum_{v=0}^{y} \frac{2^v}{v!}\cdot\frac{3^{y-v}}{(y-v)!} & 0 \le y \le z \\ \sum_{v=0}^{z} \frac{2^v}{v!}\cdot\frac{3^{y-v}}{(y-v)!} & z \le y \le 1.5z - 0.5 \\ \sum_{v=y-(1.5z-0.5)}^{z} \frac{2^v}{v!}\cdot\frac{3^{y-v}}{(y-v)!} & 1.5z - 0.5 \le y \le 2.5z - 0.5 \end{cases}$$

For all $z \in \{1,3,5,...,37,39\}$ and for all $y \in \{z, z+1, ..., 4z\}$, it holds that: $k(y) \le j(y)$. This has been numerically verified and can be easily checked.

Therefore it also holds that:

$$\sum_{y=z}^{4z} k(y)x^y \le \sum_{y=z}^{4z} j(y)x^y \qquad \text{(A.1)}$$

(A.1) is equivalent to:

$$\frac{1}{2}\cdot z!\cdot\frac{(3x)^{(1.5\cdot z-0.5)}}{(1.5\cdot z-0.5)!}\cdot\left(\sum_{y=0}^{2.5z+0.5} g(y)x^y\right)+\frac{1}{2}\cdot z!\cdot\frac{(3x)^{(1.5\cdot z+0.5)}}{(1.5\cdot z+0.5)!}\cdot\left(\sum_{y=0}^{2.5z-0.5} h(y)x^y\right) \le (2x)^z\cdot\sum_{y=0}^{(3z)} f(y)x^y \qquad (A.2)$$

By Lemma A7, the following holds:

$\left(\sum_{y=0}^{z}\frac{(2x)^y}{y!}\cdot\sum_{y=0}^{(1.5\cdot z-0.5)}\frac{(3x)^y}{y!}\right)$ is equal to: $\sum_{y=0}^{2.5z-0.5} h(y)x^y$,

$\left(\sum_{y=0}^{z}\frac{(2x)^y}{y!}\cdot\sum_{y=0}^{(1.5\cdot z+0.5)}\frac{(3x)^y}{y!}\right)$ is equal to: $\sum_{y=0}^{2.5z+0.5} g(y)x^y$,

$\left(\sum_{y=0}^{(1.5\cdot z-0.5)}\frac{(3x)^y}{y!}\cdot\sum_{y=0}^{(1.5\cdot z+0.5)}\frac{(3x)^y}{y!}\right)\cdot(2x)^z$ is equal to: $(2x)^z\cdot\sum_{y=0}^{(3z)} f(y)x^y$.

Thefore, (A.2) is identical to:

$$\frac{1}{2}\cdot z!\cdot\frac{(3x)^{(1.5\cdot z-0.5)}}{(1.5\cdot z-0.5)!}\cdot\left(\sum_{y=0}^{z}\frac{(2x)^y}{y!}\cdot\sum_{y=0}^{(1.5\cdot z+0.5)}\frac{(3x)^y}{y!}\right)+ \frac{1}{2}\cdot z!\cdot\frac{(3x)^{(1.5\cdot z+0.5)}}{(1.5\cdot z+0.5)!}\cdot\left(\sum_{y=0}^{z}\frac{(2x)^y}{y!}\cdot\sum_{y=0}^{(1.5\cdot z-0.5)}\frac{(3x)^y}{y!}\right) \le \sum_{y=0}^{(1.5\cdot z-0.5)}\frac{(3x)^y}{y!}\cdot\sum_{y=0}^{(1.5\cdot z+0.5)}\frac{(3x)^y}{y!}\cdot(2x)^z \qquad (A.3)$$

Divide both sides of (A.3) by $\sum_{y=0}^{(1.5\cdot z-0.5)}\frac{(3x)^y}{y!}\cdot\sum_{y=0}^{(1.5\cdot z+0.5)}\frac{(3x)^y}{y!}\cdot z!\sum_{y=0}^{z}\frac{(2x)^y}{y!}$ to obtain:

$$\frac{1}{2}\cdot\left(\frac{\frac{(3x)^{(1.5\cdot z-0.5)}}{(1.5\cdot z-0.5)!}\cdot\sum_{y=0}^{(1.5\cdot z+0.5)}\frac{(3x)^y}{y!}+\frac{(3x)^{(1.5\cdot z+0.5)}}{(1.5\cdot z+0.5)!}\cdot\sum_{y=0}^{(1.5\cdot z-0.5)}\frac{(3x)^y}{y!}}{\sum_{y=0}^{(1.5\cdot z-0.5)}\frac{(3x)^y}{y!}\cdot\sum_{y=0}^{(1.5\cdot z+0.5)}\frac{(3x)^y}{y!}}\right)\le\left(\frac{(2x)^z}{z!\sum_{y=0}^{z}\frac{(2x)^y}{y!}}\right) \qquad (A.4)$$

(A.4) is equivalent to:

$$\frac{1}{2}\cdot\left(\frac{(3x)^{(1.5\cdot z-0.5)}/(1.5\cdot z-0.5)!}{\sum_{y=0}^{(1.5\cdot z-0.5)}(3x)^y/y!}+\frac{(3x)^{(1.5\cdot z+0.5)}/(1.5\cdot z+0.5)!}{\sum_{y=0}^{(1.5\cdot z+0.5)}(3x)^y/y!}\right)\le\left(\frac{(2x)^z/z!}{\sum_{y=0}^{z}(2x)^y/y!}\right) \qquad (A.5)$$

It is now easily seen that (A.5) is equivalent to

$\frac{1}{2}\pi_0(3x,1.5z-0.5)+\frac{1}{2}\pi_0(3x,1.5z+0.5)\le\pi_0(2x,z)$. This completes the proof.

■

Lemma A7: Let $x \geq 0$, let $A, B \in \mathbb{N}_0$ with $B \geq A$, let $\boldsymbol{a} \in \mathbb{R}^A$, and let $\boldsymbol{b} \in \mathbb{R}^B$. Then:
$\sum_{i=0}^{A} x^i \cdot a_i \cdot \sum_{i=0}^{B} x^i \cdot b_i = \sum_{i=0}^{A+B} x^i \cdot function(i)$, where:

$$function(i) = \begin{cases} \sum_{v=0}^{i} a_v \cdot b_{i-v} & 0 \leq i \leq A \\ \sum_{v=0}^{A} a_v \cdot b_{i-v} & A \leq i \leq B \\ \sum_{v=i-B}^{A} a_v \cdot b_{i-v} & B \leq i \leq A+B \end{cases}$$

Proof:

The product of summations $\sum_{i=0}^{A} x^i \cdot a_i \cdot \sum_{i=0}^{B} x^i \cdot b_i$ can be seen as:

$$\begin{gathered} \left(a_0 + a_1 x + a_2 x^2 + \ldots + a_A x^A\right) \cdot b_0 + \\ \left(a_0 + a_1 x + a_2 x^2 + \ldots + a_A x^A\right) \cdot b_1 x + \\ \left(a_0 + a_1 x + a_2 x^2 + \ldots + a_A x^A\right) \cdot b_2 x^2 + \\ \ldots \\ + \left(a_0 + a_1 x + a_2 x^2 + \ldots + a_A x^A\right) \cdot b_B x^B \end{gathered}$$

Or, equivalently:

$$\begin{aligned} &(a_0 \cdot b_0) + x \cdot (a_1 b_0 + a_0 b_1) + x_2 \cdot (a_2 b_0 + a_1 b_1 + a_0 b_2) + x^A \cdot (a_A b_0 + a_{A-1} b_1 + \ldots + a_0 b_A) + \\ &x^{A+1} \cdot (a_A b_1 + a_{A-1} b_2 + \ldots + a_0 b_{A+1}) + \ldots + x^B \cdot (a_A b_{B-A} + a_{A-1} b_{B-A+1} + \ldots + a_0 b_B) + \\ &x^{B+1} \cdot (a_A b_{B-A} + a_{A-1} b_{B-A+1} + \ldots + a_1 b_B) + \ldots + x^{B+A} \cdot (a_A b_B) \end{aligned}$$

Rewriting the above terms to $\sum_{i=0}^{A+B} x^i \cdot function(i)$ completes the proof.

■

Appendix 8: Program description

In this appendix a description is given of the computer program (and the algorithms it implemented) that was used in the numerical experiments of chapter 5 and 6. The program was written in the Java language and a copy of the code can be obtained from the author upon request.

First, however, for all $\varphi \in \Gamma_{simple}$ we will state a formal definition of $\boldsymbol{S}^*_M(\varphi)$, the unique optimal base-stock vector for coalition $M \subseteq N$ (used in the computer program as well), and of $S_M^{tot}*(\varphi)$, the unique optimal base-stock vector sum for coalition $M \subseteq N$.

Formally, let $\varphi \in \Gamma_{simple}$ and consider coalition $M \subseteq N$. Define:

- $W(\varphi, M) = \underset{S \in N_0{}^M}{ArgMin}\left(\mathrm{K}^{\varphi_{simple};S}(M)\right)$: the set of optimal base-stock vectors for M,
- $S_M^{tot}*(\varphi) = Min\left\lfloor \sum_{i \in M} w_i \mid w \in W(\varphi, M)\right\rfloor$: lowest optimal sum of base-stock levels,
- $CompMinH(\varphi, M) = \underset{i \in M}{ArgMin}(h_i)$: set of players with the lowest holding costs rates.
- $V(\varphi, M) = \left\lfloor S_M^{tot}*(\varphi) / \mid CompMinH(\varphi, M) \mid \right\rfloor$: the base-stock levels that can be evenly allocated to all companies in *CompMinH*.
- $W(\varphi, M) = S_M^{tot}*(\varphi) - \left(V(\varphi, M) \cdot \mid CompMinH(\varphi, M) \mid\right)$: the base-stock levels that remain to be allocated.
- $X(i, \varphi, M) = \begin{cases} 1 & \text{if } i \leq W(\varphi, M) \\ 0 & \text{if } i > W(\varphi, M) \end{cases}$: how the remaining base-stock levels are allocated.
- Then $\boldsymbol{S}^*_M(\varphi) = \begin{cases} 0 & \text{if } i \in N \setminus CompMinH(\varphi, M) \\ V(\varphi, M) + X(i, \varphi, M) & \text{if } i \in CompMinH(\varphi, M) \end{cases}$.

