Cost allocation in cooperative transport: The joint network vehicle routing game

MSC ECONOMETRICS AND MANAGEMENT SCIENCE

M.A. VAN ZON (447708)

Erasmus University Rotterdam
Erasmus School of Economics
August 1, 2017

Supervisors: Dr. R. Spliet
Dr. W. van den Heuvel

Abstract

Cooperative transport provides the opportunity for logistic service providers to reduce costs while also receiving additional benefits such as a reduced carbon footprint. To this end, potentially competing companies are asked to join forces. This thesis strives to give an insight into the benefits of cooperative transportation by providing a mathematical foundation for cooperative transportation and visualising its results. In this manner, the potential benefits to each collaborator become more clear.

In this thesis a new cooperative game is introduced. The joint network vehicle routing game is a generalisation of the vehicle routing game. In this generalisation each player is allowed to have multiple locations. As the new game is a generalisation of the vehicle routing game, one can show that the core of the joint network vehicle routing game can be empty. Besides this, it is shown that the joint network vehicle routing game is monotonic, sub-additive and under some simplistic assumptions even essential.

The joint network vehicle routing game has been solved using a multitude of randomly generated instances, each with different characteristics. The amount of players, locations and locations per player have been varied along with the spacial structure of the instances. Then, each of the instances has been solved using a variety of solution algorithms. These solution algorithms are based upon relaxed as well as exact constraint and column generating techniques. Overall, the algorithms utilising constraint generation outperformed the algorithms that did not utilise constraint generation. Furthermore, it is shown that an average cost reduction from 17% up to 55% can be obtained for each of the involved players. The average savings are shown to depend heavily on the structure of the instances. Most savings are achieved if the locations of each player are not clustered but randomly located throughout the considered space.

Contents

1 Introduction

Collaborative transportation comes with great benefits. Through the utilisation of unused capacity and by combining delivery routes, transportation costs can be greatly reduced. Case studies as presented by Frisk et al. [18] and Engevall et al. [12] show that companies were able to achieve significant savings through collaboration. The benefits of such joint efforts are not just limited to cost reduction. Most of the transportation costs are directly related to fuel consumption. Because of this, collaborative transport is also likely to cause a decrease in CO_2 emissions. As such, an opportunity presents itself to decrease the carbon footprint and appeal to the current climate change movements while cutting costs. An example of this is shown by Stellingwerf et al. [45] in a case study for a Dutch wholesale company. Unfortunately, this relatively new style of transportation is not readily utilised. The collaborative technique is facing severe problems in the adoption process. Many companies do not want to exchange any information or allow their competitors to benefit from cooperative transportation. To stimulate the adoption, it is crucial to give companies insight into the way costs are allocated, and to create procedures to suit their needs. For this reason, the problem of assigning fair cost allocations in collaborative transport is considered.

The problem of assigning a fair cost allocation for cases in which each customer has exactly one location has been studied extensively in the literature. In this case, a customer is defined as an entity that provides transportation services for a set of locations. An example of a customer could be a grocery store that uses a vehicle to satisfy its demand from a central depot. The problem is approached through cooperative game theory and often referred to as the vehicle routing game. Game theory models cooperative problems using a group of rational entities, the customers. The group of all rational entities is also known as the grand-coalition. The theory assumes that each rational thinker solely tries to maximise its own profits. As collaborating poses an opportunity to reduce costs, all rational thinkers will at least consider collaborating. However, if any group of rational entities can cut more costs by collaborating with part of the grand-coalition rather than by collaborating with the entire grand-coalition, they may choose to leave the grand-coalition. Such behaviour is unwanted as the goal is to stimulate all rational thinkers to cooperate and to minimise the total costs. Finding a solution to stimulate all rational-entities into collaborating is the aim of the game. A solution to the game is also known as a cost allocation.

Due to the nature of game theory, the vehicle routing game can consist of multiple vehicle routing problems. In these vehicle routing problems the transportation costs of each subset of customers are optimised. This is done by finding optimal routes to serve each of the customers. Using these results, one can allocate to each of the customers a certain cost to stimulate the entire grand-coalition to cooperate.

Engevall et al. [13] consider a case study for a Norwegian company. In the case study, a multiple vehicle routing problem with one distribution centre is considered. The cost allocation problem is modelled using a vehicle routing game with heterogeneous fleet. The vehicle routing game used in their paper is based on the vehicle routing game as presented in Göthe-Lundgren et al. [25]. Engevall et al. propose column generation as preferred solution method for the underlying vehicle routing problems due to the high amount of feasible routes that need to be considered in each problem.

Frisk et al. [18] present a similar case study. They research cost allocation in collaborative forest transportation and introduce a new cost allocation method. The equal profit method (EPM) minimises the difference in relative cost reduction between all parties involved. Their EPM is easier to explain than most other methods and as such led to greater acceptance by the companies involved. This shows the importance of the simplicity of a method.

One of the most well known methods to allocate each customer a part of the total costs is known as the Nucleolus. The method as introduced by Schmeidler [41], utilises constraint generation to obtain a cost allocation. In their research they observe that the Nucleolus as well as the Shapley value, another important method to allocate costs as introduced by Shapley [42], are difficult to compute. The difficulty lies in the fact that both methods require the underlying vehicle routing problem to be solved for all combinations of customers. As such an exponential number of vehicle routing problems in terms of the amount of customers needs to be solved. For a couple of customers one can enumerate over all possible combinations of customers, however once the amount of customers increases, solving all problems becomes significantly more difficult. As an example, when considering 3 customers one would have to solve 8 problems, for 5 customers one needs to solve 32 problems and for 10 customers 1024 problems need to be solved. It can be seen that the amount of problems grows exponentially in the amount of customers. This shows the need for a more efficient solution method for the vehicle routing game.

A potential solution method presents itself in the form of constraint generation. While some methods, such as the Shapley value, require the optimal solution of all underlying vehicle routing problems, this is not the case for some of the other methods. The Nucleolus and Equal Profit Method require the optimal costs for each coalition as part of the constraints of their linear programming formulations. This means that the constraints in these methods can be generated and not necessarily every underlying vehicle routing problem has to be solved to optimality. This idea was first proposed by Göthe-Lundgren et al. [25]. In their paper, the authors apply a constraint generating method to solve the Nucleolus method as part of a travelling salesman game. A variant of the vehicle routing game in which only all customers have to be serviced using exactly one route.

Naber et al. [34] describe a model to allocate CO_2 emissions to customers on one distribution route. The aim of this research is to support environmentally concious companies to determine their total carbon footprint. To this end, they define a cooperative game using game theory. This game is closely related to the aforementioned vehicle routing game. Once more, they strive to minimise the total transportation costs. However, the research differs in the fact that they allocate vehicle emissions rather than costs. It should be noted that this also implies that they do not necessarily minimise the total vehicle emissions. In their paper, multiple allocation methods for CO_2 allocation are discussed. They compare the often used star method, which distributes the total CO_2 emission proportional to the stand-alone emission of each customer, to four other methods. These methods include the earlier mentioned Nucleolus and Shapley value. They find that all other methods outperform the star method in stability and robustness. However, the star method proves to be the most consistent method. The results obtained on the different allocation methods can be taken into consideration for choosing the most suitable allocation methods.

As can be seen, a lot of work has already been done in the field of allocating costs to customers serviced by one logistical service provider. Research aimed at stimulating collaborative efforts between the service providers themselves has not been actively performed. One of the early contributions in this research area is made by Wang et al. [48]. They consider the collaborative vehicle routing game with cost allocation between multiple distribution centres. To this end, they propose a hybrid approach with clustering, dynamic programming and a heuristic algorithm to solve the multi-depot vehicle routing problem. Thereafter, they use an improved Shapley value mode to distribute the gained profits among the distribution centres. This research could be utilised as a basis for cooperation between logistics service providers.

Most logistics service providers or large scale companies already have their own, potentially large, distribution network. By combining these networks less vehicles have to be used leading to a significant decrease in costs. However, allocating the costs to each network owner can be quite troublesome as the networks can include many locations on its own. Currently, not much effort has been put into designing an efficient and fair cost distribution for the network operators. The aim of this research is to enable companies and logistics service providers to combine their transportation networks and gain insight into how the costs are shared.

In this thesis the allocation of costs between different logistics service providers cooperating on multiple distribution routes is considered. To this end, multiple distribution centres can be used. The aim of this thesis is to design a fair and easily understandable allocation method to encourage cooperation between different companies. This implies that a new company willing to cooperate should not be allocated more costs than the costs without any form of cooperation. Adding a new partner to a distribution network in almost every case reduces the total costs. As such, the total costs for the other partners should also decrease in order to reflect an honest distribution of costs. Complying to these criteria should boost the willingness to adopt the new method. Furthermore, the computational performance of the method ought to be optimised. Finding an efficient algorithm for the new model is key in the adoption process. This research aims to overcome these inabilities to stimulate the currently low adoption rate. A key distinction with previously conducted research is that each company is allowed to have multiple locations in this research.

This thesis is structured in the following manner. The first two chapters cover the prerequisite knowledge required for this thesis. In Chapter 2 the basics of cooperative game theory are provided, here a thorough explanation of the core and its properties is provided. This chapter also includes an introduction to some of the most often used allocation methods. Thereafter, in Chapter 3 some prerequisite knowledge of the vehicle routing problem is presented along with a literature study on solution methods for the problem. Then, in Chapter 4 a new vehicle routing game is defined and its properties are derived. In Chapter 5 a solution procedure is proposed for the new joint network vehicle routing game. This includes an efficient method for the underlying vehicle routing problems and several approaches to solve the newly joint network vehicle routing game. Next, in Chapter 6, the experimental design of this thesis is discussed. To that end, the structure of the randomly generated instances used in this thesis is defined. Finally, the results are presented in Chapter 7 along with concluding remarks in Chapter 8.

2 Cooperative game theory

Game theory has been developed to study mathematical models concerned with conflict and cooperation between multiple rational decision-makers. Nowadays, game theory is widely utilised within the fields of economics, logic, political science, psychology, computer science and biology as described by Myerson [33]. A mathematical model concerned with conflict and cooperation between multiple rational decision-makers is also known as a game. In a game, the rational entities are also referred to as the players of the game. These players are said to be rational if the decisions made by the player are purely meant to support their own goals. For a company, a rational decision could be a decision to improve profit, or even to decrease greenhouse gas emissions. As long as the combined results from the decision supports its goals, the decision is said to be rational.

Some of the first applications of game theory revolved around the notion of zero-sum games which were developed by Neumann and Morgenstern [36]. The concept behind zero-sum games is that the loss and gain of all participants adds up to zero. As such, a gain for one player would result in a loss for another player. An example of this is presented by Haywood [27]. In this article the use of zero-sum games in military decision making is illustrated. Such games are often solved with the Nash equilibrium as introduced by Nash [35]. In his paper he also provides a clear mathematical foundation of non-cooperative games. In non-cooperative games, players are not allowed to join forces. In these games the sum of all gains and losses do not necessarily need to add up to zero. The total sum is dependent upon the behaviour on each of the players.

Another important class of games is the class of cooperative games. In cooperative games, players are allowed to join forces if the collaboration is beneficial for both players, which is in stark contrast to non-cooperative games. A formal definition of cooperative and non-cooperative games is given by Harsanyi [26]. Harsanyi states that a cooperative game is a game where commitments such as agreements, promises and threats are fully binding and enforceable. A cooperative game is defined as a game where commitments are fully binding. The foundation of cooperative game theory was laid by Neumann and Morgenstern [36]. One of the first applications of cooperative game theory is presented by Shubik [44]. In this paper the author uses game theory to distribute rewards among decision takers in a company proportional to the value created for the company. In 1959, Gillies [22] defined one of the most important solution concepts in cooperative game theory, the core.

In this chapter the basics of cooperative game theory are presented. First, some preliminaries as described by Branzei et al. [5] are discussed in Section 2.1. Next, an interpretation of the core and its properties is presented in Section 2.2. After that, some additional concepts of cost distributions are defined at the end of this chapter. Finally, five well-known cost distributions are presented in Section 2.3.

2.1 The basics

Consider a finite, non-empty set N. This set will be referred to as the grand coalition and consists of the players participating in the game. The set of all possible subsets of the grand-coalition, also known as the power set, is denoted as $\mathcal{P}(N)$. Note that the cardinality of the power set is $2^{|N|}$, here $|N|$ denotes the cardinality of the grand-coalition. Each element in the power set is referred to as a coalition S, as such $S \subseteq N$. The empty coalition consists of no players and is denoted as $\emptyset$. Besides this, players in a coalition are said to collaborate. Using these definitions a cooperative game can be formulated mathematically.

Definition 2.1. *A* ***cooperative game*** *is an ordered pair* $\langle N, c\rangle$ *consisting of the set of players* N *and the characteristic function* $c : \mathcal{P}(N) \to \mathbb{R}$ *with* $c(\emptyset) = 0$.

The characteristic function $c(S)$ can be interpreted in several ways. A common interpretation is the total cost savings of the players coalition S should they decide to form a coalition. An alternative interpretation, and the one used in this thesis, is that the function $c(S)$ denotes the total costs for a coalition S.

In the following a small example of the glove game is shown. The aim of the example is to illustrate what a game and a characteristic function might look like.

Example 2.1 (The glove game)**.** *Consider a non-empty, finite grand-coalition N. Each player $i \in N$ belongs to either the set L or the set R. As such, $L \cap R = \emptyset$. Players belonging to the set L have a left glove whereas players in R have a right glove. For each left glove that is not paired to right glove a cost of 1 is incurred and vice versa. One can derive that the characteristic function can be formulated as $c(S) = |S| - 2\min\{|S \cap L|, |S \cap R|\}$ for each coalition $S \in \mathcal{P}(N)$. In this example the characteristic function denotes the total costs of the members in a coalition. With this characteristic function the situation can be modelled as the game $\langle N, c\rangle$.*

The clove game as defined above lacks an objective. A possible objective in the clove game would be for each player to minimise its individual costs. However, in other games the objective could be chosen to maximise profit or net worth dependent on the choice of characteristic function. Similar to this example many other games can be defined. Every social interaction in which entities are allowed to cooperate can be modelled as a game. One could, for example, model people carpooling to work as a game in which each player aims to minimise its own costs.

In the following some of the basic properties of cooperative games are presented. An important note to be made is that some of the definitions are based on the interpretation that the characteristic function represents the total cost of a coalition. Some definitions may need to be adjusted should the characteristic function be interpreted otherwise.

Definition 2.2. *A game is called **monotonic** if the characteristic function satisfies the property that $c(S) \leq c(T)$ for all $S \subseteq T$ with $S, T \in \mathcal{P}(N)$.*

The monotonic property states that adding a player to a coalition in a monotonic game always leads to a cost increase. A special type of monotonic games are games in which adding a customer to a coalition raises the cost by the exact stand-alone costs of that customer. This could be the case if the players do not influence each other whatsoever. These games are called additive and are defined as follows:

Definition 2.3. *A game is called **additive** if the characteristic function satisfies the property that $c(S \cup T) = c(S) + c(T)$ for all $S, T \in \mathcal{P}(N)$ such that $S \cap T = \emptyset$.*

In an additive game, the cost of adding a player to any coalition is equal to its stand-alone cost. This means that the cost of any coalition is equal to the sum of the stand-alone costs of each player. As such, no matter the objective of a game, the choices made have no influence: there is no benefit to cooperating. In the ideal situation, adding a player to a game increases the costs by less than the stand-alone costs of that player. This idea is made formal in the following definition:

Definition 2.4. *A game is called **sub-additive** if the characteristic function satisfies the property that $c(S \cup T) \leq c(S) + c(T)$ for all $S, T \in \mathcal{P}(N)$ such that $S \cap T = \emptyset$.*

Hence, if a game is sub-additive the total cost of a coalition is always less than or equal to the sum of the stand-alone costs for each customer. This is a desirable property for a game because in this case players do benefit from collaboration. An even stronger property is the concavity of a game. It is defined as follows:

Definition 2.5. *A game is called* ***concave*** *if the characteristic function satisfies the property that* $c(S \cup T) + c(S \cap T) \leq c(S) + c(T)$ *for all* $S, T \in \mathcal{P}(N)$.

A game is only worth solving if the costs of the grand-coalition N are strictly less than the sum of the stand-alone costs of the customers. Such a game is called essential.

Definition 2.6. *A game is called* ***essential*** *if the characteristic function satisfies the following property:*

$$c(N) < \sum_{i \in N} c(\{i\}).$$

After determining the total costs, these should be divided among the coalition members. This is done through a mapping also known as a cost distribution.

Definition 2.7. *A* ***cost distribution*** *is a mapping* $f : G^N \to \mathbb{R}^{|N|}$. *Here* G^N *is the collection of characteristic functions of cooperative games with player set* N.

In essence the cost distribution f maps total costs for a coalition to the individual contributions expected of each player. Player i will be allocated $f(c)_i$ as costs for $c \in G^N$. In Section 2.3 the different kinds of cost distributions are thoroughly discussed. For now, some of the most important properties of a cost distribution are defined.