Now, we will first introduce some variables that were used in the program:

- *SPIS* is a simple spare parts inventory situation, i.e. a tuple $\left(N, (\lambda_i)_{i \in N}, \mu, (h_i)_{i \in N}, (c_i^{emer})_{i \in N}\right)$ (see Chapter 3).
- *RuleS* contains the information on the base-stock levels used in the experiment. It is a tuple (Type, index, $\mathbf{S}^{\text{indiv}}$, $\mathbf{S}^{\text{all}}$, $\mathbf{S}^{\text{high}}$, $\mathbf{S}^{\text{low}}$, $\mathbf{S}^{\text{mix}}$), where:
    - o Type=A Boolean that indicates whether the base-stock vector is fixed (type “FIX”) or whether the base-stock level is to-be-optimized (type “OPT”)
    - o index=This value is equal to 0 for type “OPT”, and equal to 1 through 5 for type “FIX”. It is equal to 1 when the base-stock vector should be set to $\mathbf{S}^{\text{indiv}}$; 2 when the base-stock vector should be set to $\mathbf{S}^{\text{all}}$; 3 when the base-stock vector should be set to $\mathbf{S}^{\text{high}}$; 4 when the base-stock vector should be set to $\mathbf{S}^{\text{low}}$; and 5 when the base-stock vector should be set to $\mathbf{S}^{\text{mix}}$.
    - o $\mathbf{S}^{\text{indiv}}$, $\mathbf{S}^{\text{all}}$, $\mathbf{S}^{\text{high}}$, $\mathbf{S}^{\text{low}}$, $\mathbf{S}^{\text{mix}}$ are base-stock vectors that are defined in section 5.4.2.

A flowchart that describes the main idea of the program is shown in figure A8.1. Afterwards, an explanation of each part of this flowchart is given.

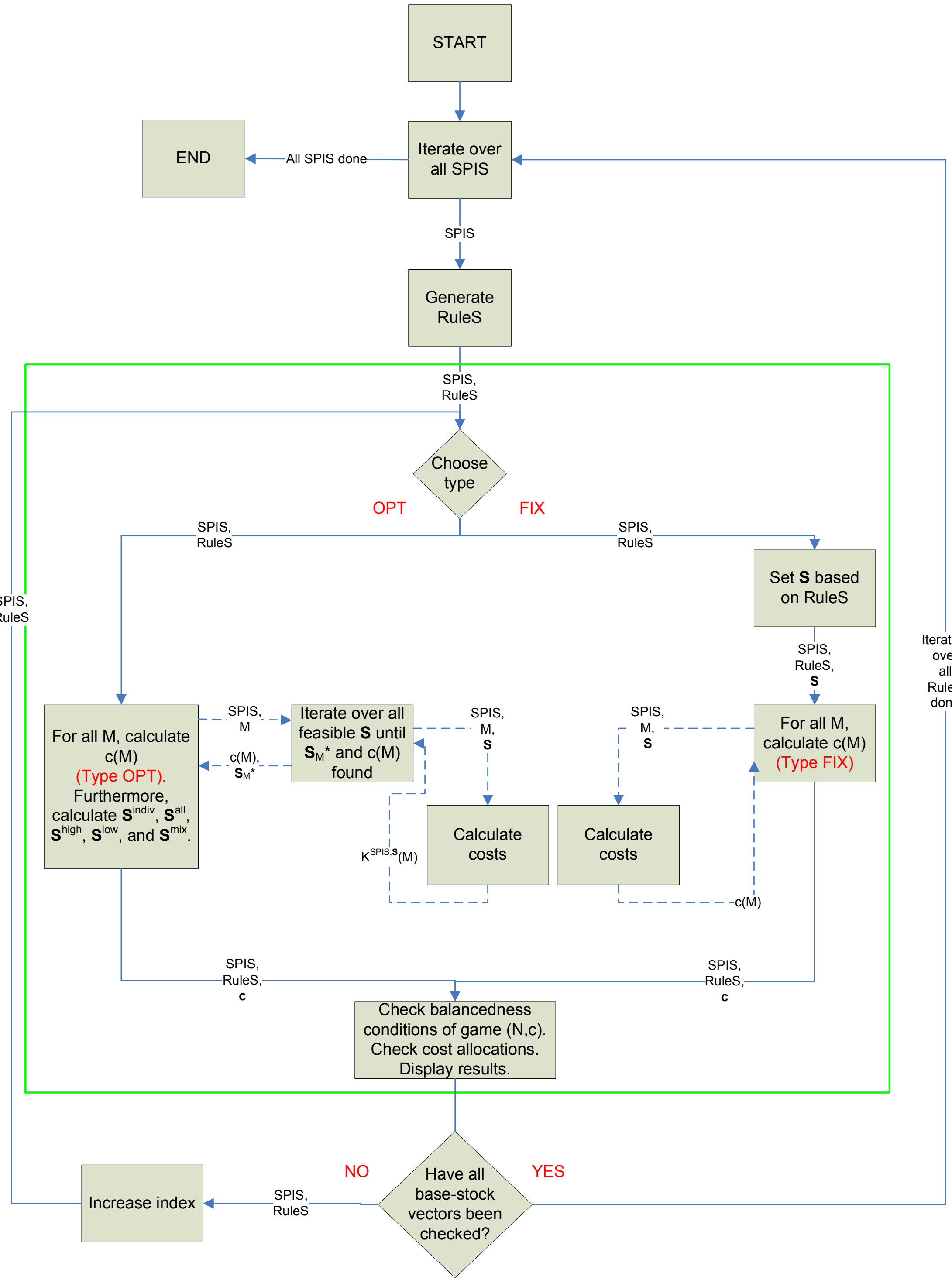
START
END
All SPIS done
Iterate over all SPIS
SPIS
Generate RuleS
SPIS, RuleS
SPIS, RuleS
Choose type
OPT
FIX
SPIS, RuleS
SPIS, RuleS
Set S based on RuleS
SPIS, RuleS, S
Iteration over all RuleS done
For all M, calculate c(M) (Type OPT). Furthermore, calculate S^indiv, S^all, S^high, S^low, and S^mix.
SPIS, M
Iterate over all feasible S until S_M* and c(M) found
c(M), S_M*
SPIS, M, S
SPIS, M, S
For all M, calculate c(M) (Type FIX)
Calculate costs
Calculate costs
K^SPIS,S(M)
c(M)
SPIS, RuleS, c
SPIS, RuleS, c
Check balancedness conditions of game (N,c). Check cost allocations. Display results.
NO
YES
Have all base-stock vectors been checked?
Increase index
SPIS, RuleS

Figure A8.1: Flowchart of the program. Boxes indicate functions or code segments. The arrows indicate the flow of the program and indicate which function is called where. The variables on the arrows represent the inputs/outputs from functions. Red text represents a choice on which branch to follow next in the program. The part within the green box represents the part of the program that generates a game (and displays results on core and cost allocations) associated with the combination of one SPIS and one (index of) RuleS.

Iterate over all SPIS
This function (based on Table 5.2) generates one SPIS and uses it to start "Generate RuleS". When all games corresponding to this SPIS and all (indices of) RuleS have been created and checked, this function generates the next SPIS and starts "Generate RuleS" again, until all SPIS have generated.

Generate RuleS
This generates a RuleS, in which it sets Type="OPT" and index=0 and leaves the rest of the information in RuleS empty for now. It then uses SPIS and RuleS to start the part of the program that generates a game (and displays results on core and cost allocations) associated with the combination of one SPIS and one RuleS.

For all M, calculate c(M) (Type OPT). Furthermore, calculate S^{indiv}, S^{all}, S^{high}, S^{low}, and S^{mix}
This function generates a cost vector $\mathbf{c}\in\mathbb{R}^{2^N\setminus\{\emptyset\}}$ (where $2^N\setminus\{\emptyset\}$ is the number of non-empty subsets of *N*) and iterates over all non-empty coalitions $M \subseteq N$. For each *M*, this function:

- Calls the function "Starting at zero, increase ***S*** until $\boldsymbol{S_M}$******* and c(*M*) found", with SPIS and *M*, which eventually returns the optimal costs c(*M*) and the optimal base-stock vector $\boldsymbol{S_M}$*******.
- If |*M*|=1 then it sets *i* to be the element of *M* and then $S_{indiv\,i} = (S^*_M)_i$.
- If *M*=*N* then it sets $\mathbf{S}^{\text{all}}$= $\boldsymbol{S_M}$*******. Furthermore, it now also calculates $\mathbf{S^{high}}$, $\mathbf{S^{low}}$, and $\mathbf{S^{mix}}$ (see section 5.4.2).

Starting at zero, increase S until S_M* and c(M) found
The algorithm used in this function requires more in-depth explanation and is broken down in consecutive steps.

<u>STEP 1:</u>
First of all, the algorithm finds the set of companies with the minimum holding cost rate in *M* (called $companiesMINh = \{i \in M \mid h_i \le h_j \forall j \in M\}$). Obviously, a solution in which any stock is held by a company that is not in $companiesMINh$ can never be optimal, since allocating that stock to a company in $companiesMINh$ instead would lead to lower holding costs and equal emergency costs. Furthermore, it is important to note that the costs are indifferent to the allocation of base stock levels over companies in $companiesMINh$, i.e. if for some spare parts inventory situation $\varphi \in \Gamma_{simple,id:\mu}$ and two base stock vectors ***S*** and ***S'*** it holds that $\sum_{i\in companiesMINh} S_i = \sum_{i\in companiesMINh} S'_i$ then $\mathrm{K}^{\varphi;S}(M) = \mathrm{K}^{\varphi;S'}(M)$. As such, we can use a simple algorithm that will only adjust the base-stock level of one company in $companiesMINh$ while the base-stock level of any other company will stay zero in order to find an optimal solution.
From $companiesMINh$, we select the company with the lowest index (i.e. company 1 rather than company 2) and refer to this company as *compMINh*.

STEP 2:
The algorithm then generates a base-stock vector ***S*** and sets the base-stock level of each company to zero. Then, the costs c(***S***) corresponding to this feasible solution when no parts are held on stock are obtained (in order to do this, it calls the function "Calculate costs" with SPIS, *M*, and ***S***) and stored in the variable *minCosts*. It may be possible to obtain lower costs by increasing the base-stock level of *compMINh*. So, $S_{compMINh}$ is increased by one and c(***S***) is obtained (once again via the function "Calculate costs", which will be described later in this appendix). If those costs are lower than *minCosts*, then *minCosts* is set to c(***S***), $S_{compMINh}$ is increased by one again and the corresponding costs are calculated again. This process repeats itself until c(***S***) is not lower than *minCosts*. Once that happens, the algorithm can stop as the optimal solution has been found: *minCosts* is the optimal costs and an optimal base-stock vector is the one that was used to obtain *minCosts*. Proof that this method truly finds the optimal costs is given in Theorem A8.1 below.
Summarizing this process formally:
<u>Step 2a</u>: Set S_i=0 for all i in M. Obtain c(**S**) and set *minCosts*=c(**S**).
<u>Step 2b</u>: Obtain c(**S**). Increase $S_{compMINh}$ by 1.
<u>Step 2c</u>: If c(**S**)<*minCosts* then set *minCosts*=c(**S**) and return to Step 2b. Otherwise, proceed to step 2d.
<u>Step 2d</u>: *minCosts* is the optimal costs of coalition *M* and *optimalTotalStock*= $S_{compMINh}$-1.

Note that during this algorithm, the base-stock level of any company other than *compMINh* remains 0 and only $S_{compMINh}$ is increased. As such, we can use formula (4.3):
$\mathrm{K}_{tot}^{\varphi_{simple};S_{tot}}(M) = h_{\min} \cdot S_{tot} + \pi_0(\rho, S_{tot}) \cdot p$, where:

- $S_{tot}=S_{compMINh}$
- $h_{min}=h_{compMINh}$
- $\rho = \dfrac{\lambda_M}{\mu}$
- $p = \sum_{i \in M} \lambda_i \cdot c_i^{emer}$

This simpler cost function will be used in Lemma A8.1 and Theorem A8.1. Lemma A8.1 states that on the domain $S_{tot} \in \mathbb{N}_0$, $\mathrm{K}_{tot}^{\varphi_{simple};S_{tot}}(M)$ will achieve a minimum. Theorem A8.1 states that this minimum is a unique minimum, hence the first minimum we find by using the algorithm just described is the optimal solution.

Lemma A8.1: Let $\varphi \in \Gamma_{simple}$ and let $M \subseteq N$. On the domain $S_{tot} \in \mathbb{N}_0$, $\mathrm{K}_{tot}^{\varphi;S_{tot}}(M)$ will achieve a minimum.
Proof:
Let $\bar{S} = \lceil p / h_{\min} \rceil$. (noting that $\mathrm{K}_{tot}^{\varphi;0}(M)$=$p$). Then:
$\mathrm{K}_{tot}^{\varphi;\bar{S}}(M) = h_{\min} \cdot \bar{S} + \pi_0(\rho,\bar{S}) \cdot p \geq h_{\min} \cdot p / h_{\min} + \pi_0(\rho,\bar{S}) \cdot p = p \cdot (1+\pi_0(\rho,\bar{S})) > p$.
Therefore, the function $\mathrm{K}_{tot}^{\varphi;S_{tot}}(M)$ is increasing in at least some part of its domain. Since it is not always decreasing, it will achieve a minimum.
■

Theorem A8.1: Let $\varphi \in_{simple}$. On the domain $S_{tot} \in \mathbb{N}_0$, $\mathrm{K}_{tot}^{\varphi;S_{tot}}(M)$ will achieve one unique minimum.

Proof:

By Lemma 4.4, $\pi_0(\rho, S_{tot})$ is convex in S_{tot}.

Therefore, $\mathrm{K}_{tot}^{\varphi;S_{tot}}(M) = h_{\min} \cdot S_{tot} + \pi_0(\rho, S_{tot}) \cdot p$ is also convex in S_{tot}.

A convex function will only have at most one minimum and by Lemma A8.1, this convex function does have a minimum.

■

STEP 3:

If $|\,companiesMINh\,| > 1$, there may be multiple optimal base-stock vectors and one with an as-equal-as-possible distribution is chosen in this step. A formal definition of this was provided at the beginning of this Appendix. The idea is to iterate over all $i \in companiesMINh$ by first allocating an item to the first company in $companiesMINh$, then allocating an item to the second company, and so on. Once all companies have been given an item, start again with company 1 and repeat this process until *optimalTotalStock* has been allocated. A formal algorithmic description:

Step 3a: Set *optimalTotalStockRemaining*= *optimalTotalStock*, then set $(S_M{}^*)_i$=0 for all $i \in$M. Set j=1.

Step 3b: If j∈*companiesMINh* and *optimalTotalStockRemaining*>0 then increase $(S_M{}^*)_i$ by one and decrease *optimalTotalStockRemaining* by one.

Step 3c: If j<|M| then increase j by one and return to step 3b. If j=|M|, if *optimalTotalStockRemaining*=0 then proceed to step 4, otherwise set j=1 and return to step 3b.

STEP 4:

Return c(M)=*minCosts* and S_M*

Calculate costs

We use the recursive formula to quickly calculate the Erlang loss probability (according to Lemma 4.5). A step-by-step breakdown of the calculations is:

Step 1: Set π_0 (0)=1. Set s=1. Set $\rho = \frac{\sum_{i \in M} \lambda_i}{\mu}$

Step 2a: If $s \le \sum_{i \in M} S_i$ then set $\pi_0(s) = \frac{\pi_0(s-1) \cdot \rho}{\pi_0(s-1) \cdot \rho + 1}$. If $s > \sum_{i \in M} S_i$ proceed to step 3.

Step 2b: Increase s by one and return to step 2a.

Step 3: The result to be returned is $\sum_{i \in M} h_i \cdot S_i + \pi_0\left(\sum_{i \in M} S_i\right) \cdot \sum_{i \in M} \lambda_i \cdot c_i^{emer}$.

Set S based on RuleS

This generates a base-stock vector ***S***, and fills it with values according to *index*.

If *index*=1 then ***S*** is set to $\boldsymbol{S}^{indiv}$; if *index*=2 then ***S*** is set to $\boldsymbol{S}^{all}$; if *index*=3 then ***S*** is set to $\boldsymbol{S}^{high}$; if *index*=4 then ***S*** is set to $\boldsymbol{S}^{all}$; if *index*=5 then ***S*** is set to $\boldsymbol{S}^{mix}$. It then uses SPIS, RuleS, and ***S*** to start "For all *M*, calculate c(*M*) (Type FIX)".

For all M, calculate c(M) (Type FIX)

This function generates a cost vector $\mathbf{c}\in\mathbb{R}^{2^N\setminus\{\emptyset\}}$ (where $2^N\setminus\{\emptyset\}$ is the number of non-empty subsets of *N*) and iterates over all non-empty coalitions $M\subseteq N$. For each *M*, this function calls the function "Calculate costs", with SPIS, *M*, and ***S***, which returns the optimal costs c(*M*).

Check balancedness conditions of game (N,c). Check cost allocations. Display results.

The balancedness conditions are hard-coded for $|N|=3$ and $|N|=4$ and can be found in Appendix 2. If at least one of the conditions do not hold, then it is displayed on screen that the core is empty. Otherwise, it is displayed on screen that the core is non-empty.

For each cost allocation (they are given in Chapter 6), the costs are calculated that are allocated to each player and stored in $x\in\mathbb{R}^N$. Subsequently the program checks the stability conditions $\sum_{i\in M} x_i \le c(M)$ for all $M\subseteq N$. Whether a cost allocation is in the core or not is displayed on screen.

<u>Important note on rounding errors:</u>

It was discovered during testing of the program that these inequality checks can sometimes be affected by rounding errors. For example, an inequality was actually $25\le25$ (true), but due to rounding errors in computations it was reduced to $24.999999999999\le25$ instead (false). An investigation was made in the range of these rounding errors, and based on this, a check was added that determined whether the difference was less than 0.00000001% or not. If the difference was this small, the two values were regarded as being equal in the balancedness and stability checks.

Have all base-stock vectors been checked? and Increase index

If type is "OPT" then type is set to "FIX" and the *index* is set to 1. Then use SPIS and RuleS to start the part of the program that generates a game again.

If type is "FIX" and the *index*<5 then increase the *index* by 1. Then use SPIS and RuleS to start the part of the program that generates a game again.

If type is "FIX" and the *index*=5 then all games associated with SPIS have been checked, so go back to "Iterate over all SPIS".

Program validation & verification

The program has been verified to work correctly by:

- Building up the program part by part. After a code segment or function was done, it was fed sample input and output was checked in order to see whether it did what it was supposed to do.
- After the program was fully done, the logic of the code syntax was double-checked.

- The results shown in chapter 5 and 6 (which were obtained by using this program) have been checked in order to determine whether they make sense. So far, no remarkably strange / clearly wrong results (for example, cases where the core is empty but a cost allocation was in the core) were obtained.
- The algorithm should give the same output as the algorithm described in Appendix 11 (for cases with non-negligible transshipment costs) if we set transshipment costs to be zero in both programs. For a couple test cases (values found in Chapter 7), this turned out to be the case.
- Furthermore, after the program was fully done, a couple test cases were used in order to check whether the entire program correctly calculated the game associated with a SPIS and SRule and correctly checked the balancedness conditions and cost allocations (using a testMode part of the program that did not iterate over all thousands of instances and that gave more output that could be used to verify the numbers). The output of the program was verified by doing all required calculations in parallel by hand or via an Excel sheet and comparing the results. The program gave the correct output for all test cases. They are included here for reference:

Test case 1 - the standard identical case - Input:
Spare parts inventory situation: N={1,2,3}; $\lambda_1 = \lambda_2 = \lambda_3 = 5$, $\mu = 25$, $h_1 = h_2 = h_3 = 4000$, $c_1^{emer} = c_2^{emer} = c_3^{emer} = 13000$.
SRule: Type="OPT"; index=0.
Program output:

```
Costs of coalition {3} = 9065.57 with optimal base-stock levels: Sopt[3]=2
Costs of coalition {2} = 9065.57 with optimal base-stock levels: Sopt[2]=2
Costs of coalition {2,3} = 12930.23 with optimal base-stock levels: Sopt[2]=2  Sopt[3]=1
Costs of coalition {1} = 9065.57 with optimal base-stock levels: Sopt[1]=2
Costs of coalition {1,3} = 12930.23 with optimal base-stock levels: Sopt[1]=2  Sopt[3]=1
Costs of coalition {1,2} = 12930.23 with optimal base-stock levels: Sopt[1]=2  Sopt[2]=1
Costs of coalition {1,2,3}= 15865.64 with optimal base-stock levels: Sopt[1]=1  Sopt[2]=1  Sopt[3]=1
The core is non-empty.
All allocations of chapter 6 are implemented in this program and each of them gives a symmetric allocation
that is in the core: x(1)=5288.55   x(2)=5288.55   x(3)=5288.55
```

$\mathbf{S}_{indiv}$={2,2,2}, $\mathbf{S}_{all}$={1,1,1}, $\mathbf{S}_{high}$={3,3,3}, $\mathbf{S}_{low}$={0,0,0}, $\mathbf{S}_{mix}$={2,3,0}.