Definition 2.8. *A cost distribution* $f : G^N \to \mathbb{R}^{|N|}$ *can satisfy the following properties:*

1. ***individual rationality*** *if* $f(c)_i \leq c(i)$ *for all* $c \in G^N$.
2. ***coalitional rationality*** *if* $\sum_{i \in S} f(c)_i \leq c(S)$ *for all* $c \in G^N, S \subseteq N$.
3. ***efficiency*** *if* $\sum_{i \in N} f(c)_i = c(N)$ *for all* $c \in G^N$.
4. ***additivity*** *if* $f(c + d) = f(c) + f(d)$ *for all* $c, d \in G^N$.

The first property ensures that for each player the allocated cost is less than its stand-alone cost. Any rational entity would not join a coalition if its costs increase as a result. A stronger version of this property is the coalitional rationality. This property states that a coalition S in the grand-coalition N should never pay more $c(S)$. If S would pay more than $c(S)$, then coalition S would be better off by forming their own grand-coalition. The third property ensures that all costs are split among the members of a coalition. Suppose a set of players belong to two separate games. The additivity property states that the players are allocated the same amount of costs if these games were to be merged. Finally, the definition of a dummy player is provided.

Definition 2.9. *A* ***dummy player*** *in a game* $\langle N, c \rangle$ *is a player* $i \in N$ *such that* $c(S \cup \{i\}) = c(S) + c(i)$ *for all* $S \subseteq N \backslash \{i\}$.

Adding a dummy player to a coalition will not decrease the cost for other coalition members as well as the dummy player itself. This means that dummy players are very undesirable entities and can be removed from the grand-coalition as they do not contribute towards a better cooperative solution.

2.2 The core

The concept of the core has been introduced by Gillies [22] in 1959. The core is a description of all possible cost allocations that are deemed agreeable by all players. A cost allocation is a vector which results from applying a cost distribution to a game. The cost allocation describes how the cost is allocated to each of the customers. This section is devoted to defining the core and its properties. First, the concept of a cost vector or cost allocation is formalised. Notice that the concepts presented here are related closely to the concepts presented for the map f.

Definition 2.10. *A* ***cost-vector*** *or* ***cost allocation*** *is a vector $\boldsymbol{x} \in R^{|N|}$ where x_i denotes the costs allocated to customer $i \in \mathbb{N}$. For each cost-vector $\boldsymbol{x}$ it should hold that $\sum_{i \in N} x_i \geq c(N)$.*

A cost allocation can have properties similar to those of the cost-distributions. Each of the following properties corresponds to one of the properties of the cost distributions. The first of these properties is the efficiency property.

Definition 2.11. *A cost allocation is said to be* ***efficient*** *if the following statement holds:*

$$\sum_{i \in N} x_i = c(N) \tag{2.1}$$

An efficient cost allocation ensures that the total costs are allocated among the players while no excess costs are located to any of the players. Note that an efficient cost distribution ensures that all resulting cost allocations are efficient. Next are the individual and coalitional rationality properties.

Definition 2.12. *A cost allocation is said to be* ***individually rational*** *if the following statement holds for all players $i \in N$:*

$$x_i \leq c(\{i\}) \tag{2.2}$$

Definition 2.13. *A cost allocation is said to be* ***coalitionally rational*** *if the following statement holds for all coalitions $S \in \mathcal{P}(N)$:*

$$\sum_{i \in S} x_i \leq c(S) \tag{2.3}$$

Both of the rationality properties should be satisfied for a cost allocation to be accepted by all players. Without these properties a group of players might be better of by leaving the grand-coalition and forming their own grand-coalition.

Now the principle of the core can be defined. The core is the set of cost-vectors $\boldsymbol{x}$ which gives none of the players an incentive to leave the grand coalition. To that purpose a cost allocation ought to be individual rational as well as coalitionally rational. Besides this, a cost allocation is also required to be efficient. The core is the collection of all of these cost allocations and is defined as follows:

Definition 2.14. *The* ***core*** *of game $\langle N, c \rangle$ is the set of all individual and coalitional rational cost-vectors. Furthermore, the total costs $c(N)$ should be covered by the individual contributions x_i. The definition can be formalised as follows:*

$$\mathit{Core}(\langle N, c \rangle) = \left\{ \boldsymbol{x} \in \mathbb{R}^{|N|} : \sum_{i \in S} x_i \leq c(S)\ \forall S \in \mathcal{P}(N), \sum_{i \in N} x_i = c(N) \right\}$$

The main idea behind the core is that any cost allocation in the core gives none of the players an incentive to leave the grand-coalition. All the cost allocations in the core are deemed acceptable. Which core allocation to choose, should the core be empty, is discussed in Section 2.3. Unfortunately, it can occur that the core of a game is empty. This is illustrated by the following instance of the glove game:

Example 2.2 (The glove game)**.** *Consider an instance of the glove game as introduced in Example 2.1. Let $N = \{1,2,3\}$, $L = \{1\}$ and $R = \{2,3\}$. The costs for each coalition $S \in \mathcal{P}(N)$ are given in Table 2.1.*

Table 2.1: The costs for each coalition.

Coalition	Cost
1	1
2	1
3	1

Coalition	Cost
1,2	0
2,3	0
1,3	0

Coalition	Cost
1,2,3	1

For any rational cost allocation the following equations should hold:

$$\begin{aligned} x_1 + x_2 &\leq c(\{1,2\}) = 0 \\ x_2 + x_3 &\leq c(\{2,3\}) = 0 \\ x_3 + x_1 &\leq c(\{3,1\}) = 0 \end{aligned}$$

Adding these equations yields that $2(x_1 + x_2 + x_3) = 0$ and thus implies that $x_1 + x_2 + x_3 = 0 < c(\{1,2,3\}) = c(N)$. This means that any cost allocation that is coalitionally rational cannot be efficient. Now it can be concluded that the core is empty for this game.

In order to determine if a solution in the core exists, one should determine whether the core is non-empty. There are multiple ways to conclude that the core is non-empty. One of the theorems to conclude that a core is empty uses the concept of balanced games. Define $\boldsymbol{e_S}$ as the indicator vector, $(e_S)_i = 1$ if $i \in S$ and 0 otherwise. Now a balanced game can be defined as follows.

Definition 2.15. *A game is* ***balanced*** *if for each map $\lambda : \mathcal{P}(N)\backslash\{\emptyset\} \to \mathbb{R}_+$ such that $\sum_{S\in\mathcal{P}(N)\backslash\emptyset} \lambda(S)\boldsymbol{e_S} = \mathbf{1}$ it holds that:*

$$\sum_{S\in\mathcal{P}(N)\backslash\emptyset} \lambda(S)c(S) \geq c(N).$$

Concluding that a game is balanced is the first way in which one can conclude that the core of a game is non-empty. This result is formally stated in the next theorem.

Theorem 2.1. *The core of a game is non-empty if and only if the game is balanced.*

Proof. The proof can be found in [5]. □

Another way to conclude that the core of a game is non-empty is to use the fact concave games are balanced. This result is formalised in the following theorem.

Theorem 2.2. *The core of a game is non-empty if the game is concave.*

Proof. The proof can be found in [43]. Note that the paper discusses the characteristic function v. This function describes the savings rather than the costs of each coalition. Hence, stating that c is sub-additive and concave is equal to stating that v super-additive and convex as discussed by Shapley. This means that the results obtained for the convex v also hold in the case as described in this thesis. □

As the core is not always non-empty, an alternative core has been introduced. The so-called strong-epsilon core is discussed by Maschler et al. [31]. The principle behind this core is that leaving the grand-coalition is punished with a penalty ε. The strong-epsilon core is formalised in the following definition.

Definition 2.16. *The **strong-epsilon core** of a game $\langle N, c\rangle$ is defined as follows:*

$$Core_{\varepsilon}(\langle N, c\rangle) = \left\{ \boldsymbol{x} \in \mathbb{R}^{|N|} : \sum_{i \in S} x_i \leq c(S) + \varepsilon \; \forall S \in \mathcal{P}(N), \sum_{i \in N} x_i = c(N) \right\}.$$

This core enforces collaborations in otherwise unstable grand-coalitions. However, it does not stimulate the formation of any new coalitions. A special case of the strong-epsilon core is the least core. The least core is defined as the intersection of all non-empty strong epsilon cores. In essence, this is the strong-epsilon core for the minimum epsilon such that the strong-epsilon core is non-empty.

Definition 2.17. *The **least-core** is defined as the intersection of all non-empty strong epsilon cores.*

Besides this, one more type of core is defined. This core is once more a relaxation of the original core.

Definition 2.18. *Let $\Omega \subseteq \mathcal{P}(N)$, the **$\boldsymbol{\Omega}$-core** of a game $\langle N, c\rangle$ is defined as follows:*

$$Core_{\Omega}(\langle N, c\rangle) = \left\{ \boldsymbol{x} \in \mathbb{R}^{|N|} : \sum_{i \in S} x_i \leq c(S) \; \forall S \in \Omega, \sum_{i \in N} x_i = c(N) \right\}.$$

Finally, some additional definitions are required to fully understand the allocation methods proposed in Section 2.3. First of all, the concept of stability is defined.

Definition 2.19. *A cost distribution $f : G^N \to \mathbb{R}^{|N|}$ is **stable** if for a balanced game $\langle N, c\rangle$ it holds that $f(c) \in Core(\langle N, c\rangle)$.*

Furthermore, if the solution is in the core the individual and coalitional rationality and efficiency must hold. Thus it can be concluded that a stable allocation implies these three properties. Another aspect to take into account is the **uniqueness** of the solution obtained by an allocation method. If the method does not yield a unique solution the cost allocation may differ each time the method is executed. As such, it is preferred that a cost distribution has the uniqueness property.

2.3 Cost distributions

Cost allocations indicate how much of the total costs are assigned to each player. In order to ensure that no player pays more than can reasonably be expected of him, the core was defined. Any cost allocation in the core is deemed to be agreeable for all of the players involved in the game. However, even if fair allocations in the core exist, there is no single method to determine the best cost allocation. For each situation a different cost allocation may be chosen dependent on the wishes of the grand-coalition. Some coalitions may prefer to minimise the difference in relative cost savings whereas others may choose to distribute the costs proportional to the stand-alone costs. These are only two examples of the many mappings that exist. Each of these distributions have very different characteristics. Some distributions may be stable, whereas others can guarantee unique solutions. Besides this, some of the cost distributions may be substantially more complex to calculate than others. As such, one can not just state that any of the cost distributions is ideal. To this end some of the most used cost distributions are introduced. These are the Nucleolus, Shapley value, Equal Profit Method, Lorenz and Star method.

The different cost distributions are thoroughly discussed in the next subsections. In each subsection exactly one of the earlier mentioned cost distributions is presented. This includes a formal definition of the general solution procedure and a short analysis on the properties that each mapping possesses. In order to effectively compare the cost distributions, an overview of the properties of the five methods is provided in Subsection 2.3.6 along with a small example. The aim of the example is to provide a bit more insight in the different cost allocations that result from each of the methods. Besides this, the example will also illustrate the procedures corresponding to the more complex methods.

2.3.1 Nucleolus

The nucleolus has been introduced by Schmeidler [41] in 1969. The method ensures that the coalitions with maximum profit retain their profit margins. If the core turns out to be empty, this can result in some coalitions becoming less profitable.

The iterative linear programming problem to find the Nucleolus is defined as in Engevall et al. [12]. Each iteration can be formulated as its own linear programming problem with decision variable z. In this case z denotes the maximum excess according to allocation $\boldsymbol{x}$ for all coalitions S in $N \setminus \bigcup_{l<m}[F_m]$. Here F_m is the set containing the coalitions S for which the excess equals, z_m, the maximum excess determined in iteration m.

The set F is initialised as $F_1 = \emptyset$. In each iteration l a new maximum excess is determined. This excess is then fixed for the corresponding coalitions by adding them to F_l. After this, the procedure is repeated until all coalitions have been considered. Now, for each iteration $l \in \mathbb{N}$ the following problem is solved:

Nucleolus iterative problem

Objective:
$$z_l = \max z \tag{2.4}$$

Subject to:
$$x_i \leq c(\{i\}) \quad \forall i \in N \tag{2.5}$$
$$\sum_{i \in S} x_i + z \leq c(S) \quad \forall S \subseteq N, S \in N \setminus \bigcup_{l<m} F_m \tag{2.6}$$
$$\sum_{i \in S} x_i + z_m = c(S) \quad \forall m < l : S \in F_m \tag{2.7}$$
$$\sum_{i \in N} x_i = c(N) \tag{2.8}$$

$$x_i \in \mathbb{R}^+ \quad \forall i \in N \tag{2.9}$$
$$z \in \mathbb{R} \tag{2.10}$$

Constraints (2.5) ensure that no customer is allocated more costs than the stand-alone costs. This ensures that the cost allocation is individually rational. Constraints (2.6) are used to find the maximum value of the excess, note that all previously found coalitions are excluded. Constraints (2.7) ensure that all previously considered coalitions F_m retain their excess z_m as determined during the iterations. Lastly, Constraint (2.8) ensures that the allocation is efficient. Denote the optimal dual variables corresponding with Constraints (2.6) as $u_l(S)$. After each iteration, the coalitions for which $u_l(S) > 0$ are added to set F_l. Göthe-Lundgren et al. [25] suggest that the Nucleolus can be efficiently solved using a constraint generation approach.

The nucleolus is in the core if it is non-empty. If the core is empty, the excess can be negative. This means that the solution can become unstable. But this is of-course to be expected if the core is empty. Furthermore, the solution to the problem is always determined through the described procedure and is guaranteed to be unique.

2.3.2 Shapley value

The Shapley value has been introduced by Shapley [42]. This method allocates the cost corresponding to the marginal cost of a customers. This marginal cost can be obtained by looking at the optimal costs of all coalitions without a certain player and observing the change in optimal cost after adding the player. Thereafter, the marginal costs over all coalitions are averaged, yielding the cost allocation. The Shapley value is defined as follows:

$$x_i = \sum_{S \subseteq N \setminus \{i\}} \frac{|S|!(n-|S|-1)!}{n!} [c(S \cup \{i\}) - c(S)] \tag{2.11}$$

The allocation is not necessarily in the core. However, for concave games it is in the core as shown by Shapley [43]. This means that the method always yields a stable solution for concave games. Besides this, the solution obtained is also unique. The largest downside of the method is the computational effort required for large instances. This is due to the fact that one has to iterate over each subset to determine the optimal costs.

2.3.3 Equal profit method

The Equal Profit Method (EPM) has been introduced by Frisk et al. [18] in 2010. They found that companies had some difficulties understanding the different, available cost allocations. It turned out that companies did not understand why some coalition members would receive higher relative savings in comparison to others. As such, they designed a method to maximise the acceptance. The allocation provided by the EPM aims to minimise the maximum difference between the allocations relative to stand-alone costs of each customer. The allocation is determined using the following linear programming problem:

Equal profit method

Objective: $\min z$ (2.12)

Subject to:

$$\frac{x_i}{c(\{i\})} - \frac{x_j}{c(\{j\})} \leq z \quad \forall i,j \in N \tag{2.13}$$

$$\sum_{i \in S} x_i \leq c(S) \quad \forall S \subseteq N \tag{2.14}$$

$$\sum_{i \in N} x_i = c(N) \tag{2.15}$$

$$x_i \in \mathbb{R}^+ \quad \forall i \in N \tag{2.16}$$

$$z \in \mathbb{R}^+ \tag{2.17}$$

The maximum relative difference in the cost allocation is denoted as z, Constraints (2.13) enforce this. Constraints (2.14) and (2.15) ensure that the solution is in the core. This also means that the LP is infeasible if the core is empty. Due to Constraints (2.13) it holds that $\frac{x_i}{c(\{i\})} - \frac{x_j}{c(\{j\})} \leq z$ and $\frac{x_j}{c(\{j\})} - \frac{x_i}{c(\{i\})} \leq z$ for all players i and j. Due to this, the solution does not need to be unique as one could potentially determine alternate allocated values for players i and j to still satisfy both constraints. Note that existence of these alternate values depend on the structure of the core.

2.3.4 Lorenz allocation

The Lorenz allocation has been introduced by Arin [2] in 2003. This allocation tries to minimise the smallest difference between the absolute allocations rather than the relative difference as the EPM does. The Lorenz method can be represented by the following linear programming problem:

Lorenz allocation

$$\textbf{Objective:} \quad \min z \tag{2.18}$$

$$\textbf{Subject to:} \quad x_i - x_j \leq z \quad \forall i, j \in N \tag{2.19}$$

$$\sum_{i \in S} x_i \leq c(S) \quad \forall S \subseteq N \tag{2.20}$$

$$\sum_{i \in N} x_i = c(N) \tag{2.21}$$

$$x_i \in \mathbb{R}^+ \quad \forall i \in N \tag{2.22}$$

$$z \in \mathbb{R}^+ \tag{2.23}$$

The maximum absolute difference in the cost allocation is denoted as z, Constraints (2.13) enforce this. Constraints (2.14) and (2.15) ensure that the solution is in the core. This means that the LP is infeasible if the core is empty. Both the Lorenz and Equal Profit Method are not unique as there could be multiple solutions to the LP. One can see this using a similar reasoning to the one used for the Equal Profit Method. Suppose a value for z corresponding to an optimal solution has been determined and consider players i and j. The constraint $x_i - x_j = z$ or the constraint $x_j - x_i = z$ must hold. Under the assumption that both solutions satisfy the core constraints it has been established that the solution is not unique.