Test case 2 - the standard identical case with fixed base-stock levels - Input:
The same spare parts inventory situation as test case 1.
SRule: Type="FIX"; index=2, $\mathbf{S}_{all}$={1,1,1}.
Program output:

```
Costs of coalition {3}     = 14833.33 with optimal base-stock levels: Sopt[3]=1
Costs of coalition {2}     = 14833.33 with optimal base-stock levels: Sopt[2]=1
Costs of coalition {2,3}   = 15027.03 with optimal base-stock levels: Sopt[2]=1  Sopt[3]=1
Costs of coalition {1}     = 14833.33 with optimal base-stock levels: Sopt[1]=1
Costs of coalition {1,3}   = 15027.03 with optimal base-stock levels: Sopt[1]=1  Sopt[3]=1
Costs of coalition {1,2}   = 15027.03 with optimal base-stock levels: Sopt[1]=1  Sopt[2]=1
Costs of coalition {1,2,3} = 15865.64 with optimal base-stock levels: Sopt[1]=1  Sopt[2]=1  Sopt[3]=1
The core is non-empty.
All allocations of chapter 6 are implemented in this program and each of them gives a symmetric allocation
that is in the core: x(1)=5288.55   x(2)=5288.55   x(3)=5288.55.
```

Test case 3 - the standard identical case with inequal base-stock levels - Input:

The same spare parts inventory situation as test case 1.

SRule: Type=”FIX”; index=5, $\mathbf{S}_{\text{mix}}$={2,3,1}.

Program output:

```
Costs of coalition {3}     = 65000.0 with optimal base-stock levels: Sopt[3]=0
Costs of coalition {2}     = 12070.96 with optimal base-stock levels: Sopt[2]=3
Costs of coalition {2,3}   = 12930.23 with optimal base-stock levels: Sopt[2]=3  Sopt[3]=0
Costs of coalition {1}     = 9065.57 with optimal base-stock levels: Sopt[1]=2
Costs of coalition {1,3}   = 15027.03 with optimal base-stock levels: Sopt[1]=2  Sopt[3]=0
Costs of coalition {1,2}   = 20007.44 with optimal base-stock levels: Sopt[1]=2  Sopt[2]=3
Costs of coalition {1,2,3} = 20069.35 with optimal base-stock levels: Sopt[1]=2  Sopt[2]=3  Sopt[3]=0
The core is non-empty.
All allocations of chapter 6 are implemented in this program. Not all give the same allocation, but none of
them is in the core.
An allocation of total costs based on the demand rate of each company:
x(1)=6689.783511008859   x(2)=6689.783511008859   x(3)=6689.783511008859
An allocation of total costs based on the holding costs of each company:
x(1)=6689.783511008859   x(2)=6689.783511008859   x(3)=6689.783511008859
An allocation of total costs based on the cEMER of each company:
x(1)=6689.783511008859   x(2)=6689.783511008859   x(3)=6689.783511008859
An allocation where each company pays its own local holding and downtime/emergency costs:
x(1)=8023.116844342192   x(2)=12023.116844342192   x(3)=23.116844342192053
The Shapley value allocation rule:
x(1)=-1604.5190111887823   x(2)=-1150.2227815334531   x(3)=22824.09232574881
An allocation of total costs based on the half the lambda and half lambda*cEMER of each company:
x(1)=6689.783511008858   x(2)=6689.783511008858   x(3)=6689.783511008858
An allocation of total costs based on lambda*cEMER of each company:
x(1)=6689.783511008858   x(2)=6689.783511008858   x(3)=6689.783511008858
An allocation where holding costs are allocated based on h and downtime/emergency costs based on
lambda*cEMER:
x(1)=6689.783511008858   x(2)=6689.783511008858   x(3)=6689.783511008858
An allocation where holding costs are allocated based on h and downtime/emergency costs based on
lambda:
x(1)=6689.783511008858   x(2)=6689.783511008858   x(3)=6689.783511008858
An allocation where holding costs are allocated based on h and downtime/emergency costs based on half
lambda and half lambda*cEMER:
x(1)=6689.783511008858   x(2)=6689.783511008858   x(3)=6689.783511008858
An allocation where holding costs are allocated based on lambda and downtime/emergency costs based on
lambda*cEMER:
x(1)=6689.783511008858   x(2)=6689.783511008858   x(3)=6689.783511008858
An allocation where holding costs are allocated based on lambda and downtime/emergency costs based on
half lambda and half lambda*cEMER:
x(1)=6689.783511008858   x(2)=6689.783511008858   x(3)=6689.783511008858
An allocation of total BENEFITS based on the demand rate of each company:
x(1)=-12956.820874893256   x(2)=-9951.433946695102   x(3)=42977.60535461494
An allocation of total BENEFITS equally amongst companies:
x(1)=-12956.82087489326   x(2)=-9951.433946695106   x(3)=42977.605354614934
An allocation of total BENEFITS according to the Shapley value:
x(1)=-1604.5190111887805   x(2)=-1150.2227815334518   x(3)=22824.092325748818
An allocation based on relative distance from individually optimal S:
x(1)=0.0   x(2)=6689.783511008859   x(3)=13379.567022017718
An allocation based on relative distance from global optimal S:
x(1)=5017.337633256644   x(2)=10034.675266513288   x(3)=5017.337633256644
```

Test case 4 - inequal holding cost rates - Input:

Spare parts inventory situation: N={1,2,3}; $\lambda_1= \lambda_2=\lambda_3=5$, $\mu=25$, $h_1=400$, $h_2=4000$, $h_3=28000$, $c_1^{emer} = c_2^{emer} = c_3^{emer} = 13000$.

SRule: Type="OPT"; index=0.

Program output:

```
Costs of coalition {3}     = 38833.33 with optimal base-stock levels: Sopt[3]=1
Costs of coalition {2}     = 9065.57 with optimal base-stock levels: Sopt[2]=2
Costs of coalition {2,3}   = 12930.23 with optimal base-stock levels: Sopt[2]=3  Sopt[3]=0
Costs of coalition {1}     = 1270.96 with optimal base-stock levels: Sopt[1]=3
Costs of coalition {1,3}   = 1692.96 with optimal base-stock levels: Sopt[1]=4  Sopt[3]=0
Costs of coalition {1,2}   = 1692.96 with optimal base-stock levels: Sopt[1]=4  Sopt[2]=0
Costs of coalition {1,2,3} = 2069.35 with optimal base-stock levels: Sopt[1]=5  Sopt[2]=0  Sopt[3]=0
The core is non-empty.
```

All cost allocations were calculated correctly (they are not shown here in order to bring this appendix down to a manageable size).

$\mathbf{S}_{indiv}$={3,2,1}, $\mathbf{S}_{all}$={5,0,0}, $\mathbf{S}_{high}$={3,3,3}, $\mathbf{S}_{low}$={0,0,0}, $\mathbf{S}_{mix}$={3,3,0}.

Test case 5 - inequal demand rates - Input:

Spare parts inventory situation: N={1,2,3}; $\lambda_1=0.5$, $\lambda_2=5$, $\lambda_3=50$, $\mu=25$, $h_1=h_2=h_3=4000$, $c_1^{emer} = c_2^{emer} = c_3^{emer} = 13000$.

SRule: Type="FIX"; index=1, **S**indiv={1,2,7}.

Program output:

```
Costs of coalition {3}     = 30236.56 with optimal base-stock levels: Sopt[3]=7
Costs of coalition {2}     = 9065.57 with optimal base-stock levels: Sopt[2]=2
Costs of coalition {2,3}   = 36263.60 with optimal base-stock levels: Sopt[2]=2  Sopt[3]=7
Costs of coalition {1}     = 4127.45 with optimal base-stock levels: Sopt[1]=1
Costs of coalition {1,3}   = 32598.91 with optimal base-stock levels: Sopt[1]=1  Sopt[3]=7
Costs of coalition {1,2}   = 12101.84 with optimal base-stock levels: Sopt[1]=1  Sopt[2]=2
Costs of coalition {1,2,3} = 40062.79 with optimal base-stock levels: Sopt[1]=1  Sopt[2]=2  Sopt[3]=7
The core is non-empty.
```

All cost allocations were calculated correctly (they are not shown here in order to bring this appendix down to a manageable size).

Test case 6 - inequal emergency costs - Input:

Spare parts inventory situation: N={1,2,3}; $\lambda_1=\lambda_2=\lambda_3=5$, $\mu=25$, $h_1=h_2=h_3=4000$, $c_1^{emer} = 2600, c_2^{emer} = 13000, c_3^{emer} = 78000$.

SRule: Type="FIX"; index=1, $\mathbf{S}_{high}$={3,3,3}.

Program output:

```
Costs of coalition {3}     = 12425.76 with optimal base-stock levels: Sopt[3]=3
Costs of coalition {2}     = 12070.96 with optimal base-stock levels: Sopt[2]=3
Costs of coalition {2,3}   = 24001.74 with optimal base-stock levels: Sopt[2]=3  Sopt[3]=3
Costs of coalition {1}     = 12014.19 with optimal base-stock levels: Sopt[1]=3
Costs of coalition {1,3}   = 24001.54 with optimal base-stock levels: Sopt[1]=3  Sopt[3]=3
Costs of coalition {1,2}   = 24000.30 with optimal base-stock levels: Sopt[1]=3  Sopt[2]=3
Costs of coalition {1,2,3} = 36000.01 with optimal base-stock levels: Sopt[1]=3  Sopt[2]=3  Sopt[3]=3
The core is non-empty.
```

All cost allocations were calculated correctly.

Test case 7 - four companies - Input:

Spare parts inventory situation: N={1,2,3,4}; $\lambda_1=\lambda_2=\lambda_3=\lambda_4=5$, $\mu=25$, $h_1=h_2=h_3=h_4=4000$, $c_1^{emer}=2600, c_2^{emer}=13000, c_3^{emer}=78000, c_4^{emer}=13000$.

SRule: Type=”OPT”; index=0.

Program output:

```
Costs of coalition {4}       = 9065.57 with optimal base-stock levels: Sopt[4]=2
Costs of coalition {3}       = 12425.76 with optimal base-stock levels: Sopt[3]=3
Costs of coalition {3,4}     = 15255.81 with optimal base-stock levels: Sopt[3]=2  Sopt[4]=1
Costs of coalition {2}       = 9065.57 with optimal base-stock levels: Sopt[2]=2
Costs of coalition {2,4}     = 12930.23 with optimal base-stock levels: Sopt[2]=2  Sopt[4]=1
Costs of coalition {2,3}     = 15255.81 with optimal base-stock levels: Sopt[2]=2  Sopt[3]=1
Costs of coalition {2,3,4}   = 17541.67 with optimal base-stock levels: Sopt[2]=2  Sopt[3]=1  Sopt[4]=1
Costs of coalition {1}       = 6166.67 with optimal base-stock levels: Sopt[1]=1
Costs of coalition {1,4}     = 12216.22 with optimal base-stock levels: Sopt[1]=1  Sopt[4]=1
Costs of coalition {1,3}     = 14883.72 with optimal base-stock levels: Sopt[1]=2  Sopt[3]=1
Costs of coalition {1,3,4}   = 17387.50 with optimal base-stock levels: Sopt[1]=2  Sopt[3]=1  Sopt[4]=1
Costs of coalition {1,2}     = 12216.22 with optimal base-stock levels: Sopt[1]=1  Sopt[2]=1
Costs of coalition {1,2,4}   = 14834.80 with optimal base-stock levels: Sopt[1]=1  Sopt[2]=1  Sopt[4]=1
Costs of coalition {1,2,3}   = 17387.50 with optimal base-stock levels: Sopt[1]=2  Sopt[2]=1  Sopt[3]=1
Costs of coalition {1,2,3,4} = 20093.11 with optimal base-stock levels: Sopt[1]=1  Sopt[2]=1  Sopt[3]=1
Sopt[4]=1
The core is non-empty.
All cost allocations were calculated correctly.
```

$\mathbf{S}_{indiv}$={1,2,3,2}, $\mathbf{S}_{all}$={1,1,1,1}, $\mathbf{S}_{high}$={3,3,3,3}, $\mathbf{S}_{low}$={0,0,0,0}, $\mathbf{S}_{mix}$={1,3,0,2}.

Appendix 9: Example games for section 5.4

In the Appendix, we provide example games that illustrate concepts discussed in Section 5.4.4 (as an illustration why low repair rates and/or low holding costs relatively often lead to games with empty cores) and in Section 5.4.5 (2-player subgames).

An illustration why low repair rates and/or low holding costs relatively often lead to games with empty cores

We present example games 9.1, 9.2, and 9.3 below. Example 9.1 will be a game with an empty core associated with a spare parts inventory situations for which companies have identical, but very low repair rates and identical, but very high holding cost rates. For example 9.2, we change the spare part pooling game by setting the repair rate to be very high instead and thanks to this change, the core of the associated game is non-empty. And for example 9.3, we change the original spare part pooling game by setting the holding cost rate to be very low instead and thanks to this change, the core of the associated game is non-empty. We keep track of how big of a factor the holding costs play in the cost function. We can observe that in examples 9.1, emergency costs are dominant (and since they differ quite a bit between companies, this leads to an empty core), while in examples 9.2 and 9.3 the holding costs are dominant (and therefore the annoying effect of different c^{emer} between companies is lessened).

Example 9.1
Consider the 3-player simple spare parts inventory situation $\varphi_7 \in \Gamma_{simple,id:\lambda,h}$ with N={1,2,3}, λ has value All-Standard (hence λ=5), μ has value All-Min (hence μ=1.67), h has value All-Max (hence h=28000), and c^{emer} has value DIFF3 (hence c_1^{emer}=6500, c_2^{emer}=78000, c_3^{emer}=6500). Suppose that we have situation OPT. The associated spare parts pooling game is described by c(M), given in table 9.1.

Table 9.1: Cost function (column 2), optimal total base-stock level sum (column 3), and the percentage of the cost function that is defined by the holding cost (column 4) for example 9.1.

Coalition M	c(M)	S_M^{tot} *	Holding cost percentage, i.e. $\left(h \cdot S_M^{tot} *\right) / c(M) * 100\%$
{1}	32,500.0	0	0%
{2}	182,721.67	5	77%
{3}	32,500.0	0	0%
{1,2}	273,856.26	7	72%
{1,3}	65,000.0	0	0%
{2,3}	273,856.26	7	72%
{1,2,3}	353,645.09	9	71%

The core (and the imputation set) is empty, since c({1,2,3}) = 353645.09 > c({1})+c({2})+c({3}) = 247721.67.

Example 9.2

Consider the 3-player simple spare parts inventory situation $\varphi_8 \in \Gamma_{simple,id:\lambda,h}$, which is the same as φ_7 (used in the previous example), with one exception: μ has value All-Max (hence μ=500) instead. Suppose that we have situation OPT. The associated spare parts pooling game is described by c(*M*), given in table 9.2.

Table 9.2: Cost function (column 2), optimal total base-stock level sum (column 3), and the percentage of the cost function that is defined by the holding cost (column 4) for example 9.2.

Coalition *M*	c(*M*)	$S_M^{tot}*$	Holding cost percentage, i.e. $\left(h \cdot S_M^{tot}*\right)/c(M)*100\%$
{1}	28,321.78	1	99%
{2}	31,861.39	1	88%
{3}	28,321.78	1	99%
{1,2}	36,284.31	1	77%
{1,3}	29,274.51	1	96%
{2,3}	36,284.31	1	77%
{1,2,3}	41,252.43	1	68%

The core is non-empty. For example, x_1=10,000, x_2=21,252.43, x_3=10,000 is a core element.

Example 9.3

Consider the 3-player simple spare parts inventory situation $\varphi_9 \in \Gamma_{simple,id:\lambda,h}$, which is the same as φ_7 (used in example 9.1), with one exception: *h* has value All-Min (hence *h*=400) instead. Suppose that we have situation OPT. The associated spare parts pooling game is described by c(*M*), given in table 9.3.

Table 9.3: Cost function (column 2), optimal total base-stock level sum (column 3), and the percentage of the cost function that is defined by the holding cost (column 4) for example 9.3.

Coalition *M*	c(*M*)	$S_M^{tot}*$	Holding cost percentage, i.e. $\left(h \cdot S_M^{tot}*\right)/c(M)*100\%$
{1}	3461.66	8	92%
{2}	4311.66	10	93%
{3}	3461.66	8	92%
{1,2}	6370.02	15	94%
{1,3}	5529.8	12	87%
{2,3}	6370.02	15	94%
{1,2,3}	8211.84	19	93%

The core is non-empty. For example, x_1=2,500, x_2=3211.84, x_3=2,500 is a core element.

Two-player sub-games

In example 9.4, the core of the 3-player game is empty, but this is not due to a problem that already exists in a 2-player sub-game, i.e. none of the sub-games had empty cores (this is unlikely to happen, according to Table 5.6). In example 9.5, the core of the 3-player game is non-empty, but this is despite problems with 2-player sub-games, i.e. two sub-games do have empty cores (this is unlikely to happen, according to Table 5.7). Lastly, in example 9.6, the core of the 3-player game is empty and two sub-games also have empty cores (this is likely to happen, according to Table 5.6).

Example 9.4

Consider the 3-player simple spare parts inventory situation $\varphi_4 \in \Gamma_{simple,id:\lambda,h}$ with N={1,2,3}, μ=1.67, h=8000; c_1^{emer}=26000, c_2^{emer}=6500; c_3^{emer}=13000; λ=0.5.

Suppose that we have situation OPT, so base-stock levels are to-be-optimized. The associated spare parts pooling game is described by (values are rounded to two decimals):

c({1})	= 10,995.39 (with $S_M^{tot}*$=1);	c({2})	= 3,250.0 (with $S_M^{tot}*$=0)
c({3})	= 6,500.0 (with $S_M^{tot}*$=0);	c({1,2})	= 14,086.14 (with $S_M^{tot}*$=1)
c({1,3})	= 15,303.37 (with $S_M^{tot}*$=1);	c({2,3})	= 9,750.0 (with $S_M^{tot}*$=0)
c({1,2,3})	= 18,764.98 (with $S_M^{tot}*$=1).		

Note that the core of this game is empty, since: c({1,2,3}) = 18,764.98 > c({2})+c({1,3}) = 18553.37. Note that the imputation set is non-empty, since; c({1,2,3}) = 18764.98 < c({1})+c({2})+c({3}) = 20745.39. If we limit ourselves to a subgame with player set {1,2}, since c({1,2}) = 14,086.14 < c({1})+c({2}) = 14,245.39, the core is non-empty. Similarly, the core of the subgame with player set {1,3} and the core of the subgame with player set {2,3} is non-empty.

Example 9.5

Consider the 3-player simple spare parts inventory situation $\varphi_5 \in \Gamma_{simple,id:h}$ with N={1,2,3}, μ=25, h=8000; c_1^{emer}=2600, c_2^{emer}=13000; c_3^{emer}=78000; λ_1=2.5, λ_2=50, λ_3=2.5.

Suppose that we have situation FIX with base-stock vector $\boldsymbol{S}$ given by S_1=0, S_2=4, S_3=1. The associated spare parts pooling game is described by (values are rounded):

c({1}) = 6,500.0;	c({2})= 93,904.76;	c({3}) = 25,727.27;
c({1,2}) = 101,460.49;	c({1,3}) = 41,583.33;	c({2,3}) = 75,952.29;
c({1,2,3}) = 81,555.08		

The core is non-empty. For example, x_1=6,055.08, x_2=55,500, x_3=20,000 is a core element. If we limit ourselves to a subgame with player set {1,2}, since c({1,2}) = 101,460.49 > c({1})+c({2}) = 100,404.76, the core is empty. Similarly, the core of the subgame with player set {1,3} is empty. The core of the subgame with player set {2,3}, however, is non-empty.

Example 9.6

Consider the 3-player simple spare parts inventory situation $\varphi_6 \in \Gamma_{simple,id:\lambda,h}$ with N={1,2,3}, μ=1.67, h=4000; c_1^{emer}=2600, c_2^{emer}=13000; c_3^{emer}=78000; λ=5. Suppose that we have situation OPT. The associated spare parts pooling game is described by (values are rounded to two decimals):

c({1})	= 13,000.0 (with $S_M^{tot}*$=0);	c({2})	= 27,120.28 (with $S_M^{tot}*$=5)
c({3})	= 35,139.95 (with $S_M^{tot}*$=8);	c({1,2})	= 41,454.51 (with $S_M^{tot}*$=8)
c({1,3})	= 52,524.77 (with $S_M^{tot}*$=12);	c({2,3})	= 53,108.61 (with $S_M^{tot}*$=12)
c({1,2,3})	= 69,099.63 (with $S_M^{tot}*$=16).		

The core is empty, since: c({1,2,3}) = 69,099.63 > c({1})+c({2,3}) = 66,108.61. Note that the imputation set is non-empty as c({1,2,3}) = 69,099.63 < c({1})+c({2})+c({3}) = 75,260.23. If we limit ourselves to a subgame with player set {1,2}, since c({1,2}) = 41454.51 > c({1})+c({2}) = 40120.28, the core is empty. Similarly, the core of the subgame with player set {1,3} is empty. The core of the subgame with player set {2,3}, however, is non-empty.

Appendix 10: All cost allocations

We now provide a list of cost allocation rules that intuitively may seem reasonable. The ones that are defined already in Chapter 6 are not included here. For each cost allocation rule, we show the percentage of games for which it gave a core element in the numerical experiment of Chapter 6 over 46080 games. For all these formulas, let $\varphi \in \Gamma_{simple}$, let (N,c) be the associated game and let $i \in N$. We will only provide the formula for a game with to-be-optimized base-stock levels and formulas for games with fixed base-stock levels should follow naturally from the description given, unless noted otherwise.