2.3.5 Star method

The Star method is dicussed in Naber et al. [34]. The method allocates the total cost proportional to the stand-alone costs of all customers. This leads to the following expression for the costs allocated to a customer:

$$x_i = \frac{c(i)}{\sum_{i \in N} c(i)} c(N). \tag{2.24}$$

This method is by far the easiest to understand and yields a unique cost allocation. The largest downside of the method is that the solution is not necessarily in the core, should the core be non-empty.

2.3.6 A comparison

The properties of the different cost distributions are summarised in Table 2.2. It can be seen that each cost distribution satisfies the efficiency property, as such the cost distributions always allocate all costs over the customers. Furthermore, the Nucleolus, Equal Profit Method and Lorenz method guarantee a solution in the core if the core exists. This means that any allocation by these methods is expected to be fair according to the principles of the core. The Shapley value can only guarantee a solution in the core if the game is concave. However, this is often not the case. Another important aspect is the uniqueness of the solution. This is only guaranteed by the Nucleolus, Shapley value and the Star method. This means that the other cost distributions may yield a different allocation each time the method is executed. This is quite hard to explain to any rational thinker and as such not wanted.

Table 2.2: An overview of the discussed methods and their properties. Properties with a * only apply for concave games.

Method	Stability	Additivity	Efficiency	Coalitional rationality	Uniqueness
Nucleolus	✓		✓	✓	✓
Shapley	*	✓	✓		✓
Equal profit method	✓		✓	✓	
Lorenz	✓		✓	✓	
Star		✓	✓		✓

In order to illustrate each of the aforementioned cost distributions, a simple example is provided below.

Example 2.3. *Consider the game $\langle N, c\rangle$ where $N = \{1, 2, 3\}$. Furthermore, the characteristic function $c(S)$ is given in Table 2.3.*

Table 2.3: The costs for each coalition in the game.

Coalition	Cost	Coalition	Cost	Coalition	Cost
1	10	1,2	13	1,2,3	18
2	10	2,3	15		
3	6	3,1	15		

Now, for each cost distribution the costs allocated to each player are determined. Applying the cost distributions as defined above leads to the following results as displayed in Table 2.4. The derivations of the non-trivial solutions are presented Appendix A.

Table 2.4: The result of applying the cost distributions.

Method	x_1	x_2	x_3	Core
Stand-alone	10	10	6	
Nucleolus	6.25	6.25	5.5	✓
Shapley value	6.33	6.33	5.33	✓
Equal profit method	6.5	6.5	5	✓
Lorenz	6	6	6	✓
Star	6.92	6.92	4.15	

It can be seen that all cost distributions except for the star-method yield a cost allocation in the core. The allocation provided by the equal profit method and the Lorenz method may however be rejected by some of the parties. For the equal profit allocation it holds that players 1 and 2 do not profit from the addition of player 3. This in contrast to the Lorenz allocation where player 3 does not profit from adding player 1 and 2 to its coalition. Both the Nucleolus and Shapley value yield suitable allocations, none of the players are have any incentive to reject these solutions.

3 The vehicle routing problem

In 1959 the now well-known vehicle routing problem was introduced as the truck dispatching problem by Dantzig and Ramser [9]. Dantzig and Ramser state that the problem may be considered as a generalisation of the travelling salesman problem. Later, the problem was generalised to a linear optimisation problem by Clarke and Wright [8]. The vehicle routing problem concerns itself with finding the set of least expensive routes to serve a set of customers. Each of the customers has a demand that has to be satisfied by a centralised depot using a set of identical vehicles with a maximum capacity. Furthermore, the problem is known to be NP-hard as concluded by Lenstra and Kan [29].

Throughout the years various variants of the problem have been introduced. A Notable variant of the vehicle routing problem is the vehicle routing problem with time-windows. In the vehicle routing problem with time windows each customer has a certain time window during which they need to be serviced. Besides this variant, many other variants of the vehicle routing problem exist.

A lot of research has been performed on the vehicle routing problem. Braekers et al. [4] present a literature overview of the most recent advancements in the field of vehicle routing. A more complete overview of the literature is presented in Golden et al. [24]. The aim of this chapter is to introduce the vehicle routing problem. To this purpose some basics of graph theory are introduced in Section 3.1. Then, using graph theory a formal mathematical description of the problem is presented in Section 3.2. Finally, a literature study on the different models and methods to solve the vehicle routing problem is presented in Section 3.3.

3.1 Graph theory

In this section a short introduction to graph theory is provided. The presented definitions are in accordance to Papadimitriou and Steiglitz [37]. First, the concept of a graph is formalised.

Definition 3.1. *A **graph** is an ordered pair $G = (V, E)$ consisting of the set V of vertices or nodes and the set E consisting of edges. An edge is a subset of the set of vertices V with cardinality 2.*

If $e = (v_1, v_2) \in E$ it is said that v_1 is **adjacent** to v_2 and vice versa. Using this property, the **degree** of a node is defined as the amount of adjacent nodes. In some cases one might not want a connection between nodes in both directions. For this purpose the directed graph is defined.

Definition 3.2. *A **directed graph** is an ordered pair $D = (V, A)$ consisting of the set V of vertices or nodes and the set A consisting of arcs. An arc is an ordered pair of vertices, hence $A \subseteq V \times V$.*

Contrary to a graph, no degree can be defined for any node in a directed graph. However, similar relations based on the amount of arcs towards and away from exist. In a directed graph $D = (V, A)$ the **indegree** of a node $v \in V$ is the number of arcs of the form (u, v) in A. Similarly, the **outdegree** of a node $v \in V$ is the number of arcs of the form (v, u) in A. Now different types of walks can be defined.

Definition 3.3. *A **directed walk** in a directed graph D is a sequence $[v_1, v_2, \ldots, v_k]$ of nodes in V with $k \geq 1$ such that $(v_j, v_{j+1}) \in A$ for $j = 1, \ldots, k-1$.*

In directed walks it can occur that some nodes are visited more than once, in many cases this is unwanted. A directed walk which visits each node at most once is called a directed path.

Definition 3.4. *A **directed or elementary path** in a directed graph D is directed walk $[v_1, v_2, \ldots, v_k]$ such that no nodes are repeated.*

A special kind of directed path is a directed path which starts and ends at the same node. Such a path is also known as a cycle.

Definition 3.5. *A **cycle** in a directed graph D is directed path $[v_1, v_2, \ldots, v_k]$ such that $k > 1$ and $v_1 = v_k$.*

3.2 Problem description

In this section a formal problem description of the vehicle routing problem is provided. To that end, the definitions of the previous section are utilised. In the vehicle routing problem a set of customers each with a demand have to be serviced from a centralised depot v_d. Each of these customers are represented by a node v_i with $i \in \mathbb{N}$. Furthermore, the demand of each customer i is denoted as d_i. Between each node $v_i, v_j \in V$ an arc may exist, the set of all of these arcs is denoted as A. Each arc $a = (v_i, v_j) \in A$ has a cost of c_{ij}. Now the network belonging to the vehicle routing problem can be defined as follows.

Definition 3.6. *A **network** $W = (v_d, V, A, c, d, Q)$ is a directed graph $D = (V, A)$ in which v_d represents the depot node, c_{ij} represents the cost of arc $(i, j) \in A$, Q represents the maximum arc capacity and d_v represents the demand of a node $v \in V$.*

In the vehicle routing problem customers can be serviced using an infinite homogeneous fleet of vehicles with capacity Q. Any vehicle is allowed to serve exactly one set of customers on a route.

Definition 3.7. *A **route** in a network $W = (v_d, V, A, c, d, Q)$ is a cycle $[v_1, v_2, \ldots, v_k]$ in the directed graph $D = (V, A)$ such that $v_1 = v_d$.*

Corresponding to each route is a cost. This cost is characterised by the arcs used in the route. let $w = [v_1, v_2, \ldots, v_k]$ be a route in a network $W = (v_d, V, A, c, d, Q)$. The cost of this route can be denoted as:

$$cost(w) = \sum_{j=1}^{k-1} c_{j,j+1} \tag{3.1}$$

However, not all all routes may be feasible as the total demand of the customers visited on a route can exceed Q. As such, not all routes are feasible for the problem.

Definition 3.8. *A **feasible route** for a capacity Q is a route in a network $W = (v_d, V, A, c, d, Q)$ such that the cumulative demand of all visited nodes does not exceed the capacity Q.*

Using this definition of feasible routes the purpose of the vehicle routing game can be defined. The goal is to determine a set of routes K that visits all customers exactly once, such that the cumulative costs of the routes are minimised. This minimum cost is denoted as $VRP(W)$. Besides this, the minimum cost to visit all customers exactly once, allowing each route to be used a fractional amount of times, is denoted as $\overline{VRP(W)}$.

3.3 Literature review

The aim of this section is provide a detailed overview of known solution procedures for the vehicle routing problem. If the vehicle routing problem concerns a really small amount of customers, the solution can easily be determined by enumerating all possible combinations of feasible routes. However, in the worst case scenario, the amount of feasible routes can grow exponentially as a function of the amount of customers. This means that other, more clever, solution procedures are required to solve larger instances in a reasonable amount of time.

In the first approach, the vehicle routing problem with homogeneous fleet is modelled as a flow model. Here the flow corresponds to the arcs used on a route. As such, the formulation uses binary variables to indicate if a vehicle travels between two customers. The resulting model is a mixed integer programming problem. Originally, this formulation was proposed by Garvin et al. [19] to model the flow of oil. Then the formulation was adapted and extended by Gavish and Graves [20].

Gheysens et al. [21] present a similar model, but for the vehicle routing problem with heterogeneous fleet. They introduce a flow variable with three indices, indicating if a truck of a certain type travels between two customers. The authors suggest the use of heuristics to solve the problem. To that end they review existing heuristics and compare these to a new heuristic they present themselves. Golden et al. [23] et al. propose an approach similar to that of Gheysens et al. [21] for the the vehicle routing problem with heterogeneous fleet. However, in their approach the capacity and sub-tour constraints are modelled using an extension of the Miller-Tucker-Zemlin inequalities as proposed in Miller et al. [32].

Balinski and Quandt [3] propose a substantially different way of modelling the vehicle routing problem. They introduce the set partitioning model. In this model each feasible route is associated with a binary variable. Through these binary variables it is ensured that each customer is visited at least once. The major drawback of this method is that listing all feasible routes is very inefficient for large instances of the problem. As such the linear programming relaxation of the set partitioning formulation is often solved using column generation. This way columns can be generated as required and do not need to be readily available. An introduction to the principles of column generation is provided by Desaulniers et al. [10]. The generation of columns gives rise to the so-called pricing problem. Desrochers et al. [11] suggest a labelling algorithm to solve the pricing problem efficiently. However, they relax the set partitioning formulation in such a manner that each route can visit a customer multiple times. Feillet et al. [15] argue that this weakens the lower bound given by the relaxation. As such they propose an extended algorithm for the pricing problem such that only routes that are feasible to the non-relaxed problem are generated. Often, column generation is utilised within a branch-and-bound procedure to attain the exact solution to the vehicle routing problem. This approach is also known as the branch-and-price method and is described by Feillet [14].

Due to the intrinsic complexity of the vehicle routing problem it is often solved using heuristics. For large instances of the problem, heuristics prove to be the only realistic solution method. As such, some heuristics for the vehicle routing problem are discussed. An overview of the different heuristics along with a comparison of the performance is presented in Golden et al. [24]. The heuristics are based on the vehicle routing problem with heterogeneous fleet. Note that these heuristics can also be applied to the vehicle routing problem with homogeneous fleet as it is a special case of the vehicle routing problem with heterogeneous fleet.

In the comparison as presented in Golden et al., the record-to-record heuristic proposed by Li et al. [30] stands out as the best construction heuristic. Besides this, the meta-heuristic proposed by Pisinger and Ropke [38] for the single depot vehicle routing problem outperforms all previously performed methods in any of the instances used. They modelled the single depot vehicle routing problem as a rich pick-up and delivery problem with time windows and solved it using an adaptive large neighbourhood search algorithm. Note that only heuristic methods which where published before 2009 have been considered in this comparison.

4 The joint network vehicle routing game

Consider a group of logistics service providers, all of whom already have their own customers and logistic networks. Suppose these logistics service providers decide to join forces, and as such form a coalition. Now, the total costs need to be allocated among the members of the coalition. As each logistics service provider comprises of multiple customers, it is rather difficult to precisely determine for which part of the total costs each logistic service provider is responsible. In order to tackle this problem, a new game is designed using game theory.

In this chapter the joint network vehicle routing game is introduced. The aim of this chapter is to provide a complete overview of the game and its characteristics. To this end, in Section 4.1, a clear mathematical definition for the joint network vehicle routing game along with an illustrative example of an instance of the game is provided. Next, in Section 4.2, special instances of the joint network vehicle routing game are presented. These games are known in the literature as the vehicle routing game and the travelling salesman game. Finally, in Section 4.3 some properties of the joint network vehicle routing game are derived and presented.

4.1 Definition of the game

In this section a mathematical formulation of the joint network vehicle routing game is presented. To this end, the principles of game theory as defined in Chapter 2 are utilised. First, define the set of players as N. The players in the game are defined as different logistics service providers. This implies that each player already services multiple customers. The set of customers belonging to a player i is denoted as V_i. Each customer v has its own demand d_v and locations are serviced from a single centralised depot v_d. It is assumed that the depot has an unlimited homogeneous fleet of vehicles with capacity Q available.

Next, the characteristic function $C(S)$ is defined for each coalition S. Consider a coalition $S \subset N$ and let $V_S = (\bigcup_{i \in S} V_i)$. Furthermore, let A_S be the set of all arcs between the vertices in V_S and c_{ij} represent the cost of using the arc (i, j) with $i, j \in V_S$. Now define the following network $W_S = (v_d, V_S \cup v_d, A, c, d, Q)$. Using this network the characteristic function is defined as the minimum cost to service all customers of the players in coalition S. Hence $C(S) = VRP(W_S)$. From now on, the joint network vehicle routing game is denoted as $\langle N, C \rangle$. Next, an illustrative instance of the joint network vehicle routing game is provided.

Example 4.1. *Consider the instance of the joint network vehicle routing game as displayed in Figure 4.1. Note that the arcs between all locations have been omitted for the sake of clarity. The example consists of three players A, B and C. Rather than each player servicing their own locations, the locations are now serviced as if they all belong to one logistic service provider. The goal of the game is to assign each player a share of the total costs without giving any player the incentive to leave the grand-coalition.*

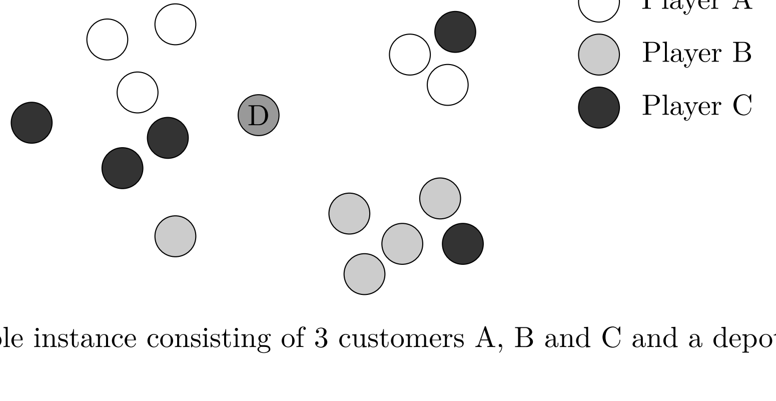


Figure 4.1: An example instance consisting of 3 customers A, B and C and a depot D.

4.2 Similar games

In this section an overview of games similar to the joint network vehicle routing game is presented along with the most important properties of these games. Besides this, it is proven that the joint vehicle routing game is a generalisation of both the travelling salesman game and the vehicle routing game. Later on, this result is used to determine some of the properties of the joint network vehicle routing game.

4.2.1 The travelling salesman game

A travelling salesman game consists of network W and a grand-coalition N. For each coalition S the characteristic function $C(S)$ is defined as the minimum cost cycle in W that starts and ends at the home town, or depot, while visiting all players in S. Now the travelling salesman game is defined as $\langle N, C\rangle$, with the goal to find a distribution of the costs among the members of S.

One of the earliest mentions of the travelling salesman game are found in a paper by Fishburn and Pollak [17]. They designed the game to distribute costs among host institutions on an academic trip throughout the country. Nevertheless, the corresponding travelling salesman game was formally introduced by Potters et al. [39]. In their paper they mathematically formalise the game and propose a class of travelling salesman games with non-empty core.

However, not every travelling salesman game has a non-empty core. This assertion is proven by Tamir [47]. The author proves that the core of the travelling salesman game is always non-empty if the game consists of four or less players. Further research by Kuipers [28] shows that the core is guaranteed to be non-empty for five or less players. Furthermore, Tamir [47] shows that for travelling salesman games consisting of 6 or more players, a non-empty core can no longer be guaranteed. This result is summarised in the following theorem.

Theorem 4.1. *The core of a travelling salesman game may be empty for 6 or more players.*

Proof. The proof can be found in [47]. □

Now, it is proven that the travelling salesman game is embedded in the joint network vehicle routing game. This result can be used to show that properties of the travelling salesman game also hold for the joint network vehicle routing game.

Theorem 4.2. *Each travelling salesman game can be modelled as a joint network vehicle routing game.*

Proof. Consider an instance of the travelling salesman game with grand-coalition N and underlying network W. The characteristic function of a coalition S is defined as the minimum cost cycle that visits all players in S, denote this cost as $C(S)$.

Now, consider a joint network vehicle routing game $\langle N, D\rangle$ with N the grand-coalition, D the characteristic function. Furthermore, let $d_v = 1$ for all locations and let the maximum vehicle capacity be $Q = \sum_{v\in V_S} d_v = |N|$. The characteristic function is defined as the minimum cost to service all players in a coalition S. As such, $D(S) = VRP(W_S)$. Note that an optimal solution to the corresponding travelling salesman problem is also an optimal solution to the corresponding vehicle routing problem. This statement holds as both problems are solved using the same network and no negative fixed costs for using additional vehicles are utilised. From this it follows that the characteristic functions are equal, hence $C(S) = D(S)$ and the allocation problems are the same as well. Now it can be concluded that each travelling salesman game can be modelled as a joint network vehicle routing game. □

4.2.2 The vehicle routing game

In the vehicle routing game, a grand-coalition N and underlying network W are considered. In this game each player has exactly one location. The characteristic function $C(S)$ is defined as minimum costs to visits all players in S. To that end, multiple routes can be utilised, each with a maximum capacity of Q.

First, a proof is presented that the vehicle routing game is essential under the following assumptions:

Theorem 4.3. *A vehicle routing game is essential if there exists a pair of locations i and j such that the following statements hold:*

1. $c_{i,j} < c_{i,k} + c_{k,j}\ \forall k \in N\backslash\{i,j\}$
2. $d_i + d_j \leq Q$

Proof. In this proof a feasible route is constructed for the vehicle routing problem of the grand-coalition. It is shown that with this feasible route the total costs of serving the grand-coalition is always smaller than the sum of the stand-alone costs. Note that all players only have one location. As such $|V_i| = 1$ for all $i \in \mathbb{N}$. For all customers i in $V_N\backslash\{v_d\}$ the optimal stand-alone route is denoted by σ_i. The costs of this route are equal to $c_{v_d,i} + c_{i,v_d}$. Now suppose a pair $i,j \in V_N$ exist such that $c_{ij} < c_{ik} + c_{kj}\ \forall k \in V_S\backslash\{i,j\}$. Then it holds that

$$C(\{i,j\}) = c_{v_d,i} + c_{ij} + c_{j,v_d} < c_{v_d,i} + c_{i,v_d} + c_{v_d,j} + c_{j,v_d} = C(\{i\}) + C(\{j\}).$$

Furthermore, as $d_i + d_j \leq Q$ the route is feasible. Now it follows that

$$C(N) \leq C(\{i,j\}) + \sum_{k \in C\backslash\{i,j\}} C(\{k\}) < \sum_{k \in C} C(\{k\}).$$

Using Definition 2.6 it can now be concluded that the game is essential. □

The first paper to discuss cost allocation in the vehicle routing game is Göthe-Lundgren et al. [25]. In their paper it is shown that the core of the game is non-empty under certain circumstances. The resulting theorem is formalised in the following.

Theorem 4.4. *The core of a vehicle routing is game is non-empty if and only if the $\underline{VRP(W_N)} = \overline{VRP(W_N)}$.*

Proof. The proof can be found in [25]. □

Next, a proof that the core can actually be empty is provided. To that end, an instance of the vehicle routing game with an empty core is considered.

Theorem 4.5. *A vehicle routing game can have an empty-core.*

Proof. This proof uses an example from Göthe-Lundgren et al. [25]. Consider the completely symmetric instance of the game as presented in Figure 4.2. This instance will serve as an example to show that the core of a joint network vehicle routing game can be empty. The instance consists of 3 players each having a demand of 1. The players are serviced from the centralised depot using trucks with a capacity of 2.

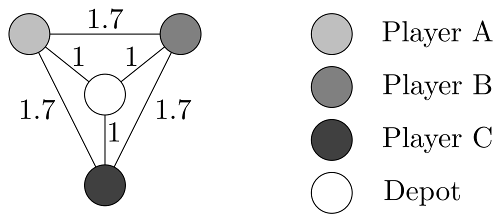


Figure 4.2: An instance consisting of 3 customers A, B and C and a centralised depot. Each customer has a demand of 1.

The optimal costs for each coalition are given in Table 4.1. Note that for the grand-coalition a total of two trucks have to be used.

Table 4.1: The optimal costs for each coalition in the example.

Coalition	Cost
A	2
B	2
C	2

Coalition	Cost
A,B	3.7
B,C	3.7
C,A	3.7

Coalition	Cost
A,B,C	5.7

Now, for any solution in the core the following equations must be satisfied:

$$x_A + x_B \leq 3.7 \tag{4.1}$$
$$x_B + x_C \leq 3.7 \tag{4.2}$$
$$x_C + x_A \leq 3.7 \tag{4.3}$$
$$x_A + x_B + x_C = 5.7 \tag{4.4}$$

Adding equations (4.1), (4.2) and (4.3) yields $2(x_A + x_B + x_C) \leq 11.1$. Dividing this equation by 2 gives $x_A + x_B + x_C \leq 5.55$ which implies that no solution in the core exists. Using the contra-position of Theorem 2.2 it can be concluded that the core of a vehicle routing game can be empty. □

Note that the vehicle routing game is actually quite similar to the newly defined joint network vehicle routing game. Finally, it is shown that an instance of the vehicle routing game is also an instance of the joint network vehicle routing game.

Theorem 4.6. *Each vehicle routing game is a joint network vehicle routing game.*

Proof. Each vehicle routing game can be modelled as a joint network vehicle routing game with the same players. In this case each player in the joint network vehicle routing game only has one location. As such, the vehicle routing game is a special case of the joint network vehicle routing game. □

4.3 Properties of the game

In this section, properties of the newly defined joint network vehicle routing game are derived. First, it is shown that the game is monotonic.

Theorem 4.7. *The joint network vehicle routing game $\langle N, C\rangle$ is monotonic if the distances in the corresponding network W_N adhere to the triangle inequality.*

Proof. Let $S, T \subseteq N$ such that $S \subseteq T$. Define σ as a set of routes with optimal costs $C(T)$ that serves the customers in T. Now, denote σ' as the route that visits the locations in S in the order of σ. Due to the triangle inequality it holds that the costs of this route have to be equal or lower than the costs of σ as some nodes have been taken out of the route. Furthermore, as $S \subseteq T$ it also holds that σ' serves exactly the customers in S. Since $C(S)$ is the minimal cost to serve all customers in S it follows that $C(T)$ is an upper-bound to $C(S)$. Hence, $C(S) \leq C(T)$. □

A desirable characteristic of the game is the sub-additivity property. With this property it is guaranteed that cooperating is beneficial, or in the worst-case scenario does not increase the total costs. The proof that the game is sub-additive is presented below.

Theorem 4.8. *The joint network vehicle routing game $\langle N, C \rangle$ is sub-additive.*

Proof. Let $S, T \subseteq N$ such that $S \cap T = \emptyset$. Define σ_T as a set of routes with optimal costs $C(T)$ that serves the customers in T and define σ_S as a set of routes with optimal costs $C(S)$ that serves the customers in S. Now, a feasible set of routes to serve the customers in $S \cup T$ is $\sigma_S \circ \sigma_T$ with cost $C(S) + C(T)$. Here the $\circ$ indicates that set of routes σ_S is carried out after the set of routes σ_T. This means that $C(S \cup T) \leq C(S) + C(T)$ as the costs ought to be minimised. □

It is key that a game is essential. If this is not the case, the players would be better off by not cooperating at all. It is shown that the game is essential under one easily verifiable assumption.

Theorem 4.9. *The game $\langle N, C \rangle$ is essential if there exist players $i, j \in N$ such that*

$$VRP(W_{\{i\}}) + VRP(W_{\{j\}}) > VRP(W_{\{i,j\}}).$$

Proof. The result follows from the following observation:

$$C(N) \leq VRP(W_{\{i,j\}}) + \sum_{k \in N \setminus \{i,j\}} VRP(W_{\{k\}}) < \sum_{k \in N} VRP(W_{\{k\}}) = \sum_{k \in N} C(\{k\}).$$

□

Another very desirable characteristic is the concavity of the game. Concave games have non-empty cores and as such can ensure that the grand-coalition is stable. Unfortunately, it can be proven that even a simplified version of the game is non-concave as the core of a joint network vehicle routing game can be empty.

Theorem 4.10. *The core of a joint network vehicle routing game $\langle N, c \rangle$ can be empty.*

Proof. Consider an instance of the vehicle routing game such that the core is empty. The fact that such an instance exists was proven in Theorem 4.5. According to Theorem 4.6 this instance is also an instance of the joined network vehicle routing game. As such, it can be concluded that instances of the joint network vehicle routing game for which the core is empty exist. □

5 Solution Procedures

In this section different methodologies to solve the joint network vehicle routing game are introduced. First, a solution procedure for the capacitated vehicle routing problem is proposed in Section 5.1. Next, in Section 5.2, several solution procedures for the joint network vehicle routing game are proposed. The solution procedure utilises a branch-and-price algorithm for the vehicle routing problem and incorporates the branch-and-price approach in a variety of algorithms to efficiently determine the total costs for each coalition.

5.1 Vehicle routing problem

To allocate costs among players or to determine whether an allocation is in the core, the vehicle routing problem may have to be solved an exponential amount of times with respect to the amount of players. As such, it is key that an efficient solution procedure is used to limit the required computation time. In Section 3.3 an overview of some of the different solution methods is presented. The aim of this section is to clearly illustrate the methods used to solve an instance of the vehicle routing problem in this thesis.

First, two often used formulations of the vehicle routing problem are presented. In Section 5.1.1 the flow formulation for the vehicle routing problem is presented. Next, in Section 5.1.2 the set partitioning formulation is presented along with an approach to solve this formulation. A detailed overview of the solution algorithm for the vehicle routing problem as used in this problem is provided in Section 5.1.3.

5.1.1 Flow formulation

In this subsection the vehicle routing problem is modelled as a flow problem. For this the formulation provided in Golden et al. [24] is adapted to the capacitated vehicle routing problem. The binary decision variable x_{ij} denotes whether an arc is used in the optimal solution. Furthermore, the decision variable y_{ij} is used to keep track of the demand used on each of the routes in the solution. This variable can then be utilised to enforce the maximum vehicle capacity.

Flow formulation

Objective:
$$VRP(W_S) = \min \sum_{i,j \in A} c_{ij} x_{ij} \tag{5.1}$$

Subject to:
$$\sum_{i \in V_S \cup \{v_d\}} x_{ij} = 1 \quad \forall j \in V_S \tag{5.2}$$
$$\sum_{i \in V_S \cup \{v_d\}} x_{ik} - \sum_{j \in V_S \cup \{v_d\}} x_{kj} = 0 \quad \forall k \in V_S \tag{5.3}$$
$$\sum_{i \in V_S \cup \{v_d\}} y_{ij} - \sum_{i \in V_S \cup \{v_d\}} y_{ji} = d_j \quad \forall j \in V_S \tag{5.4}$$
$$d_j x_{ij} \leq y_{ij} \leq (Q - d_i) x_{ij} \quad \forall i,j \in V_S \cup \{v_d\} : i \neq j \tag{5.5}$$
$$y_{ij} \geq 0 \quad \forall i,j \in V_S \cup \{v_d\} : i \neq j \tag{5.6}$$
$$x_{ij} \in \{0,1\} \forall i,j \in V_S \cup \{v_d\} : i \neq j \tag{5.7}$$

Constraints (5.2) ensure that each location is visited exactly once as required. Constraints (5.3) ensure that if a location has an incoming arc it must also have an outgoing arc. This ensures that each path can only end and start at the depot. Constraints (5.4) are also known as the commodity flow constraints. These constraints ensure that the difference in flow before and after a location equals the demand of that location. As such, these constraints help to keep track of the total demand supplied by a route. Constraints (5.5) ensure that the maximum vehicle capacity is not exceeded by any of the routes. Constraints (5.6) ensure that the flow of goods is positive and finally constraints (5.7) ensure that each arc is used once or not at all.

This formulation of the vehicle routing problem can be solved using a mixed integer problem solver such as CPLEX [1]. Most of the solvers relax the flow formulation and implement the problem in a branch-and-bound scheme. However, the bounds provided by the flow formulation are known to be not very tight. As such, another tighter formulation is presented in the form of the set partitioning formulation.

5.1.2 Set partitioning formulation

In this subsection the vehicle routing problem is modelled using the set partitioning formulation. In order to solve this formulation all feasible routes ought to be known. As the process of listing all routes can be quite computationally intensive, a solution strategy to circumvent this issue is presented. For this, the master problem is introduced and discussed in Subsection 5.1.2.1. In the master problem, each route is allowed to be used a non-integer amount of times. As such, this problem is the LP-relaxation of the set partitioning problem. The columns of the master problem are represented by the set of all feasible routes. Unfortunately, this set can be exponentially large in the amount of vertices used in the problem. In order to circumvent the issue of listing all routes, the restricted master problem is introduced. The restricted master problem is thoroughly discussed in Subsection 5.1.2.2. This problem no longer considers all routes and is initialised with a small set of routes such that a feasible solution exists. Any other feasible routes that could potentially lower the costs are generated by the so called pricing problem. Next, the new routes are added to the restricted master problem, which in turn is solved again. This process is repeated until no more beneficial routes are generated by the pricing problem. A detailed description of this problem and an algorithm to solve the problem is given in Subsection 5.1.2.3. Using this method the (restricted) master problem can be solved to optimality. However, the optimal solution to the master problem is not necessarily integral. As such, a branch-and-bound procedure is proposed in Subsection 5.1.2.4. Through this procedure an integral solution to the original master problem is determined.

First, the set partitioning formulation of the vehicle routing problem is presented. Consider a joint network vehicle routing game with grand-coalition N and network W. In the following the vehicle routing game with network W_S, belonging to coalition S, is considered. The goal is to determine the optimal costs $VRP(W_S)$ along with the optimal routes used to attain this cost. Next, the vehicle routing problem is modelled as a set partitioning problem.

To this end some notation is introduced. The set of all feasible routes in the vehicle routing problem is denoted as K, note that the cardinality of K can be exponential in the amount of nodes in the network. Furthermore, a_{vk} denotes whether a location v is on route $k \in K$. It holds that $a_{vk} = 1$ if location v is on route k and zero otherwise. Now the set partitioning formulation for the vehicle routing problem can be stated as follows:

Set partitioning formulation

$$\textbf{Objective:} \quad VRP(W_S) = \min \sum_{k \in K} c_k x_k \tag{5.8}$$

$$\textbf{Subject to:} \quad \sum_{k \in K} a_{vk} x_k = 1 \quad \forall v \in V_S \tag{5.9}$$

$$x_k \in \{0,1\} \quad \forall k \in K \tag{5.10}$$

Constraints (5.9) ensure that each customer is visited exactly once as desired. Constraints (5.10) ensure that each route is used exactly once or not at all in the final solution.

5.1.2.1 Master problem

In the following, the master problem corresponding to the set partitioning formulation of the vehicle routing problem is presented. This formulation relaxes constraints (5.10). The formulation is presented below:

Master problem

$$\textbf{Objective:} \quad \overline{VRP(W_S)} = \min \sum_{k \in K} c_k x_k \tag{5.11}$$

$$\textbf{Subject to:} \quad \sum_{k \in K} a_{vk} x_k = 1 \quad \forall v \in V_S \tag{5.12}$$

$$x_k \in \mathbb{R}^+ \quad \forall k \in K \tag{5.13}$$

Constraints (5.12) ensure that each customer is visited exactly once as desired. Constraints (5.13) ensure that each route is used a non-negative amount of times in the final solution. For smaller instances one might be able to explicitly list the set K. For larger instances this process can become very resource intensive. To alleviate this issue the restricted master problem is introduced.

5.1.2.2 Restricted master problem

The restricted master problem restricts the set of columns, feasible routes, to be used in the final solution. The restricted master problem is initialised with a small set of all feasible routes. This smaller set is denoted as K' and can be initialised in many different ways. The possible ways of initialising K' are discussed later. For now, it is important to know that K' has to be initialised such that a feasible solution to the restricted master problem exists. Besides this, any other, potentially profitable, routes have to be generated. A more thorough explanation on how these routes are generated follows later. First, the restricted problem master is formulated:

Restricted master problem

$$\textbf{Objective:} \quad \min \sum_{k \in K'} c_k x_k \tag{5.14}$$

$$\textbf{Subject to:} \quad \sum_{k \in K'} a_{vk} x_k = 1 \quad \forall v \in V_S \tag{5.15}$$

$$x_k \in \mathbb{R}^+ \quad \forall k \in K' \tag{5.16}$$

Constraints (5.15) ensure that each customer is visited exactly once. Constraints (5.16) ensure that each route is visited a positive amount of times. Note that this time a route can be visited a non-integer amount of times. As such the minimum costs for the restricted master problem provides an upper bound to the minimum costs of the master problem. As the set K' does not contain all routes, some routes need to be generated. This is done using a so called pricing problem.

Initialisation of the routes

The manner in which the set K' is initiated can affect the performance of the algorithm. If the restricted master problem is initiated with a bad feasible solution, more iterations of the pricing problem may be required to generate the optimal routes. Here, two different ways to initialise the set of routes are presented.