Allocation AH (51%): An allocation of total costs based on the holding costs rate of each company: $AH_i^{OPT}(\varphi) = c(N) \cdot \dfrac{h_i}{\sum_{j \in N} h_j}$

Allocation AC (45%): An allocation of total costs based on the emergency (shipment and downtime) costs of each company: $AC_i^{OPT}(\varphi) = c(N) \cdot \dfrac{c_i^{emer}}{\sum_{j \in N} c_j^{emer}}$

Allocation A-local (55%): An allocation where each company pays its own local holding and emergency costs: A-local$_i^{OPT}(\varphi) = h_i \cdot \left(S_N^*\right)_i + \pi_0\left(\sum_{j \in N} \lambda_j / \mu, \sum_{j \in N} \left(S_N^*\right)_j\right) \cdot \lambda_i \cdot c_i^{emer}$.

Allocation HALF (57%): An allocation of total costs based on the half the demand rate and half the demand rate times emergency costs of each company:

$$HALF_i^{OPT}(\varphi) = c(N) \cdot \left(\frac{\lambda_i}{2\sum_{j \in N} \lambda_j} + \frac{\lambda_i \cdot c_i^{emer}}{2\sum_{j \in N} \lambda_j \cdot c_j^{emer}} \right)$$

Allocation ALC (57%): An allocation of total costs based on the demand rate times emergency costs of each company: $ALC_i^{OPT}(\varphi) = c(N) \cdot \dfrac{\lambda_i \cdot c_i^{emer}}{\sum_{j \in N} \lambda_j \cdot c_j^{emer}}$

Allocation SPLITH1 (63%): An allocation where holding costs are allocated based on holding cost rates and emergency costs based on the demand rate times emergency costs:

$$SPLITH1_i^{OPT}(\varphi) = \frac{h_i}{\sum_{j \in N} h_j} \cdot \left(\sum_{j \in N} h_j \cdot \left(S_N^*\right)_j \right) + \pi_0\left(\sum_{j \in N} \lambda_j / \mu, \sum_{j \in N} \left(S_N^*\right)_j\right) \cdot \frac{\lambda_i \cdot c_i^{emer}}{\sum_{j \in N} \lambda_j \cdot c_j^{emer}}$$

Allocation SPLITH2 (57%): An allocation where holding costs are allocated based on holding cost rates and emergency costs based on the demand rate:

$$SPLITH2_i^{OPT}(\varphi) = \frac{h_i}{\sum_{j \in N} h_j} \cdot \left(\sum_{j \in N} h_j \cdot \left(S_N^*\right)_j \right) + \pi_0\left(\sum_{j \in N} \lambda_j / \mu, \sum_{j \in N} \left(S_N^*\right)_j\right) \cdot \frac{\lambda_i}{\sum_{j \in N} \lambda_j}$$

Allocation SPLITH3 (57%): An allocation where holding costs are allocated based on holding cost rates and emergency costs based on half the demand rate and half the demand rate times emergency costs:

$$SPLITH3_i^{OPT}(\varphi)=\frac{h_i}{\sum_{j\in N} h_j}\cdot\left(\sum_{j\in N} h_j\cdot\left(S_N^*\right)_j\right)+\pi_0\left(\sum_{j\in N}\lambda_j/\mu,\sum_{j\in N}\left(S_N^*\right)_j\right)\cdot\left(\frac{\lambda_i}{2\sum_{j\in N}\lambda_j}+\frac{\lambda_i\cdot c_i^{emer}}{2\sum_{j\in N}\lambda_j\cdot c_j^{emer}}\right)$$

Allocation SPLITL (60%): An allocation where holding costs are allocated based on demand rates and emergency costs based on half the demand rate and half the demand rate times emergency costs:

$$SPLITL_i^{OPT}(\varphi)=\frac{\lambda_i}{\sum_{j\in N}\lambda_j}\cdot\left(\sum_{j\in N} h_j\cdot\left(S_N^*\right)_j\right)+\pi_0\left(\sum_{j\in N}\lambda_j/\mu,\sum_{j\in N}\left(S_N^*\right)_j\right)\cdot\left(\frac{\lambda_i}{2\sum_{j\in N}\lambda_j}+\frac{\lambda_i\cdot c_i^{emer}}{2\sum_{j\in N}\lambda_j\cdot c_j^{emer}}\right)$$

Allocation MARGINAL (52%): An allocation of total costs based on the square root of the demand rate of each company: $MARGINAL_i^{OPT}(\varphi)=c(N)\cdot\frac{\sqrt{\lambda_i}}{\sum_{j\in N}\sqrt{\lambda_j}}$

Allocation BE (63%): An allocation of total benefits equally amongst companies:

$$\mathrm{BE}_i^{OPT}(\varphi)=c(\{i\})-\frac{1}{|N|}\cdot\left(\sum_{j\in N}c(\{j\})-c(N)\right)$$

Allocation DISTANCE1 (37%) An allocation based on relative distance of fixed base-stock vector ***S*** from individually optimal base-stock vector $\boldsymbol{S}^{indiv}$ (only for situation FIX):

$$\mathrm{DISTANCE1}_i^{FIX}(\varphi,\boldsymbol{S})=\begin{cases}\frac{|S_i-S_i^{indiv}|}{\sum_{j\in N}|S_j-S_j^{indiv}|}\cdot c(N) & \text{if } \sum_{j\in N}|S_j-S_j^{indiv}|>0\\ c(N)/|N| & \textit{otherwise}\end{cases}.$$

Allocation DISTANCE2 (26%) An allocation based on relative distance of fixed base-stock vector ***S*** from globally optimal base-stock vector $\boldsymbol{S}_N^*$ (only for situation FIX):

$$\mathrm{DISTANCE2}_i^{FIX}(\varphi,\boldsymbol{S})=\begin{cases}\frac{|S_i-\left(S_N^*\right)_i|}{\sum_{j\in N}|S_j-\left(S_N^*\right)_j|}\cdot c(N) & \text{if } \sum_{j\in N}|S_j-\left(S_N^*\right)_j|>0\\ c(N)/|N| & \textit{otherwise}\end{cases}.$$

Appendix 11: Proof to Lemma 6.3

Lemma 6.3: Allocation rules AL$^{\text{FIX}}$, SPLIT$^{\text{FIX}}$, BL$^{\text{FIX}}$, Φ^{FIX}, AL$^{\text{OPT}}$, SPLIT$^{\text{OPT}}$, BL$^{\text{OPT}}$, and Φ^{OPT} are symmetric.
Proof:
Let $\varphi \in \Gamma_{simple}$ such that $\lambda_i=\lambda_j$, $h_i=h_j$, and $c_i^{emer} = c_j^{emer}$ for some $i,j \in N$. Let $\boldsymbol{S} \in \mathbb{N}_0^{\ N}$ and let $\boldsymbol{S}^{\boldsymbol{identical}} \in \mathbb{N}_0^{\ N}$ such that $S_i=S_j$. Let the game associated with φ be (N,c^{OPT}) and the game associated with φ and $\boldsymbol{S}$ be (N,c^{FIX}).

Step 1 (AL$^{\text{FIX}}$ and AL$^{\text{OPT}}$):

$$AL_i^{FIX}(\varphi,\boldsymbol{S}) = K^{\varphi,S}(N)\cdot\frac{\lambda_i}{\sum_{k\in N}\lambda_k} = K^{\varphi,S}(N)\cdot\frac{\lambda_j}{\sum_{k\in N}\lambda_k} = AL_j^{FIX}(\varphi,\boldsymbol{S}).$$

By the above, we have $AL_i^{FIX}(\varphi,\boldsymbol{S}^{\boldsymbol{identical}}) = AL_j^{FIX}(\varphi,\boldsymbol{S}^{\boldsymbol{identical}})$ and hence AL$^{\text{FIX}}$ is symmetric. By the above, we also have $AL_i^{OPT}(\varphi) = AL_i^{FIX}(\varphi, \boldsymbol{S}^*_N(\varphi)) =$ $AL_j^{FIX}(\varphi,\boldsymbol{S}^*_N(\varphi)) = AL_j^{OPT}(\varphi)$ and hence AL$^{\text{OPT}}$ is symmetric.

Step 2 (SPLIT$^{\text{FIX}}$ and SPLIT$^{\text{OPT}}$):

$$SPLIT_i^{FIX}(\varphi,\boldsymbol{S}) = \frac{\lambda_i}{\sum_{k\in N}\lambda_k}\cdot\left(\sum_{k\in N}h_k\cdot S_k\right)+\pi_0\left(\frac{\sum_{k\in N}\lambda_k}{\mu},\sum_{k\in N}S_k\right)\cdot\lambda_i\cdot c_i^{emer} =$$

$$SPLIT_j^{FIX}(\varphi,\boldsymbol{S}) = \frac{\lambda_j}{\sum_{k\in N}\lambda_k}\cdot\left(\sum_{k\in N}h_k\cdot S_k\right)+\pi_0\left(\frac{\sum_{k\in N}\lambda_k}{\mu},\sum_{k\in N}S_k\right)\cdot\lambda_j\cdot c_j^{emer}$$

By the above, we have $SPLIT_i^{FIX}(\varphi,\boldsymbol{S}^{\boldsymbol{identical}}) = SPLIT_j^{FIX}(\varphi,\boldsymbol{S}^{\boldsymbol{identical}})$ and hence SPLIT$^{\text{FIX}}$ is symmetric. By the above, we also have $SPLIT_i^{OPT}(\varphi) = SPLIT_i^{FIX}(\varphi,\boldsymbol{S}^*_N(\varphi))$ $= SPLIT_j^{OPT}(\varphi) = SPLIT_j^{FIX}(\varphi,\boldsymbol{S}^*_N(\varphi))$ and hence SPLIT$^{\text{OPT}}$ is symmetric.