In the first approach the set of routes is initialised with the so called trivial routes. A trivial route is a route that travels from the depot to exactly one customer and then back to the depot. As there is no limit on the amount of vehicles used, a feasible solution can be created with this set of routes. Unfortunately, this set of routes yields the worst possible feasible solution to the problem, should the triangle equality hold. As such, a second procedure to initialise the routes is presented. This approach is based on the savings algorithm as presented by Clarke and Wright [8]. In the following the procedures of the savings algorithm are elaborated upon. First, define the savings of an arc as:

$$s(i,j) = c_{v_d i} + c_{j v_d} - c_{ij} \tag{5.17}$$

Using these savings, the algorithm heuristically decides which node should be preferably combined in a route. The algorithm starts by creating the trivial routes. Next, the algorithm calculates all savings and sorts these savings in decreasing order. Then, the algorithm chooses the highest savings $s(i,j)$ and tries to merge the route ending at i with the route starting with j if this leads to a feasible route. This process is repeated until all nodes have been considered and no more feasible routes can be produced. The savings algorithm can be summarised as follows:

Algorithm 1 Clarke and Wright parallel savings algorithm

```
Initialisation
 1: Calculate the savings $s(i,j)$ for all pairs of vertices $(i,j) \in V \backslash v_d$.
 2: Let $L$ be the ordered list of savings in descending order.

 3: Set $K'$ as the set of trivial routes.
Procedure
 4: while $|L| > 0$ do
 5:     Let $l = (i,j)$ be the first element of $L$.
 6:     if the routes starting at $i$ and ending at $j$ exist and the merger is feasible then
 7:         Merge the routes in $K'$.
 8:     Remove $l$ from $L$.
 9: end
```

More advanced construction heuristics than the one presented here exist. An example of such an heuristic is given by Pisinger and Ropke [38].

5.1.2.3 Pricing problem

The goal of the pricing problem is to find routes with negative reduced costs and add these routes to the set K' used in the restricted master problem. If no route with negative reduced costs can be found, the (restricted) master problem has been solved to optimality. In order to efficiently determine a set of routes with negative reduced costs, the pricing problem is formulated as an elementary shortest path problem with resource constraints.

Let u_v denote the optimal values of the dual variables corresponding to constraints (5.15) after solving the restricted master problem with the feasible routes in K'. Using these optimal dual values, the reduced costs can be defined. For a route k it holds that the reduced costs, corresponding to the restricted master problem with set K', are equal to $c_k - \sum_{v \in V_S} u_v a_{vk}$. The goal is to find feasible routes such that these reduced costs are minimised. Such a problem is also known as an elementary shortest path problem with resource constraints. A possible formulation of the problem is as follows:

Elementary shortest path problem with resource constraints

Objective:
$$\min \sum_{(i,j) \in (V_S \cup \{v_d\} \times V_S \cup \{v_d\})} r_{ij} y_{ij} \tag{5.18}$$

Subject to:
$$\sum_{j \in (V_S \cup \{v_d\})} [y_{ij} - y_{ji}] = 0 \quad \forall i \in (V_S \cup \{v_d\}) \tag{5.19}$$
$$\sum_{j \in V_S} y_{v_d,j} \geq 1 \tag{5.20}$$

$$q_i - q_j + d_j \leq M(1 - y_{ij}) \quad \forall i, j \in V_S \tag{5.21}$$
$$0 \leq q_i \leq Q \quad \forall i \in V_S \tag{5.22}$$

$$y_{ij} \in \{0, 1\} \quad \forall (i, j) \in (V_S \cup \{v_d\} \times V_S \cup \{v_d\}) \tag{5.23}$$
$$q_i \in \mathbb{R}^+ \quad \forall i \in V_S \tag{5.24}$$

where M is a natural number such that $M > 2Q$ and r_{ij} denotes the reduced costs of an arc. This cost is defined as $r_{ij} = c_{ij} - u_j$ where $u_{v_d} = 0$. Constraints (5.19) enforce that the amount of arcs incoming is equal to the amount of arcs outgoing for each customer. Constraint (5.20) ensures that each route starts from the depot. Constraints (5.21), (5.22) and (5.24) are an adaptation of the MTZ-subtour elimination constraints as described by Miller et al. [32] to include the maximum capacity. Furthermore, constraints (5.23) ensure that each arc is used at most once or not at all.

Feillet et al. [15] propose an efficient algorithm to solve the elementary shortest path problem with resource constraints. In the following their improved label correcting algorithm to solve the elementary shortest path problem with resource constraints is presented.

The ESPPRC algorithm

The ESPPRC algorithm determines the shortest path from an origin node s to a destination node e in a graph with n nodes in an efficient manner. In the worst case scenario an exponentially large amount of paths may have to be considered. In order to distinguish each of these paths, a labelling is introduced. The idea behind the label setting algorithm is to first create a label at the origin. This label is extended towards each successor of the origin node. In this manner new paths are created. However, some extensions may not be possible. To account for this, the principle of unreachable nodes is introduced.

Definition 5.1. *A node is said to be unreachable for a (partial) route p if adding the node to the (partial) route causes a violation of one of the following statements:*

1. *Each node can only be used once in a route*
2. *The total demand of the route should not exceed the capacity.*

Furthermore, not all paths need to be extended. This means that some labels can be removed early on. Hence, limiting the amount of labels that need to be considered. This procedure is governed by a dominance rule. Before this rule is stated, a clear definition of the labels used in the algorithm is given. Throughout the algorithm each path ending at a node i is assigned a label (λ_i, C_i). Here $\lambda_i = (q_i, s_i, U_i^1, \ldots, U_i^n)$, where q_i indicates the vehicle capacity which has already been used, s_i denotes the total amount of unreachable nodes and U_i denotes the vector of unreachable nodes, a one indicating that a node is unreachable and a zero indicating that a node can still be reached. The second part of label, C_i, indicates the costs of the partial route starting at the source v_d to a node i in the graph.

In order to reduce the computational effort required, not all paths are considered. To that end Feillet et al. [15] efficiently determine the non-dominated paths. They use the following dominance theorem to determine if a path is non-dominated.

Theorem 5.1 (Dominance)**.** *Let p_1 and p_2 be paths from the origin node s to a node i. Both paths have respective associated states $\lambda_i^{p_1} = (q_i, s_i, U_i^1, \ldots, U_i^n)$ with costs C_1 and $\lambda_i^{p_2} = (q_i', s_i', U_i'^1, \ldots, U_i'^n)$ with costs C_2. Path p_1 dominates path p_2 if and only if $C_1 \leq C_2, s_i \leq s_i', q_i \leq q_i'$ and $U_i^k \leq U_i'^k$ for all nodes k with $k \in \{1, \ldots, n\}$ and the paths are not equal.*

Let Λ_i denote the set of labels associated with node i. The function Extend(λ_i, j) is used to extend a label λ_i to a node j. If the extension is possible the function returns an updated label. If the extension is not possible the function returns nothing. In the case that node j is reachable, the function first updates the resource constraints. Thereafter the function updates the amount of and list of unreachable nodes. Besides this, the function also updates the total costs of the new label. Now the following algorithm as described by Feillet et al. [15] is executed to determine the shortest path.

Algorithm 2 ESPPRC label setting algorithm (Feillet et al. [15])

```
Initialisation
 1: Set $\Lambda_{v_d} = \{(0, 0, 0, \ldots, 0, 0)\}$.
 2: for $i \in V_S$ do
 3:     set $\Lambda_i = \emptyset$.
 4: end
 5: E = $\{v_d\}$.

Procedure
 6: repeat
 7:     Choose $i \in E$
 8:     for $j$ which is a successor of $i$ do
 9:         Set $F_{i,j} = \emptyset$ .
10:         for $\lambda_i \in \Lambda_i$ do
11:             if $U_i^j = 0$ then
12:                 $F_{i,j} = F_{i,j} \cup \{\text{Extend}(\lambda_i, j)\}$.
13:         end
14:         $\Lambda_j = \Lambda_j \cup F_{i,j}$
15:         Remove all non-dominant paths from $\Lambda_j$.
16:         if $\Lambda_j$ changed then
17:             $E = E \cup \{j\}$.
18:     end
19:     $E = E \backslash \{i\}$.
20: until $E = \emptyset$
```

An illustrative example of the algorithm is provided in Appendix B. The elementary shortest path problem with resource constraints is known to be an NP-hard problem [12] and cannot be solved in polynomial time. The time-complexity of the algorithm depends on the structure of the graph. For highly constrained graphs, relatively large pricing problems can be solved as less labels have to be created.

The algorithm as presented by Feillet et al. [15] may require some labels to be extended multiple times to the same successor as all non-dominated labels of a node are considered each time the node is considered for extensions within the algorithm. Range [40] proposes an algorithm to iterate more efficiently over the labels. To this end, the author introduces the set Λ_i^k which contains all non-dominated labels of partial paths ending at node i with a total of k unreachable nodes. Besides this, the author introduces the set E_k, denoting the set of nodes with a label of length k that ought to be considered during the iterations. Next, the author proposes the following algorithm.

Algorithm 3 ESPPRC label setting algorithm (Range [40])

```
Initialisation
 1: Set $\Lambda_{v_d} = \{(0, 0, 0, \dots, 0, 0)\}$.
 2: for $i \in V_S$ do
 3:     set $\Lambda_i = \emptyset$.
 4: end
 5: $\mathrm{E}_0 = \{v_d\}$.

Procedure
 6: for $k = 0$ to $k = |V_S|$ do
 7:     while $E_k \neq \emptyset$ do
 8:         Choose $i \in E_k$.
 9:         for $\lambda \in \Lambda_i^k$ do
10:             for $j$ which is a successor of $i$ do
11:                 if $\lambda$ can be extended towards $j$ then
12:                     Set $\lambda' = \mathrm{Extend}(\lambda, j)$.
13:                     if $\lambda'$ is not dominated by $\Lambda_j$ then
14:                         Remove all labels from $\Lambda_j$ that are dominated by $\lambda'$.
15:                         Add $\lambda'$ to $\Lambda_j$.
16:                         Set $u$ as the amount of unreachable nodes of label $\lambda'$.
17:                         $E_u = E_u \cup \{j\}$.
18:                     end
19:                 end
20:             end
21:         end
22:         $E_k = E_k \backslash \{i\}$.
23:     end
24: end
```

Using either of these algorithms the pricing problem can be solved exactly. Each time the pricing problem is solved, the algorithm yields an overview of all non-dominated, feasible routes to the problem. From this set of feasible routes a certain subset, can be added to K'. After this, the restricted master problem and pricing problem, if required, are solved again.

Heuristic adjustment

Solving the elementary shortest path problem with resource constraints may prove to be a resource intensive process in determining the optimal solution to the master problem. Because the pricing problem can be executed multiple times, the algorithm may negatively influence the overall computational performance. To this end, it might be useful to utilise a heuristic approach to the pricing problem. Remember that the sole purpose of the pricing problem is to find columns with negative reduced cost. As such, it is not necessary to determine all columns with negative costs, let alone determine the column with the most negative reduced costs. This means that the pricing problem can be solved heuristically to determine any profitable columns. However, if the heuristic pricing problem cannot generate a route with negative reduced cost, the pricing problem ought to be solved exactly to ensure that no profitable columns exist. By introducing a heuristic procedure for the pricing problem, the times that the pricing has to be solved to optimality can be reduced.

In the following, a heuristic adjustment as introduced by Chabrier [7] is made to both ESPPRC algorithms which have been presented in the foregoing. The main idea behind the adjustment is to make the dominance rule more strict. As such, less labels persist throughout the algorithm, greatly improving the efficiency of the algorithm. The only downside is that the heuristic dominance rule potentially dominates labels which turn out to be profitable. The new, relaxed dominance is defined as follows.

Theorem 5.2 (Relaxed dominance)**.** *Let p_1 and p_2 be paths from the origin node s to a node i. Both paths have respective associated states $\lambda_i^{p_1} = (q_i, s_i, U_i^1, \ldots, U_i^n)$ with costs C_1 and $\lambda_i^{p_2} = (q_i', s_i', U_i'^1, \ldots, U_i'^n)$ with costs C_2. Path p_1 dominates path p_2 if and only if $C_1 \leq C_2$, $q_i \leq q_i'$ and the paths are not equal.*

Both algorithms for the elementary shortest path problem with resource constraint can now be executed with the relaxed dominance theorem to find columns with negative reduced cost.

5.1.2.4 Branch-and-bound

In a branch-and-bound approach one generally branches on a decision variable. In this case that would create two different problems; one in which a route is enforced and one in which a route is forbidden to be used in the solution. Enforcing a route may not seem problematic at all, the decision variable can be used to account for this. However, enforcing that a route is not used is more difficult. In this case, the pricing problem would be no longer allowed to generate certain routes. This cannot easily be implemented in the elementary shortest path algorithm. As such, a different branching rule as presented by Feillet [14] is utilised.

Whether an arc (i, j) is used in route k is denoted by b_{ijk}. Now, the times an arc is used in a solution can be expressed as $f_{ij} = \sum_{k \in K'} b_{ijk} x_k$. The new branching rule uses the fact that in each integer solution, an arc, rather than a route, is used exactly once or not at all. In the restricted master problem it can occur that an arc is used a fractional amount of times. This leads to the following branching rule:

- Select an arc (i, j) such that $0 < f_{ij} < 1$.
- Create two branches such that:
    - the use of arc (i, j) is enforced.
    - the use of arc (i, j) is forbidden.

Enforcing an arc (i, j) in the problem is quite easy. One can simply remove all arcs $(l, j) : l \neq i$ and $(i, l) : l \neq j$ from the network W_s. In this manner at least one route in the solution must use the arc (i, j) as both locations have to be visited. All columns in the master problem that utilise one of the deleted arcs have to be removed from the problem. To ensure that an arc (i, j) is not used, the arc can simply be removed from the network W_S. Besides this, all columns in the master problem that utilise this arc have to be removed.

5.1.3 Solution algorithm

In this section, a method to solve the vehicle routing problem corresponding to a coalition S is proposed. The aim of this section is to link the different sections and illustrate some design choices that had to be made in the solution algorithm.

Due to the fact that the flow formulation does not provide tight bounds, the set partitioning formulation is utilised in this thesis. The set partitioning formulation is known for its good bounds and as such is expected to perform well in a branch-and-bound scheme. As such, the previously proposed scheme, also known as a branch-and-price procedure, will be utilised to solve the vehicle routing problem to optimality. However, in the branch-and-price procedure some design choices still have to be made. Most of these choices are motivated by a combination of theoretical insights and preliminary results. These results have been generated using a random set of instances. The characteristics of the instances along with the results of the different comparisons can be found in Appendix D.

The initialisation

First, the initialisation of the branch-and-price procedure is discussed. In Subsection 5.1.2.2 two different methods to initialise the set of routes K' have been proposed. One of the proposed methods utilised the trivial routes, whereas the second proposed method efficiently generates routes by merging the aforementioned trivial routes. As both methods are very simplistic, the computational time required for both methods is expected to be relatively small. The only somewhat intensive operation is the sorting of all savings values in the second method. In order to observe the impact of the initialisations on the solution a comparison of the computational time required with both initialisations has been performed. The results of this comparison can be found in Appendix D.1. From this comparison it follows that the savings construction heuristic negatively impacts the computation time and as such is not used in the branch-and-price procedure.

The pricing problem

The pricing problem has to be solved in each node at least once. This means that the impact of the pricing problem on the solution algorithm can be quite severe. As such, it is important to optimise the procedure of finding any profitable columns. To this end multiple procedures exist. Two algorithms by Feillet et al. [15] and Range [40] have been introduced in Subsection 5.1.2.3. Both algorithms yield the same results and as such only have to be compared on computational performance. The preliminary results are displayed in D.2 and suggest the use of the algorithm as presented by Range [40].

After solving the elementary shortest path problem with resource constraints using one of the label setting algorithms, the column with most negative reduced costs is known. However, even after adding this column to the restricted master problem it is not guaranteed that the column is used in the optimal solution of the master problem at all. As such, it might be beneficial to add multiple columns at once up to a certain limit. In the pricing problem, the first 100 generated profitable labels ending at the depot are added to the restricted master problem. This ensures that the elementary shortest path problem with resource constraints do not have to be solved to optimality most of the times. The preliminary results for varying number of columns added at once can be found in Appendix D.1.

As the pricing problem does not necessarily have to be solved to optimality, an heuristic adjustment was proposed. This adjustment introduced a relaxed dominance theorem to heuristically solve the elementary shortest problem with resource constraints. This problem can be executed rather than the exact algorithms until no solution can be found using the heuristic. In this case, the exact algorithm is executed. A comparison has been included in D.2 to indicate the performance gains of using this heuristic. As such, this heuristic is also applied in the final procedure.

Besides these choices, it should also be noted that computational errors persist throughout the execution of the algorithm. These numerical errors may be small at first but a build up of errors can occur in larger instances. As such, a numerical margin ε is utilised in deciding whether a column is deemed profitable. This margin is taken as $\varepsilon = 10^{-6}$ meaning that only columns with a reduced cost of less than $-\varepsilon$ are accepted as profitable columns. This means that despite the theoretical exactness of the algorithm, errors may occur due to the numerical precision and slightly profitable columns may not be generated by the pricing problem. This effect is not expected to have any influence on the results for this thesis.

The branch-and-bound procedure

Now that the procedures to solve each node have been discussed, the only thing that remains is to decide on a strategy for the branch-and-bound procedure. First, a decision must be made regarding the arc to branch on. Two different possibilities are considered. In the first case, the first fractionally used arc encountered is branched on. In the second case, the arc which is most fractionally used is selected to branch on. This is the arc of which the usage in the solution to the relaxed problem is closest to $\frac{1}{2}$. However, for this procedure the usage of all arcs needs to be calculated. As there are no clear theoretical motivations for one or the other approach, the choice is solely based on the preliminary results. In these results it was found that using the first arc found to branch on resulted in a significant amount of additional nodes during the branching procedure. So much nodes that the testing rig often ran out of RAM. As such, the procedure branches on the arc that was used closest to $\frac{1}{2}$ times in the solution.

Finally, a node strategy has to be determined. After branching, two new nodes are created an added to the tree. Two strategies of evaluating nodes are considered, the last in first out (depth-first) strategy and the first in first out (width-first) strategy. The depth-first strategy tries to get fast as possible as deep as possible into tree trying to find a better bound. In contrary, the width first strategy, first solves the top layers of the tree. To determine which strategy to use, a comparison is provided in Appendix D.3. From these results it follows that first in first out strategy is favourable.

An overview

In the following the entire branch-and-price procedure as described in the previous subsections is summarised. To this purpose Figure 5.1 described the actions taken to obtain the optimal solution to the vehicle routing problem.

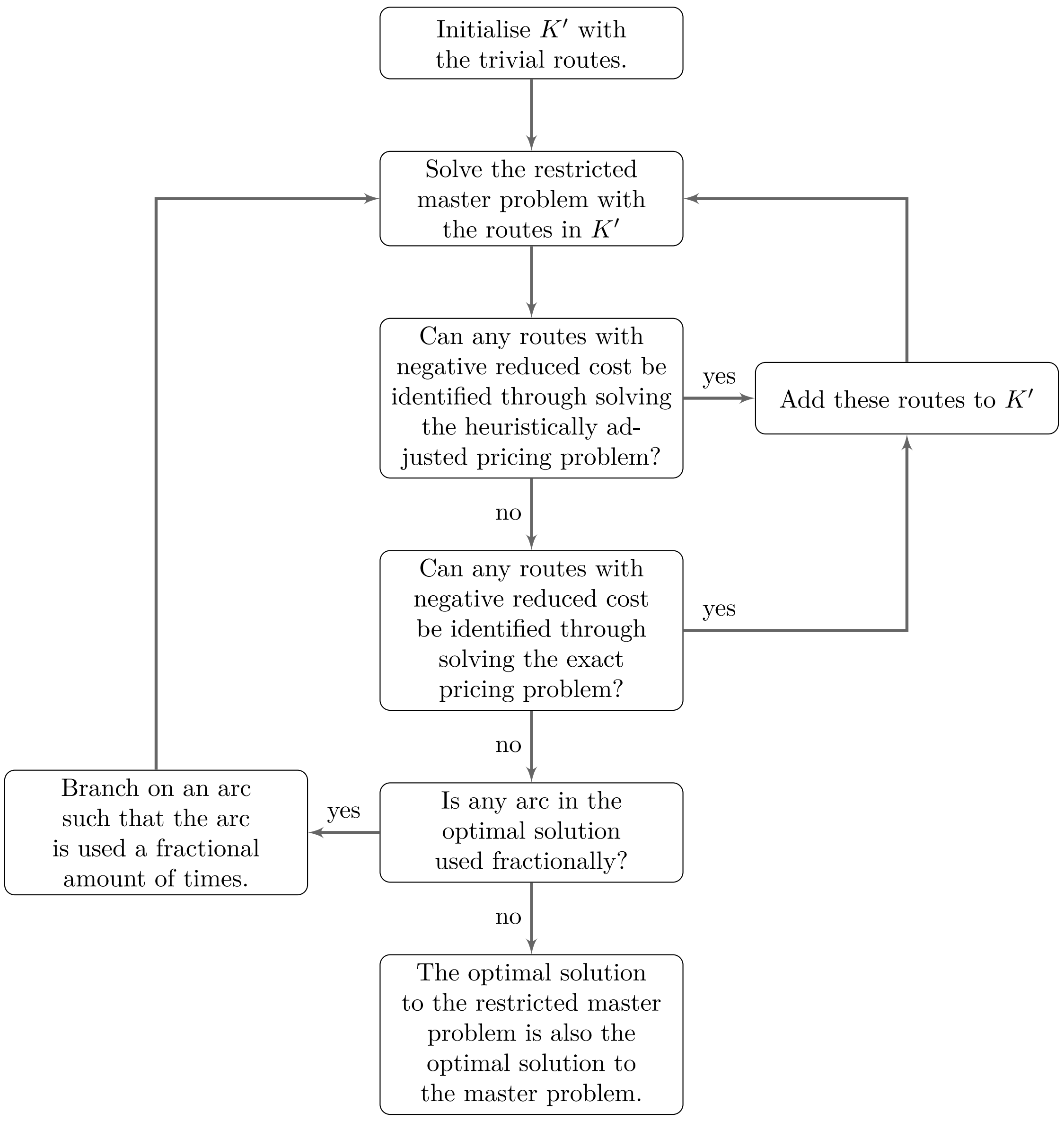

Initialise K' with the trivial routes.
Solve the restricted master problem with the routes in K'
Can any routes with negative reduced cost be identified through solving the heuristically adjusted pricing problem?
yes
Add these routes to K'
no
Can any routes with negative reduced cost be identified through solving the exact pricing problem?
yes
no
Is any arc in the optimal solution used fractionally?
yes
Branch on an arc such that the arc is used a fractional amount of times.
no
The optimal solution to the restricted master problem is also the optimal solution to the master problem.

Figure 5.1: An overview of the branch-and-price procedure.

5.2 The joint network vehicle routing game

In this section several procedures to determine the different cost allocations for the joint network vehicle routing game are presented. First, in Subsection 5.2.1, two algorithms are proposed. These algorithms each determine the costs of all coalitions. Such algorithms are required to calculate some of the more complex cost allocation such as the Shapley allocation. The first of the algorithms presented is a simplistic version in which all optimal costs are calculated independently. The second algorithm utilises information of closely related coalitions to determine the optimal costs more efficiently. Next, in Subsection 5.2.2, a constraint generation approach to solve the Equal Profit Method is proposed. Using this procedure, one does not necessarily need to determine the optimal costs for each coalition. Then, the constraint generating approach is implemented in four different manners, yielding four additional algorithms to determine an allocation of the Equal Profit Method. Finally, an overview of the algorithms is presented in Subsection 5.2.3.

5.2.1 Enumerative algorithms

Some methods require information about the optimal cost for each coalition. A prime example of this is the Shapley value. As the game consists of an exponential amount of coalitions it is crucial that all optimal costs are calculated in an efficient manner. In order to have a basic benchmark, the first algorithm introduced calculates the cost in the most basic way imaginable. In this manner, the worst-case performance, as provided by the basic method, can be compared to each of the proposed methods.

5.2.1.1 `BSC`: Basic algorithm

This algorithm independently calculates the optimal costs of each coalition. The basic algorithm is presented in Algorithm 4.

Algorithm 4 Optimal costs (`BSC`)

```
Procedure
1: for all $S \subseteq N$ do
2:     Initialise $K'$ as the trivial routes.
3:     Solve the vehicle routing problem $VRP(W_S)$ to determine $C(S)$.
4: end
```

5.2.1.2 `ADV`: Advanced algorithm

Most of the vehicle routing problems which are solved for each coalition are very similar. Some coalitions differ only by one player and as such have a similar structure for the underlying vehicle routing problem. Due to this, a coalition may be efficiently initialised with the routes of its subcoalitions. If these routes provide a good starting point, the amount of iterations of the pricing problem may turn out to lower and thereby increase the performance of the algorithm. To that end, let $\Omega_k^S = \{S' \subseteq S : |S'| = k\}$ denote the set of all subsets of coalition S of size k and define R_S as one of the possible set of routes corresponding to the optimal solution to the vehicle routing problem of coalition S. For each coalition S and network W_S, the branch-and-price procedure of the vehicle routing problem is initialised with routes:

$$K'_S = \bigcup_{S' \in \Omega_1^S} R_{S'} \cup \bigcup_{S' \in \Omega_{|S|-1}^S} R_{S'}. \tag{5.25}$$

The inclusion of the routes belonging to coalitions Ω_1^S is to ensure that a feasible solution exists. Because of this initialisation, it is key that the smaller problems are solved first. Then, using the results of the smaller instances, the larger vehicle routing problems are solved. The complete procedure to obtain the optimal costs for each coalition is summarised in Algorithm 5:

Algorithm 5 Optimal costs (`ADV`)

```
Initialisation
 1: for all $i \in V_N$ do
 2:     Initialise $K'$ as the trivial routes.
 3:     Solve the vehicle routing problem $VRP(W_i)$ using the branch-and-price method.
 4:     Set $R_{\{i\}}$ as the routes used in the optimal solution.
 5:     Set $C(\{i\})$ as the optimal cost.
 6: end

Procedure
 7: for $i \in \{2, \dots, |N|\}$ do
 8:     for $S \subseteq N : |S| = i$ do
 9:         Initialise $K'$ as $K'_S$.
10:         Solve the vehicle routing problem $VRP(W_S)$ using the branch-and-price method.
11:         Set $R_S$ as the routes used in the optimal solution.
12:         Set $C(S)$ as the optimal costs.
13:     end
14: end
```

For each coalition S it holds that the amount of routes with which the branch-and-price procedure is initialised is at most $|S|^2 \max_{i \in S}\{|V_i|\}$. A derivation of this fact is provided in Appendix C.

5.2.2 Constraint generating approach

The amount of coalitions within the grand-coalition is exponential in the number of players. In the previous approaches the optimal costs of all coalitions have been determined. However, not in all cases one is required to determine all costs to allocate costs or determine whether an allocation is in the core. To this end, constraint generation can be used to determine if any coalitional rationality constraints are violated. The Equal Profit, Lorenz and Nucleolus methods lend themselves for constraint generation of the coalitional rationality constraints. In this section an in-depth research is performed concerning the applicability of constraint generation for the equal profit method.

Consider the Equal Profit Method as presented in Subsection 2.3.3. Constraints 2.14 will be generated if necessary. In order to ensure that the problem is bounded, the method is initialised with constraints of the stand-alone coalitions. Let the set of coalitions considered in the constraints be denoted as N'. This gives rise to the following restricted problem:

Restricted Equal Profit Method

$$\textbf{Objective:} \quad \min z \tag{5.26}$$

$$\textbf{Subject to:} \quad \frac{x_i}{c(\{i\})} - \frac{x_j}{c(\{j\})} \leq z \quad \forall i, j \in N \tag{5.27}$$

$$\sum_{i \in S} x_i \leq c(S) \quad \forall S \in N' \tag{5.28}$$

$$\sum_{i \in N} x_i = c(N) \tag{5.29}$$

$$x_i \in \mathbb{R}^+ \quad \forall i \in N \tag{5.30}$$

$$z \in \mathbb{R}^+ \tag{5.31}$$

Next, it should be determined if any of the non-considered constraints are violated. This is done through a sub-problem. A coalitional rationality constraint is violated for coalition S if it holds that $\sum_{i \in S} x_i - c(S) > 0$. Now the sub-problem can be defined as the following maximisation problem:

$$\text{SP} = \max_{S \subseteq N} \left\{ \sum_{i \in S} x_i - c(S) \right\} \tag{5.32}$$

if SP > 0 it holds that at least one constraint is violated for a coalition $S \subseteq N$. Now, the rationality constraint for coalition S is added to the set of constraints by setting $N' = N' \cup \{S\}$. Thereafter, the Restricted Equal Profit Method is solved again with updated set N'. This process is repeated until it holds that $SP \leq 0$. In this case an optimal solution to the non-restricted problem has been found.

Note that in the described procedure the optimal costs for all coalitions still ought to be known. In the following some approaches are described such that the pricing problem can be solved without necessarily requiring the optimal costs of all coalitions.

5.2.2.1 `CGL`: Enumerative constraint generation with lower bounds

The approach presented in the following is similar to the constraint generation approach as presented by Göthe-Lundgren et al. [25]. In their paper, the authors apply a constraint generating method to solve the Nucleolus method for the travelling salesman game. Rather than calculating all optimal costs of each coalitions, they calculate all lower bounds for each coalition. As the set partitioning formulation is known to provide good bounds for the vehicle routing problem, the approach is mimicked for the joint vehicle routing network game.

Define the lower bound to the optimal cost of each coalitions as $\overline{C(S)} = \overline{VRP(W_S)}$. Note that for any coalition S it holds that $\sum_{i \in S} x_i - c(S) < \sum_{i \in S} x_i - \overline{C(S)}$. As such, if $\sum_{i \in S} x_i - c(S) > 0$ it must hold that $\sum_{i \in S} x_i - \overline{C(S)} > 0$. The contrary does not need to be true. Now the sub-problem can be solved heuristically by solving the following problem:

$$\overline{\text{SP}} = \max_{S \subseteq N : S \notin N'} \left\{ \sum_{i \in S} x_i - \overline{C(S)} \right\} \tag{5.33}$$

If a coalition S is determined such that the lower bound to the costs violate a constraint, the optimal costs $C(S)$ are calculated and the corresponding constraint is added to the problem. In this manner, not all optimal coalition costs have to be calculated. The following algorithm is suggested to solve the problem associated with the Equal Profit Method. All vehicle routing problems to be solved are initialised with the trivial routes.

Algorithm 6 EPM (CGL)

```
Initialisation
 1: Set $N' = N$
 2: for $i \in N$ do
 3:     Determine $C(\{i\})$
 4: end
 5: Determine $C(N)$.
 6: Set $\overline{\text{SP}} = \infty$.
 7: Determine $\overline{C(S)}$ for all coalitions $S \subset N$.

Procedure
 8: Solve the restricted problem.
 9: for $S \subset N : 1 < |S| < |N|$ do
10:     Determine $\overline{\text{SP}}$.
11:     if $\overline{\text{SP}} > 0$ then
12:         Determine $C(S)$.
13:         Add the corresponding constraint to the restricted problem.
14:         Remove $S$ from $N'$.
15:         Restart the procedure.
16:     end
17: end
```

Unfortunately, this method still requires all lower bounds to be determined. Hence, the algorithm is still exponential in the amount of players.

5.2.2.2 Joint price-collecting vehicle routing game

Rather than solving the sub-problem using the lower bounds, the problem can also be formulated as a mixed integer programming problem. In contrast to the flow formulation presented in Subsection 5.1.1, it is no longer necessary to visit each location exactly once. However, if all locations of a player are visited the cost allocation of that player is added to the objective. This problem could be seen as a generalisation of the price-collecting vehicle routing problem which in turn is a generalisation of the price-collecting travelling salesman problem. In the price-collecting travelling salesman problem, a salesman can visit a number of locations on his trip. With each location a profit is associated. The objective of the game is to maximise the total profits minus the travel costs associated with each visit. A more detailed analysis of the problem is provided by Feillet et al. [16]. In their paper they propose a classification of travelling salesman problems with profits and vehicle routing problems with profits and provide an overview of often used applications and heuristics as well as exact solution procedures to the problem.

In the price-collecting vehicle routing problem, the same principles are applied to the vehicle routing game. Consider the vehicle routing problem. This time a profit is associated with the visiting a location, furthermore, visiting all locations is no longer a requirement. The goal is to find the optimal set of locations to visit such that the profits are maximised. An analysis on the price-collecting vehicle routing problem with non-linear costs and its solution procedures is provided by Stenger et al. [46]. Butt and Ryan [6] propose to use a column generation approach for the price-collecting vehicle routing problem. Their results show promising computational results.

In the following, two solution paths to solve the price-collecting joint vehicle routing problem are explored. First, an approach altering the flow formulation for the price-collecting vehicle routing problem is introduced. Next, an approach to the price-collecting joint vehicle routing problem using a branch-and-price procedure is introduced.

Flow formulation.