Step 3 (BL$^{\text{FIX}}$ and BL$^{\text{OPT}}$):

$$BL_i^{FIX}(\varphi,\boldsymbol{S}^{\boldsymbol{identical}}) = \mathrm{K}^{\varphi;S^{identical}}(\{i\}) - \frac{\lambda_i}{\sum_{k\in N}\lambda_k}\cdot\left(\sum_{k\in N}\mathrm{K}^{\varphi;S^{identical}}(\{k\}) - \mathrm{K}^{\varphi;S^{identical}}(N)\right) =$$

$$BL_j^{FIX}(\varphi,\boldsymbol{S}^{\boldsymbol{identical}}) = \mathrm{K}^{\varphi;S^{identical}}(\{j\}) - \frac{\lambda_j}{\sum_{k\in N}\lambda_k}\cdot\left(\sum_{k\in N}\mathrm{K}^{\varphi;S^{identical}}(\{k\}) - \mathrm{K}^{\varphi;S^{identical}}(N)\right)$$

Hence BL$^{\text{FIX}}$ is symmetric. To show BL$^{\text{OPT}}$ is symmetric, we use $S^*_{\{i\}}(\varphi) = S^*_{\{j\}}(\varphi)$:

$$BL_i^{OPT}(\varphi) = \mathrm{K}^{\varphi;S^*_{\{i\}}(\varphi)}(\{i\}) - \frac{\lambda_i}{\sum_{k\in N}\lambda_k} \cdot \left(\sum_{k\in N} \mathrm{K}^{\varphi;S^*_{\{k\}}(\varphi)}(\{k\}) - \mathrm{K}^{\varphi;S^*_N(\varphi)}(N) \right) =$$

$$BL_j^{OPT}(\varphi) = \mathrm{K}^{\varphi;S^*_{\{j\}}(\varphi)}(\{j\}) - \frac{\lambda_j}{\sum_{k\in N}\lambda_k} \cdot \left(\sum_{k\in N} \mathrm{K}^{\varphi;S^*_{\{k\}}(\varphi)}(\{k\}) - \mathrm{K}^{\varphi;S^*_N(\varphi)}(N) \right) =$$

Step 4 (Φ^{FIX} and Φ^{OPT}):

In similar fashion as Lemma 6.1, we use for $i\in N$: $\Phi_i^{game}(N, c^{OPT}) = \Phi_i^{OPT}(\varphi)$ and $\Phi_i^{game}(N, c^{FIX}) = \Phi_i^{FIX}(\varphi, S)$. Since Φ_i^{game} satisfies symmetry in terms of a game (see Appendix 2), and since $K^{\varphi,S}(M \cup \{i\}) = K^{\varphi,S}(M \cup \{j\})$ for all $M \subseteq N \setminus \{i, j\}$, it follows that allocation rules Φ^{FIX} and Φ^{OPT} are symmetric.

■

Appendix 12: Algorithms needed to calculate cost functions of a general spare parts pooling game

Calculation of the cost functions for general spare parts pooling game (which are covered in Chapter 7) involves creating a Markov chain and calculating steady-state probabilities. Furthermore, calculating optimal costs when base-stock levels are to-be-optimized involves finding an optimal solution in an infinite space (which can be bounded and enumerated efficiently). The algorithms that do this are described in this Appendix and are implemented in a Java application. Particularly Algorithm A1, A2, and A3 are important, while the other algorithms describe program implementations.

Algorithm A8: Calculating the minimum costs of a coalition M when S is to-be-optimized.

Necessary input: $M, (\lambda_i)_{i\in M}, (\mu_i)_{i\in M}, (h_i)_{i\in M}, (c_i^{emer})_{i\in M}, (c_{ij}^{trans})_{i\in M, j\in M}$.

Step 1 (sorting): Sort all the companies in decreasing order of h_i, such that $h_1 \geq h_2 \geq h_{|M|}$ (this will increase computation speed due to the way Algorithm A2 operates).
Step 2a (initializing): Create $\boldsymbol{S} \in \mathbb{N}_0^M$ such that $S_i=0$ for all i in M.
Step 2b (initializing): Set *costWhenNothingOnStock*= $\sum_{i\in M} c_i^{emer} \cdot \lambda_i$.
Step 2c (initializing): Set *minCosts*=*costWhenNothingOnStock* and set *Sopt*=***S***.
Step 3a (find next feasible base-stock vector ***S***): Use algorithm A2 to set ***S*** to the next vector. If this algorithm returns "no feasible base-stock vectors that have not been already checked remain" then proceed to step 4.
Step 3b (calculate costs of ***S***): Use algorithm A3 to set *currentCosts*.
Step 3c (check whether this is an improvement): If *currentCosts*<*minCosts* then set *minCosts*=*currentCosts* and set *Sopt*=***S***. Either way, subsequently return to step 3a.
Step 4 (end): The optimal base-stock vector is *Sopt* and the associated minimum costs are *minCosts*.

Algorithm A2: Find the next feasible base-stock vector S.

Necessary input: ***S***,*minCosts*, $M, (h_i)_{i\in M}$.

Explanation: The cost function consists of a part holding costs that are increasing in ***S*** and of a part emergency/lateral costs that are positive. Consider a base-stock vector ***S*** for which the holding costs are already higher than the current *minCosts* of a coalition *M*. Then we know that ***S*** can never yield the optimal solution (in this case it is also not necessary to go through the computational effort of constructing the Markov chain and calculating emergency/lateral shipment and downtime costs). Furthermore, for this ***S***, we will know that any $\boldsymbol{S}^{worse}$, for which $S_i^{worse} \geq S_i$ for all i in M, can never yield the optimal solution either and is hence not feasible. We will use an approach that provides subsequent feasible base-stock vectors in lexicographic order. How this works may best be illustrated by an example. Let M={1,2,3} and h={600,300,200}. Algorithm A1 starts with ***S***={0,0,0} and *minCosts*=1000 (i.e. emergency shipment costs when nothing is on stock). Assume for this example that ***S***={0,0,0} and *minCosts*=1000 is the optimal solution (usually, this is not the case and *minCosts* will decrease as we iterate).

Now, algorithm A2 returns subsequently {1,0,0}, {0,1,0}, {1,1,0}, {0,2,0}, (0,3,0}, {0,0,1}, {1,0,1}, {0,1,1}, {0,2,1}, {0,0,2}, {0,1,2}, {0,0,3}, {0,0,4}, and "no feasible base-stock vectors that have not been already checked remain".

Step 1 (initializing): Set i=1.
Step 2a (increase base-stock level): Set $S_i=S_i+1$.
Step 2b (calculate holding costs): Set *holdingCosts*= $\sum_{i\in M} h_i \cdot S_i$.
Step 3 (check whether this is a feasible base-stock vector): If *holdingCosts*<*minCosts* then proceed to Step 5. Else, set S_j=0 for all $j \in \{1,2,...,i\}$ and continue to Step 4.
Step 4 (continue the iteration): Set $i=i+1$. If $i \leq M$ return to Step 2a, else proceed to Step 5.
Step 5 (end): If S_i=0 for all i in M then return "no feasible base-stock vectors that have not been already checked remain". Else, return $\boldsymbol{S}$.

Algorithm A3: Calculating the costs of a coalition M for a given S.
Necessary input: $\boldsymbol{S}, M, (\lambda_i)_{i\in M}, (\mu_i)_{i\in M}, (h_i)_{i\in M}, (c_i^{emer})_{i\in M}, (c_{ij}^{trans})_{i\in M, j\in M}$.
Explanation: Note that a Markov state is defined by $\boldsymbol{x}$, where x_i is the on-hand inventory at company i. The variable *index* pinpoints a Markov state by an integer number; this is explained in more detail in algorithm A6. *index* ranges from 1 to *maxIndex*. Transshipments to company i are selected as follows. Source from the company j with the lowest transshipments costs to company i, c_{ji}^{trans} . Break ties by sourcing from the company with the highest current on-hand inventory, x_j. Break remaining ties by sourcing from the company with the lowest index, j.
Step 1 (initializing): Set *maxIndex*= $\prod_{i\in M}(S_i+1)$.
Step 2 (calculating the steady-state probabilities): Use Algorithm A4 to populate the Markov chain. Then use algorithm A5 to calculate the steady-state probabilities $\boldsymbol{\pi}$.
Step 3 (holding costs): Set *costs*= $\sum_{i\in M} h_i \cdot S_i$.
Step 4 (emergency shipment costs): Set *costs*=*costs* $+\sum_{i\in M}\pi_1 \cdot \lambda_i \cdot c_i^{emer}$.[23]
Step 5a (transshipments; initializingA): Set *index*=2.
Step 5b (transshipments; find state): Use algorithm A6 to obtain the state $\boldsymbol{x}$ corresponding to *index*.
Step 5c (transshipments; initializingB): Set i=1.
Step 5d (transshipments; select companies that need transshipments in this state): If x_i=0, perform steps 5e and 5f. Else, proceed to step 5g.

[23] Notation clarification: π_1 is the steady-state probability of being in a state where x_i=0 for all i in M. We have used π_0 for this in Chapter 4 and π_{0M} in Chapter 7. In these algorithms we will pinpoint a Markov state with an integer number and due to the method chosen for this, the state where x_i=0 for all i in M corresponds to the integer number 1.

Step 5e (transshipments; find company *j* that will source transshipment): Set $JJ = \{ j \mid j \in (M \setminus i) \wedge c_{ji}^{trans} = \min_{j \in (M \setminus i)} (c_{ji}^{trans}) \}$. Set $J = \{ j \mid j \in JJ \wedge x_j = \max_{j \in JJ}(x_j) \}$. Set $j = \min(J)$.

Step 5f (transshipments; add costs): Set *costs*=*costs*+ $\pi_{index} \cdot \lambda_i \cdot c_{ji}^{trans}$.

Step 5g (transshipments; iterationB): Set *i*=*i*+1. If *i*≤*M* return to Step 5d, else proceed to step 5h.

Step 5h (transshipments; iterationA): Set *index*=*index*+1. If *index*≤*MaxIndex* return to Step 5b, else proceed to step 6.

Step 6 (end): Return *costs*.

Algorithm A4: Populating the Markov chain.

Necessary input: $S, M, (\lambda_i)_{i \in M}, (\mu_i)_{i \in M}, (c_{ij}^{trans})_{i \in M, j \in M}$.

Explanation: This algorithm sets the steady-state state occupancy equations of the Markov chain in the variable *Matrix*. They will be used to calculate the steady-state probabilities ***π*** in algorithm A5. The variable *index* pinpoints a Markov state by an integer number; this is explained in more detail in algorithm A6. *index* ranges from 1 to *maxIndex*. The transition rate from state *X* to state *Y* is $Matrix_{Y,X}$. The sum of all transition rates out of state *X* is $-Matrix_{X,X}$. $Matrix_{X,MaxIndex+1}$=0 for all $X \in \{1,2,...,MaxIndex-1\}$. $Matrix_{MaxIndex,X}$=1 for all $X \in \{1,2,...,MaxIndex+1\}$ (this is the property that the sum of all steady-state occupancies is 1 and "overwrites" the superfluous equations for state *MaxIndex*). Hence, a "row" $Z \in \{1,2,...,MaxIndex\}$ of *Matrix* corresponds to the steady-state occupancy equation $\sum_{index=1}^{MaxIndex} \pi_{index} \cdot Matrix_{Z,index} = Matrix_{Z,MaxIndex+1}$.

Step 1 (initializing): Set *maxIndex*= $\prod_{i \in M}(S_i + 1)$. Set $Matrix_{X,Y} = 0$ for all $X \in \{1,2,...,MaxIndex\}$ and for all $Y \in \{1,2,...,MaxIndex+1\}$. Set *index*=1.

Step 2 (find state): Use algorithm A6 to obtain the state ***x*** corresponding to *index*.

Step 3a (regular demands; initializing): Set *i*=1.

Step 3b (regular demands; select companies that face regular demands in this state): If x_i>0, perform steps 3c, 3d and 3e. Else, proceed to step 3f.

Step 3c (regular demands; add transition out): Set $Matrix_{index,index} = Matrix_{index,index} - \lambda_i$.

Step 3d (regular demands; find the index of the state with one less on-hand inventory for company *i*): Set $x_j^{temp} = x_j$ for all *j* in *M*. Subsequently set $x_i^{temp} = x_i^{temp} - 1$. Then use algorithm A7 to obtain the $index^{temp}$ corresponding to state $\boldsymbol{x^{temp}}$.

Step 3e (regular demands; add transition in): Set $Matrix_{index^{temp},index} = Matrix_{index^{temp},index} + \lambda_i$.

Step 3f (regular demands; iteration): Set *i*=*i*+1. If *i*≤*M* return to Step 3b, else proceed to step 4a.

Step 4a (transshipments; initializing): Set *i*=1.

Step 4b (transshipments; select companies that need transshipments in this state): If *index*≠1 (i.e. the state where no company has any on-hand inventory) and x_i=0, perform steps 4c, 4d, 4e and 4f. Else, proceed to step 4g.

Step 4c (transshipments; find company *j* that will source transshipment): Set $JJ = \{j \mid j \in (M \setminus i) \wedge c_{ji}^{trans} = \min_{j \in (M \setminus i)}(c_{ji}^{trans})\}$. Set $J = \{j \mid j \in JJ \wedge x_j = \max_{j \in JJ}(x_j)\}$. Set $j = \max(J)$.

Step 4d (transshipments; add transition out): Set $Matrix_{index,index} = Matrix_{index,index} - \lambda_i$.

Step 4e (transshipments; find the index of the state with one less on-hand inventory for company *j*): Set $x_k^{temp} = x_k$ for all *k* in *M*. Subsequently set $x_j^{temp} = x_j^{temp} - 1$. Then use algorithm A7 to obtain the $index^{temp}$ corresponding to state $\boldsymbol{x^{temp}}$.

Step 4f (transshipments; add transition in): Set $Matrix_{index^{temp},index} = Matrix_{index^{temp},index} + \lambda_i$.

Step 4g (transshipments; iteration): Set *i*=*i*+1. If *i*≤*M* return to Step 4b, else proceed to step 5.

Step 5a (repairs; initializing): Set *i*=1.

Step 5b (repairs; select companies that get repairs in this state): If $x_i<S_i$, perform steps 5c, 5d and 5e. Else, proceed to step 5f.

Step 5c (repairs; add transition out): Set $Matrix_{index,index} = Matrix_{index,index} - \mu_i \cdot (S_i - x_i)$.

Step 5d (repairs; find the index of the state with one more on-hand inventory for company *i*): Set $x_j^{temp} = x_j$ for all *j* in *M*. Subsequently set $x_i^{temp} = x_i^{temp} + 1$. Then use algorithm A7 to obtain the $index^{temp}$ corresponding to state $\boldsymbol{x^{temp}}$.

Step 5e (repairs; add transition in): Set $Matrix_{index^{temp},index} = Matrix_{index^{temp},index} + \mu_i \cdot (S_i - x_i)$.

Step 5f (repairs; iteration): Set *i*=*i*+1. If *i*≤*M* return to Step 5b, else proceed to step 6.

Step 6 (normalizing equation): Set $Matrix_{MaxIndex,X}$=1 for all $X \in \{1,2,...,MaxIndex+1\}$.

Step 7 (end): Return *Matrix*.

Algorithm A5: Gaussian Elimination to calculate the steady-state probabilities

Necessary input: *Matrix*, *m*.

Explanation: Recall from algorithm A4 that a "row" $Z \in \{1,2,...,MaxIndex\}$ of *Matrix* is the steady-state occupancy equation $\sum_{index=1}^{MaxIndex} \pi_{index} \cdot Matrix_{Z,index} = Matrix_{Z,MaxIndex+1}$. For notational ease, let *m*=*MaxIndex*. This Gaussian elimination algorithm takes a set of *m* equations with *m* unknowns: π_1 through π_m. It first eliminates π_1 from all equations below the first, then eliminates π_2 from all equations below the second, etc. This forward-elimination puts the system into triangular form. The second part of the algorithm, back-substitution, consists of solving for the unknowns in reverse order. Since Gaussian Elimination is a well-known algorithm, only a brief description will be given here.

Step 1a (forward elimination; initializing): Set i=1 and set j=1.

Step 1b (forward elimination; find pivot): Set $Maxi = \arg\max_{k \in \{i+1,i+2,...,m\}} (abs(Matrix_{k,j}))$.

Step 1c (forward elimination; row operations): If $Matrix_{i,j} \neq 0$ then swap rows *i* and *Maxi*, subsequently divide each entry in row *i* by $Matrix_{i,j}$, subsequently for all $r \in \{i+1, i+2,...,m\}$ subtract $Matrix_{r,j}$ ·(row *i* of *Matrix*) from (row *r* of *Matrix*), and finally set *i*=*i*+1.

Step 1d (iteration): Set j=j+1. If $i \le m$ and $j \le m+1$ then return to step 1b, else proceed to step 2a.
Step 2a (backwards elimination; initializing): Set r=m.
Step 2b (backwards elimination; start calculation): Set π_r=$Matrix_{r,m+1}$. Set k=m.
Step 2c (backw. el.; iterative calculation): If $k \ge 1+r$ then set $\pi_r = \pi_r - \pi_k \cdot Matrix_{r,k}$, subsequently set k=k-1 and finally do step 2c again; else proceed to step 2d.
Step 2d (backw. el.; iteration): Set r=r-1. If $r \ge 1$ return to step 2b, else proceed to step 3.
Step 3 (end): Return $\boldsymbol{\pi}$.

Algorithm A6: Obtaining the Markov state corresponding to its index number.
Necessary input: *index*, *M, **S***.
Explanation: We can pinpoint an array $\boldsymbol{x}$ of integers, $0 \le x_i \le S_i$, $i \in M$, with a unique index by the following formula: $index = 1 + \sum_{i=1}^{M} \left(x_i \cdot \prod_{j=i+1}^{M} (S_j + 1) \right)$ (where the empty product is 1), i.e. *index* iterates over all states in a semi-lexicographic way. We will illustrate how this formula works via an example. Let M=3 and let $\boldsymbol{S}$={2,4,9}. The minimum index is 1 and corresponds to $\boldsymbol{x}$={0,0,0}. Index 2 corresponds to $\boldsymbol{x}$={0,0,1}, index 10 corresponds to $\boldsymbol{x}$={0,0,9} and index 11 corresponds to $\boldsymbol{x}$={0,1,0}. State $\boldsymbol{x}$={0,3,6} corresponds to index 1+(3·10)+(6)=37. State $\boldsymbol{x}$={2,4,9} corresponds to the maximum index: 1+(2·5·10) +(4·10)+(9)=150. For this $\boldsymbol{S}$, 150 is used in other algorithms as the variable *MaxIndex*; it is the number of states in the Markov chain. Finding the state corresponding to an index number (i.e. the goal of this algorithm) is slightly more difficult and uses the notion of a vector ***indexEquivalent***. $indexEquivalent_i$ basically says that in order to reach a state with x_i=1 and x_j=0 for all $j \ne i$, we would have had to go over $indexEquivalent_i$ indices previously. For example, if $\boldsymbol{S}$={2,4,9} then in order to reach $\boldsymbol{x}$={0,1,0} we previously had {0,0,0}, {0,0,1}, …, {0,0,9}, which is 10 states and in order to reach $\boldsymbol{x}$={1,0,0} we previously had (4+1) ·(9+1) states, so $indexEquivalent_3$=50.
Step 1 (initializing): For all i in M, set x_i=0 and set $indexEquivalent_i = \prod_{j=i+1}^{M} S_j + 1$ (where the empty product is 1). Set i=1.
Step 2a (check whether the array associated with this remaining index reached an increase of array$_i$): If $index > indexEquivalent_i$ then proceed to step 2b. Else proceed to step 2c.
Step 2b (decrease remaining index and increase array): Set $index = index - indexEquivalent_i$, and subsequently set $x_i = x_i + 1$. Finally, return to step 2a.
Step 2c (iteration): Set i=i+1. If $i \le M$, return to step 2a. Else proceed to step 3.
Step 3 (end): return **x**.

Algorithm A7: Obtaining the index number corresponding to its Markov state.
Necessary input: ***x***, *M, **S***.
Step 1 (calculation): Return $1 + \sum_{i=1}^{M} \left(x_i \cdot \prod_{j=i+1}^{M} (S_j + 1) \right)$ (where the empty product is 1).

Appendix 13: Algorithms needed to calculate cost functions of a simple partial pooling game

Calculation of the cost functions for simple partial parts pooling game (which are covered in Chapter 7) involves creating a Markov chain and calculating steady-state probabilities. Furthermore, calculating optimal costs when base-stock levels are to-be-optimized involves finding an optimal solution in an infinite space (which can be bounded and enumerated efficiently). The algorithms that do this are quite briefly described in this Appendix and are implemented in a Java application. The description will be brief and informal, as most can already be inferred from slightly adjusting the algorithms that were presented in Appendices 8 and 12.

Algorithm A8: Calculating the minimum costs of a coalition M when S is to-be-optimized.

Necessary input: $M, (\lambda_i)_{i\in M}, \mu, (h_i)_{i\in M}, (c_i^{emer})_{i\in M}$.

This part is extremely similar to the part "*Starting at zero, increase S until S_M* and c(M) found*" described in Appendix 8. However, it obtains *minCosts* via a call to Algorithm A9 instead.

Algorithm A9: Calculating the costs of a coalition M for a given S.

Necessary input: $\boldsymbol{S}, M, (\lambda_i)_{i\in M}, \mu, (h_i)_{i\in M}, (c_i^{emer})_{i\in M}$.

This algorithm starts by creating a vector $\boldsymbol{T}\in\mathbb{N}_0^M$ such that T_i=0 for all *i* in *M* and calculating $Stot = \sum_{i\in M} S_i$. It then obtains a Markov chain via Algorithm A10 and finds steady-state probabilities $\boldsymbol{\pi}$ using Gaussian elimination. It then calculates corresponding costs using equation (7.3). Then it obtains the next $\boldsymbol{T}$ for which $0 \le T_i \le S_M$ for all $i \in M$, finds $\boldsymbol{\pi}$ and calculates costs again via (7.3). This continues until all $\boldsymbol{T}$ for which $0 \le T_i \le S_M$ have been checked and the minimum costs found for all those is returned.

Algorithm A10: Populating the Markov chain.

Necessary input: $Stot, M, (\lambda_i)_{i\in M}, \mu, (T_i)_{i\in M}$.

This function iterates over all states 0≤*x*≤*Stot*. For each state, it adds the transition types described in Section 7.4 to a *Matrix* containing the steady-state state occupancy equations of the Markov chain. So, this algorithm is very similar to Algorithm A4 (see Appendix 12), albeit with less complex transition types and with a simpler state space.