The joint price-collecting vehicle routing problem is a generalisation of the price-collecting vehicle routing problem. Rather than awarding a reward if a location is visited, a reward is given for visiting a set of locations. Let x_i be the reward for visiting all location of player i and define z_i as the decision variable indicating whether all locations of player i are visited in the current solution. Besides this, let δ_v be the variable indicating if location v has been visited in the current solution. This gives rise to the following linear price-collecting vehicle routing problem:

Price-collecting vehicle routing problem

Objective:
$$SP = \max \sum_{l \in N} z_l x_l - \sum_{i,j \in A} c_{ij} x_{ij} \tag{5.34}$$

Subject to:
$$\sum_{i \in V_N \cup \{v_d\}} x_{ij} = \delta_j \quad \forall j \in V_N \tag{5.35}$$
$$\sum_{i \in V_N \cup \{v_d\}} x_{ik} - \sum_{j \in V_N \cup \{v_d\}} x_{kj} = 0 \quad \forall k \in V_N \tag{5.36}$$
$$\sum_{i \in V_N \cup \{v_d\}} y_{ij} - \sum_{i \in V_N \cup \{v_d\}} y_{ji} = d_j \quad \forall j \in V_N \tag{5.37}$$
$$d_j x_{ij} \leq y_{ij} \leq (Q - d_i) x_{ij} \quad \forall i,j \in V_N \cup \{v_d\} : i \neq j \tag{5.38}$$
$$z_l \leq \delta_i \quad \forall l \in N, i \in V_{\{l\}} \tag{5.39}$$
$$\sum_{l \in N} z_l \geq 2 \tag{5.40}$$
$$\sum_{l \in N} z_l \leq |N| - 1 \tag{5.41}$$

$$\sum_{(i,j) \in T \times T : i \neq j} x_{ij} \leq 1 \quad \forall T \subseteq V_n : |T| = 2, q(T) \leq Q \tag{5.42}$$
$$\sum_{(i,j) \in T \times T : i \neq j} x_{ij} \leq 0 \quad \forall T \subseteq V_n : |T| = 2, q(T) > Q \tag{5.43}$$
$$\sum_{(i,j) \in T \times T : i \neq j} x_{ij} \leq 2 \quad \forall T \subseteq V_n : |T| = 3, q(T) \leq Q \tag{5.44}$$
$$\sum_{(i,j) \in T \times T : i \neq j} x_{ij} \leq 1 \quad \forall T \subseteq V_n : |T| = 3, q(T) > Q \tag{5.45}$$

$$\delta_i \in \{0,1\} \quad \forall i \in V_N \tag{5.46}$$
$$y_{ij} \geq 0 \quad \forall i,j \in V_N \cup \{v_d\} : i \neq j \tag{5.47}$$
$$x_{ij} \in \{0,1\} \quad \forall i,j \in V_N \cup \{v_d\} : i \neq j \tag{5.48}$$
$$z_l \in \{0,1\} \quad \forall l \in N \tag{5.49}$$

Constraints (5.35) ensure that each location is visited exactly once or not all. Constraints (5.36) ensure that if a location has an incoming arc it must also have an outgoing arc. This ensures that each path can only end and start at the depot. Constraints (5.37) are also known as the commodity flow constraints. These constraints ensure that the difference in flow before and after a location equals the demand of that location. As such, these constraints help to keep track of the total demand supplied by a route. Constraints (5.38) ensure that the maximum vehicle capacity is not exceeded by any route. Constraints (5.47) ensure that the flow of goods is positive. Constraints (5.48) ensure that each arc is used once or not at all. Constraints (5.39) ensure that the cost allocation of a player is only added to the objective if all of its locations are visited exactly one. In this case the player is added to the coalition. Constraints (5.40) and (5.41) are cuts to the tighten the formulation. Constraints (5.42),(5.43),(5.44) and (5.45) represent cuts as proposed by Yaman [49]. These inequalities are only applied to subsets of size 2 or 3. Constraints (5.46) ensure that each location is visited exactly once or not at all and constraints (5.49) enforce the binary choice whether a player in included in the coalition.

Set-partitioning formulation.

The joint price-collecting vehicle routing problem can also be solved by the previously defined branch-and-price procedure. To this end, the joint price-collecting set partitioning formulation is defined:

Joint price-collecting set partitioning formulation

$$\textbf{Objective:} \quad VRP(W_S) = \max \sum_{i \in N} x_i z_i - \sum_{k \in K} c_k x_k \tag{5.50}$$

$$\textbf{Subject to:} \quad \sum_{k \in K} a_{vk} x_k = z_i \quad \forall\, v \in V_N, i : v \in V_{\{i\}} \tag{5.51}$$

$$x_k \in \{0,1\} \quad \forall k \in K \tag{5.52}$$

$$z_i \in \{0,1\} \quad \forall i \in N \tag{5.53}$$

Constraints (5.51) ensure that each player is visited exactly once or not at all. Constraints (5.52) ensure that each route is used exactly once or not at all in the final solution. Constraints (5.53) ensure that the indicator, indicating whether each player is included in the most violating coalition, is binary. Now, the following master problem can be defined.

Joint price-collecting Master Problem

$$\textbf{Objective:} \quad VRP(W_S) = \max \sum_{i \in N} x_i z_i - \sum_{k \in K} c_k x_k \tag{5.54}$$

$$\textbf{Subject to:} \quad \sum_{k \in K} a_{vk} x_k = z_i \quad \forall\, v \in V_N, i : v \in V_{\{i\}} \tag{5.55}$$

$$x_k \in \mathbb{R}^+ \quad \forall k \in K \tag{5.56}$$

$$z_i \in \mathbb{R}^+ \quad \forall i \in N \tag{5.57}$$

Constraints (5.55) ensure that each player is visited exactly once or not at all. The constraints (5.56) and (5.57) are relaxed to allow for non-binary values. Once more, constraints (5.55) are used to generate the dual variables used in the calculation of the reduced cost. Now, the procedure as explained in Section 5.1.2.2 through 5.1.3 can be utilised to exactly solve the price-collecting problem.

Note that one does not need to branch on the inclusion of any of the players. As the decision variables x_k are binary after the branch-and-price procedure, one can see that constraints (5.55) enforce that the decision variables z_i are binary as well. However, in this thesis, the choice has been made to branch on the players in order to improve overall performance.

In the following, three algorithms are proposed. Each of these algorithms are based on a formulation of the joint price-collecting vehicle routing problem.

5.2.2.3 `CGP`: Constraint generation using the exact flow formulation of the price-collecting vehicle routing problem

The flow formulation of the joint price-collecting vehicle routing problem can be solved using a mixed integer problem solver such as CPLEX. The algorithm for solving the joint network vehicle routing game using the exact flow formulation of the joint price-collecting problem can be formalised as follows:

Algorithm 7 EPM (`CGP`)

```
Initialisation
 1: for $i \in N$ do
 2:     Determine $C(\{i\})$.
 3: end
 4: Determine $C(N)$.

Procedure
 5: Solve the restricted problem.
 6: Determine SP using the price-collecting vehicle routing problem.
 7: while $SP > 0$ do
 8:     Determine $C(S)$.
 9:     Add the corresponding constraint to the restricted problem.
10:     Solve the restricted problem.
11:     Determine SP using the price-collecting vehicle routing problem.
12: end
```

Note that the optimal costs of the coalition which violates the coalitional constraints the most have been determined already by solving the joint-price collecting problem. As such, the branch-and-price procedure does not need to be utilised for this method.

5.2.2.4 `MIX`: Constraint generation using the relaxed flow formulation of the price-collecting vehicle routing problem

The exact formulation of the joint price-collecting vehicle routing problem can be resource intensive to solve. As such, one might consider to use the relaxation of the joint price-collecting vehicle routing problem. In this relaxation the use of an arc x_{ij} is relaxed such that $x_{ij} \in \mathbb{R}_+$. Then, with the relaxed joint price-collecting vehicle routing problem any possibly violated constraint could be generated using the lower bound to the costs of the coalition. This approach is very similar to the first constraint generation approach presented. After a possibly violated constraint has been determined, the branch-and-price procedure is executed and the optimal coalition costs are used to add the corresponding coalitional rationality constraint. After generating the constraint, the sub-problem should be adjusted to ensure that the same coalition S

with the relaxed price-collecting vehicle routing problem is not generated again. This is done through adding the following constraints to the relaxation of the price-collecting vehicle routing problem:

$$\sum_{l \in S} z_l - \sum_{l \in N \setminus S} z_l \leq |S| - 1. \tag{5.58}$$

Without these constraints the algorithm can continue to cycle on a coalition. This happens when the lower bounds to the optimal costs cause the algorithm to think a constraint is violated, even if it already has been added to the problem. Using these cuts, a finite termination of the procedure is guaranteed. For clarification the procedure is summarised in the following algorithm:

Algorithm 8 EPM (`MIX`)

```
Initialisation
 1: for $i \in N$ do
 2:     Determine $C(\{i\})$.
 3: end
 4: Determine $C(N)$.

Procedure
 5: Solve the restricted problem.
 6: Determine $\overline{\text{SP}}$ using the relaxed price-collecting vehicle routing problem.
 7: while $\overline{\text{SP}} > 0$ do
 8:     Determine $C(S)$.
 9:     Add the corresponding constraint to the restricted problem.
10:     Solve the restricted problem.
11:     Add a cut (5.58) to the relaxed price-collecting vehicle routing problem to ensure $S$ is
    not generated again.
12:     Determine $\overline{\text{SP}}$ using the relaxed price-collecting vehicle routing problem.
13: end
```

5.2.2.5 `CGC`: Constraint generation using the set partitioning formulation of the price-collecting vehicle routing problem

Using this procedure the most violated constraint is generated and added to the restricted problem. This time, the set-partitioning formulation for the joint price-collecting vehicle routing game is utilised. The procedure is summarised in the following algorithm.

Algorithm 9 EPM (CGC)

```
Initialisation
 1: for $i \in N$ do
 2:     Determine $C(\{i\})$
 3: end
 4: Determine $C(N)$.

Procedure
 5: Solve the restricted problem.
 6: Determine $SP$ using the branch-and-price approach for the price-collecting vehicle routing
    problem.
 7: while $SP > 0$ do
 8:     Determine $C(S)$.
 9:     Add the corresponding constraint to the restricted problem.
10:     Solve the restricted problem.
11:     Determine $SP$ using the branch-and-price approach for the price-collecting vehicle rout-
    ing problem.
12: end
```

Note that the optimal solution to the sub-problem does not only provide any information about which coalition is the most violating coalition. The optimal costs of this coalition can also be determined from the optimal solution. As such, the branch-and-price procedure for the vehicle routing game is not utilised to determine the optimal costs of any generated coalition. The branch-and-price procedure is used during the initialisation of the algorithm.

5.2.3 An overview of the algorithms

In this subsection a short overview of all proposed algorithms is provided. The different characteristics of each algorithm are provided in Table 5.1. If no text is present for a combination of one of the characteristics and one of the algorithms, the characteristic does not apply to the algorithm.

Table 5.1: An overview of the characteristics of each of the proposed solution algorithms.

	Algorithms					
Characteristics	BSC	ADV	CGL	CGP	MIX	CGC
General:						
Coalitions considered	All	All	All	Generated	Generated	Generated
Coalitions solved to optimality	All	All	Generated	Generated	Generated	Generated
Retains columns from previous solutions	No	Yes	No	No	No	Yes
Constraint generation:						
Sub problem solved to optimality			No	Yes	No	Yes
Price-collecting formulation				Flow	Flow	Set partitioning

6 Experimental design

In this chapter the experimental design of this thesis is discussed. The aim of the experiments is to determine which of the solution algorithms (as discussed in Subsection 5.2) proves to be efficient in solving the game. Furthermore, the general performance of the algorithms as well as the performance of the allocation methods is discussed based on multiple criteria. To execute such experiments a set of instances has to be designed. In the literature study that has been performed for this thesis, no benchmark cases for the joint network vehicle routing game were found. As such, part of this chapter is concerned with the introduction of several sets of randomly generated instances of the joint network vehicle routing game. These random instances are designed in a variety of manners to observe the general performance of the solution approaches and the applicability of the game.

First, in Section 6.1, a procedure to randomly generate instances of the joint network vehicle routing game is proposed. This procedure also introduces a set of parameters to shape the generated instances. Besides this, by varying the parameters of the randomly generated instances several types of instances are introduced. The aim of the different types of instances is to observe the real-life applicability of the method and to push the algorithms to their limits. As such, the spread of the locations and the amount of players and customers will be varied. Next, In Section 6.2 several performance criteria for the allocation methods are introduced. Using these criteria the performance of the allocation methods on the different types of instances of the joint network vehicle routing game can be discussed.

6.1 Game instances

In this subsection a procedure to generate random game instances is introduced. The procedure allows for the generation of instances with varying amounts of players and locations. Furthermore, parameters are added to adjust the shape of the instance and whether the locations of a player are clustered. Subsequently, several sets of instances are defined.

6.1.1 Generation procedure

Each instance is designed with a centralised depot in mind. As such, all locations are located in a circle around the origin. Despite the fact that the instance is designed around this centralised depot, one could easily relocate the depot afterwards. Next, each player is assigned a coordinate. Then, the locations of each player are generated around this coordinate. To that end, consider a game of n players. A polar coordinate, given by a r_i and φ_i, is generated for each player using a maximum radius R and the range $[0, 2\pi]$. Hence, it should hold that $0 \leq r_i \leq R$ and $0 \leq \varphi_i \leq 2\pi$ for each player i. These variables are randomly generated in such a manner that the distribution of the player coordinates is uniform in the region described by the parameter R. The process of generating these coordinates is visualised in Figure 6.1 for two players; A and B.

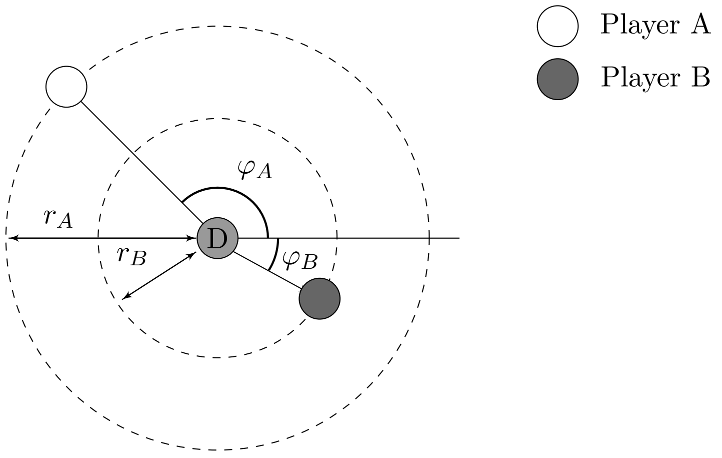
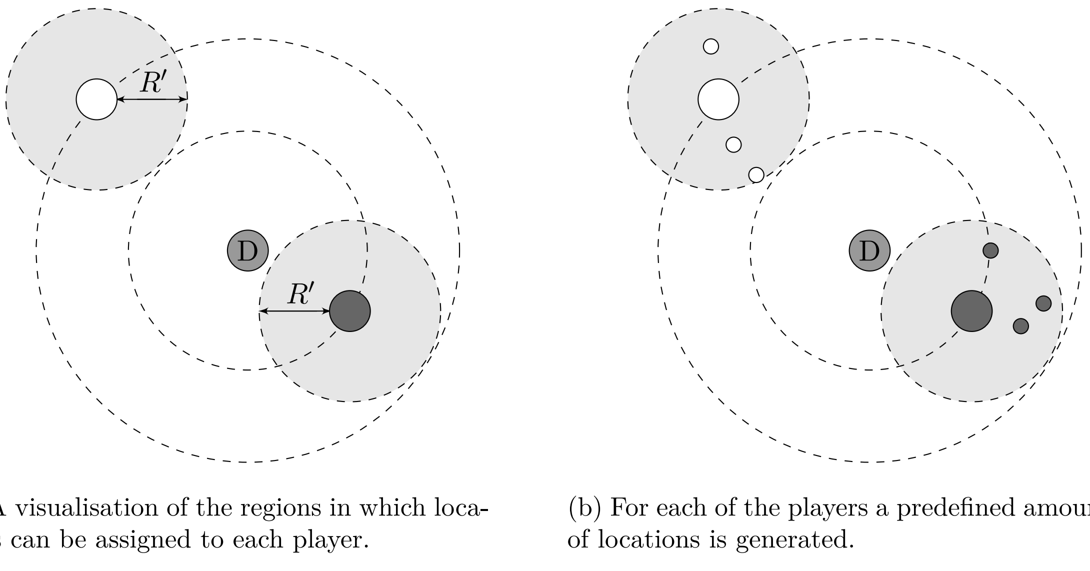


Figure 6.1: Step one of the instance creation procedure. Locations are assigned to each player.

After the coordinates of each player have been defined, the locations are assigned to each of the players. For this purpose the parameter m, representing the maximum amount of locations to be generated, is introduced. If the parameter m is a multiple of the amount of players, each player is assigned an equal amount of locations. If m is not a multiple of the amount of players, some players are assigned an additional location. In this manner, the difference in locations shall never exceed one. Next, the locations belonging to each player are generated. Each location is generated with the player coordinate at its centre using parameter R'. The coordinate of each location l is parametrised by r'_l and φ'_l such that $0 \leq r'_l \leq R'$ and $0 \leq \varphi'_l \leq 2\pi$. This process is visualised in Figure 6.2 for two players.



(a) A visualisation of the regions in which locations can be assigned to each player.

(b) For each of the players a predefined amount of locations is generated.

Figure 6.2: The generation of the locations for each player.

Note that these examples solely serve as illustrations to the generation procedures. The example as presented above does not represent any of the generated instances. However, it does show how the parameters affect the generated instances.

Another example of an instance with parameters $R = 0, m = 15, R' = 125$ and $n = 5$ is presented in Figure 6.3. The costs corresponding to each arc, between each of the locations, are determined using the Euclidean distance.

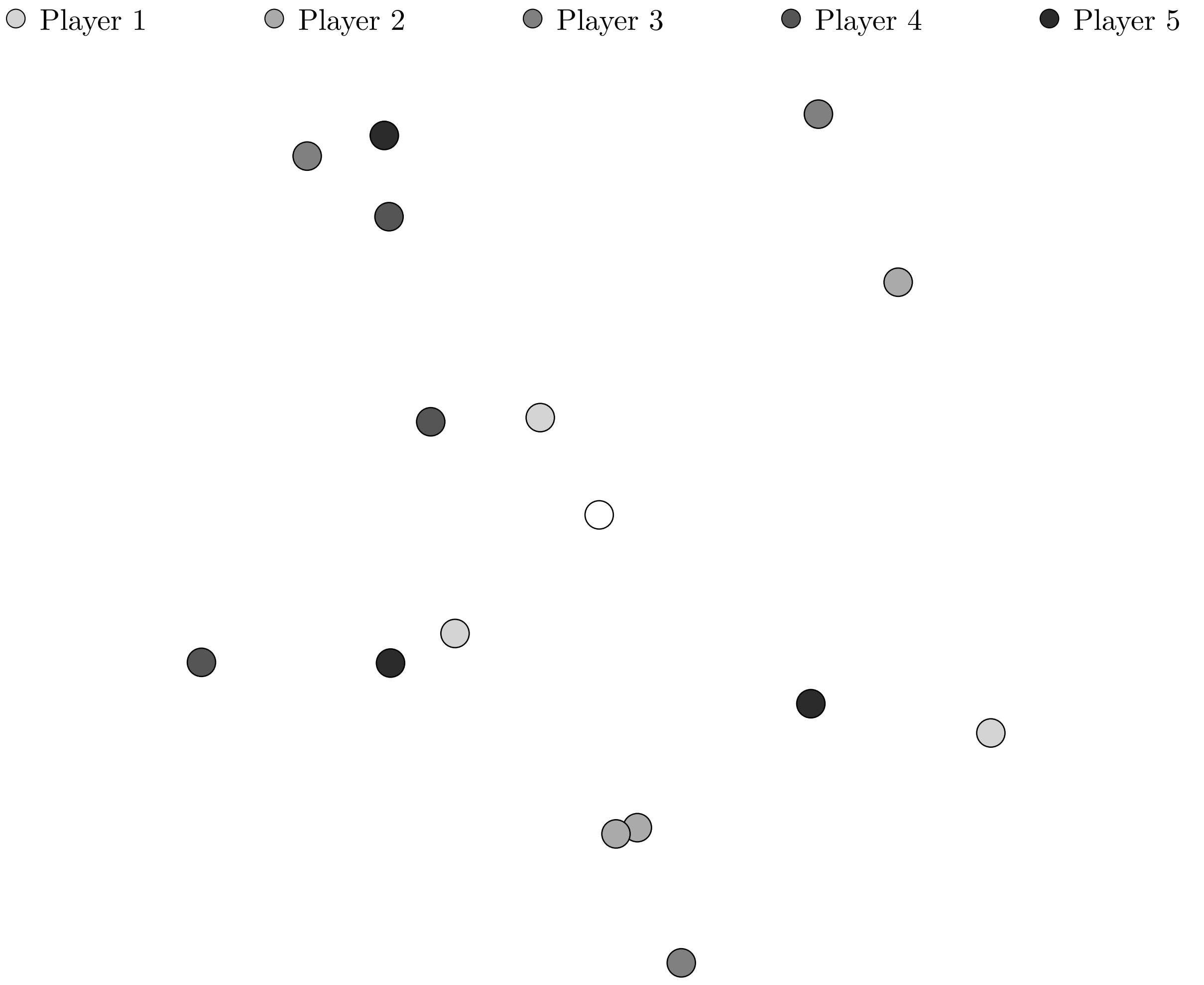


Figure 6.3: An example of a generated instance, the white dot represents the depot.

6.1.2 Instance types

Each of the instances is fully characterised by the parameters n, m, R and R'. To evaluate the methods in a variety of settings several values for the parameters will be used. The following instance types will be generated:

- **A**: Small instances of the vehicle routing game. These instances consist of 10 players and a total of 10 locations. The locations are spread randomly, with parameters $R = 0$ and $R' = 125$.
- **B**: Small instances of the joint network vehicle routing game. These instances consist of 5 players and a total of 15 locations. The locations are spread randomly, with parameters $R = 0$ and $R' = 125$.
- **C**: Small instances of the joint network vehicle routing game. These instances consist of 5 players and a total of 15 locations. The locations are clustered around their players, with parameters $R = 100$ and $R' = 25$.

D: Medium instances of the joint network vehicle routing game. These instances consist of 10 players and a total of 20 locations. The locations are spread randomly, with parameters $R = 0$ and $R' = 125$.

E: Medium instances of the joint network vehicle routing game. These instances consist of 10 players and a total of 20 locations. The locations are clustered around their players, with parameters $R = 100$ and $R' = 25$.

F: Large instances of the joint network vehicle routing game. These instances consist of 5 players and a total of 25 locations. The locations are spread randomly, with parameters $R = 0$ and $R' = 125$.

G: Large instances of the joint network vehicle routing game. These instances consist of 12 players and a total of 12 locations. The locations are spread randomly, with parameters $R = 0$ and $R' = 125$.

For each of these types of instances except for types E and G, sets of 50 instances will be generated in which each location has a demand uniformly distributed between 0 and 40. For type E a total of 20 instances is generated and for instances of type G a total of 10 instances are generated. The maximum vehicle capacity is taken as 100.

6.2 Performance criteria

Each allocation method will be judged based on three different criteria. These criteria are: the percentage of allocations in the core, the consistency of the method and the computational time required. The first and last criteria do not require any additional explanation. The consistency of a method can be judged in many different ways, as such the methodology used to judge the consistency of an allocation method is described below.

The consistency of each method will be based on the determined allocations and characteristics of each instance. For this purpose a linear regression model, similar to the one presented by Naber et al. [34], is utilised. The correlation between several factors and the allocated cost to a player is taken into account. These factors are as follows:

1. Average distance of the player's locations to the locations of all other players.
2. Average distance of the player's locations to the depot.
3. Average distance between the locations of the player.
4. Total demand.

It should be noted that the stand-alone cost is not taken into account as it is not necessarily independent from the other factors presented. As such, including this variable may influence the results.

7 Computational results

In this chapter the computational performance and results of the proposed algorithms and allocation methods are presented. Solving the joint network vehicle routing game can be a tedious procedure. At first sight, one ought to solve a problem with exponential time complexity an exponential amount of times. The proposed basic and advanced algorithms are prime examples of this. Such procedures become very computationally intensive as the amount of locations increases. Luckily enough, many similar vehicle routing instances have to be solved. The advanced algorithm has been designed to take advantage of this property. However, one does not necessarily need to compute all optimal costs for each coalition. This is due to the fact that the optimal costs of all coalitions are primarily required to ensure that a cost allocation is in the core and not to determine the actual cost allocation. As such, a constraint generation approach has been utilised in four different manners. The first approach based on constraint generation utilises the lower-bounds to the set-partitioning formulation of the vehicle routing problem to determine whether a constraint needs to be generated. In the second and third approach the constraints are generated using the exact formulations of the price-collecting joint vehicle routing problem. In the fourth and final approach both aforementioned approaches are combined to generate any possibly violated constraints. For now the constraint generation approaches have only been utilised for the Equal Profit Method, but one could easily extend this method to the Lorenz and Nucleolus allocations.

This chapter is structured in the following manner. First, in Section 7.1, a visualisation of the results is provided for three different instances of the joint network vehicle routing game. For each of these instances the situations before and after collaboration are presented along with the savings achieved. The purpose of this section is to give some insight into the newly defined joint network vehicle routing game and its applications. Each of these examples have been selected to show a particular characteristic of the game and its allocations. As such, they provide an indication of what to expect, in terms of savings and behaviour, from the joint network vehicle routing game. In Section 7.2, the cost allocations methods are compared based on stability, consistency and computational performance of each method. These characteristics are thought to be crucial in the adaptability of the new game in combination with the allocation methods. Finally, in Section 7.3, the computational efficiency of the proposed algorithms is discussed. For this purpose an overview of the computational time required for each algorithm on each type of instance is presented. Using this data, the overall performance of the algorithms is discussed.

7.1 A visualisation of the results

The purpose of this section is to provide some insight in the instances of the joint network vehicle routing game and its solutions. For the numerical results, multiple instances have been created and solved using the six presented algorithms. The different types of instance have been introduced in Section 6.1.2 and differ in amount of players, locations and structure of the locations. A short summary of the instance characteristics is provided in Table 7.1.

Table 7.1: A short summary of the types of instances used for the computational results.

	Instance type						
Characteristic	A	B	C	D	E	F	G
Amount of players	10	5	5	10	10	5	12
Amount of locations	10	15	15	20	20	25	12
Amount of instances	50	50	50	50	20	50	10
Clustered locations	no	no	yes	no	yes	no	no

It should be noted that the amount of instances has been limited for instances of type E and G. This is due to the computational performance observed in these instances. Both types required significantly more computational resources compared to the other instance types. Nevertheless, the amounts of instances of type E and G are still large enough to observe any difference in performance between the proposed algorithms.

In the following, several examples of instances of the joint vehicle routing game are presented. These examples are taken from instance types A, B and C. The motivation behind this choice is that these instances provide an insight to the different characteristics of the instances while still being small enough to visualise.

First, an example of an instance of type A is given. The instance of the joint network vehicle routing game is presented in Figure 7.1. In all following figures, the locations are given different shades of grey corresponding to the player they belong to. In this case each player has a different colour and as such belongs to a different player. The white dot represents the depot, the origin and sink of all routes. The routes are displayed as the black arcs in the figures. Note that the figure is split into two separate figures. The figure on the left indicates the situation before any form of collaboration. In this case, each player is responsible for his own transport and as such, for each player, a route exists from the depot to their location and back to the depot. The figure on the right shows the optimal result if all players do cooperate. No longer are ten vehicles required to serve each customer, with collaborative transport only three vehicles are required. Besides this, the optimal costs have also been reduced significantly. The cost before collaborating was 1550.00 while collaborating reduced the total cost to 762.80, a cost reduction of 51%. This shows the effectiveness of collaborative transportation.

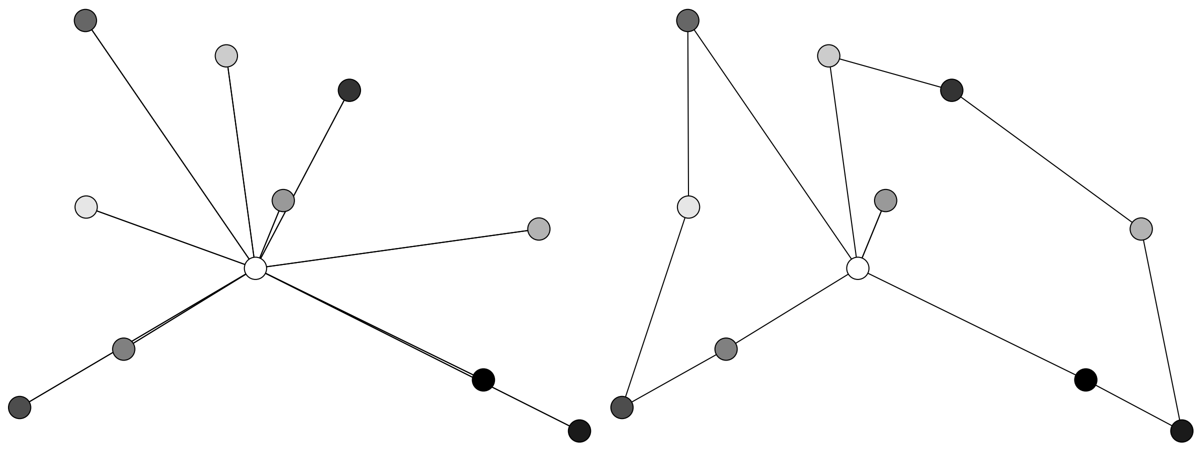
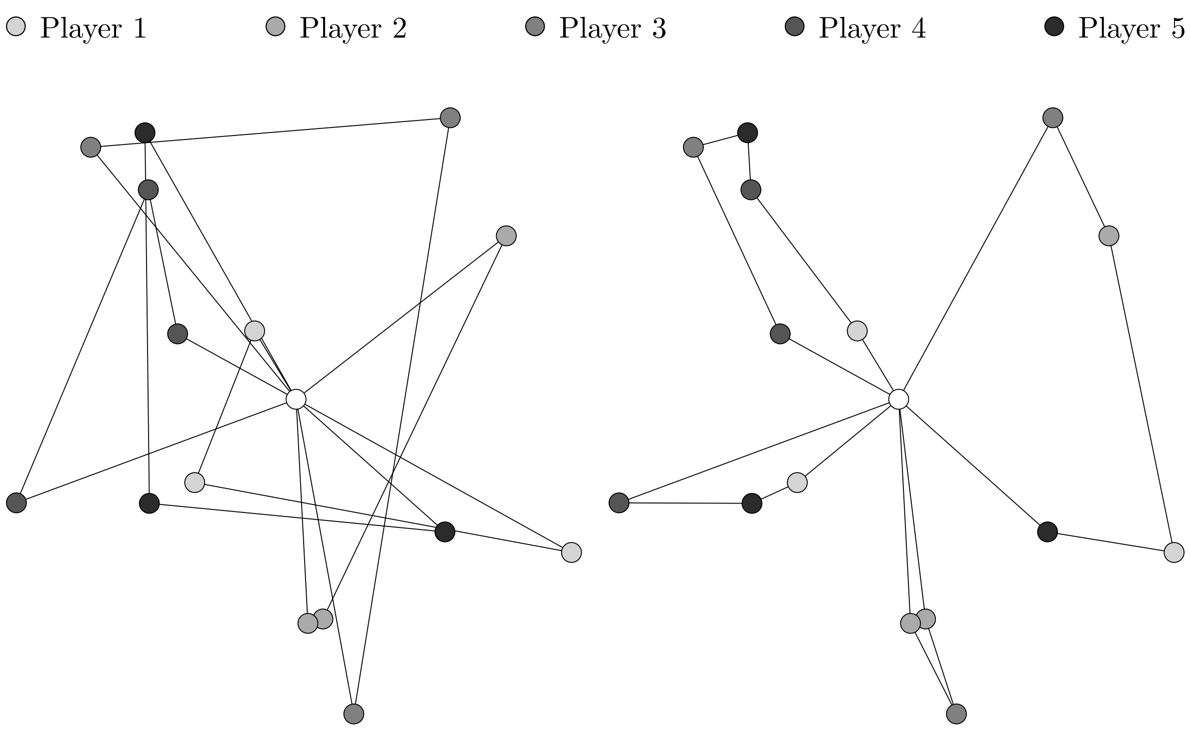

(a) Before collaboration. (b) After collaboration.

Figure 7.1: A comparison of the situation before and after collaboration. This example corresponds to an instance of type A. The total cost before collaboration equals 1550.00 and after collaboration the total cost equals 762.80. In this example, the core is empty.

One might notice the fact that in the optimal solution, one player is still serviced as if he has not joined the grand-coalition. This phenomenon can be attributed to the fact that with an average demand of 20 units and a total of 10 locations, it is unlikely that all locations can be serviced using two trucks with a capacity of 100 units. This means that in the optimal solution this location cannot be serviced on the two other routes.

This phenomenon has some further implications for the core of the game. It can be deduced that the 'abandoned' player can work together with at least one other player in a small coalition, as the demand of each player cannot exceed 40. Furthermore, as the cost of each arc is proportional to the Euclidean distance, the optimal cost for these two players ought to be lower than the sum of the stand-alone costs. As such, it can be concluded that profitable alternatives exist for the 'abandoned' player should he choose to leave the grand-coalition. However, if the other nine players form a coalition they would pay the current optimal cost minus the stand-alone costs of the 'abandoned' player. This indicates that the coalition of nine are not willing to pay more than their optimal costs, whereas the 'abandoned' player will not pay his stand-alone costs since in his vision profitable alternatives exist. This results in the core being empty for this game.

Now an example of a solution of an instance of type B is provided. The instance with both initial and optimal routes is displayed in Figure 7.2. It can be seen that the locations of each player are not clustered. This causes the initial solution the be quite chaotic. For each player, trucks have to travel multiple times across the entire space causing for lengthy routes and high stand-alone costs. The sum of all stand-alone costs equals 2101.49. After collaboration it can be seen that the routing is significantly more efficient, the total costs have been reduced to 1158.27, a cost reduction of about 45%.



(a) Before collaboration. (b) After collaboration.

Figure 7.2: A comparison of the situation before and after collaboration. This example corresponds to an instance of type B. The total cost before collaboration equals 2101.49 and after collaboration the total cost equals 1158.27. In this example, the core is non-empty.

For the game corresponding to the example of instance type B, the core is non-empty. As such, no player benefits in any way whatsoever from leaving the grand-coalition. Unfortunately, it is rather difficult to visualise the actual core of the game belonging to this example. However, it is possible to give some insight into the benefits each coalition can attain through collaboration. A visualisation of the optimal costs when collaborating versus the sum of stand-alone costs of the players belonging to each coalition is presented in Figure 7.3. The outer polygon represents the sum of the stand-alone costs for the players in a coalition. The corresponding players are indicated by the numbers near the vertices of the polygon. The inner polygon represents the optimal costs for a coalition divided by the stand alone costs. As such, the grey area is an indication of the savings that each coalition can achieve relative to the sum of their stand-alone costs. The relative savings are indicated in a linear fashion from 0 percent at the outer circle inwards to 100 percent at the origin.

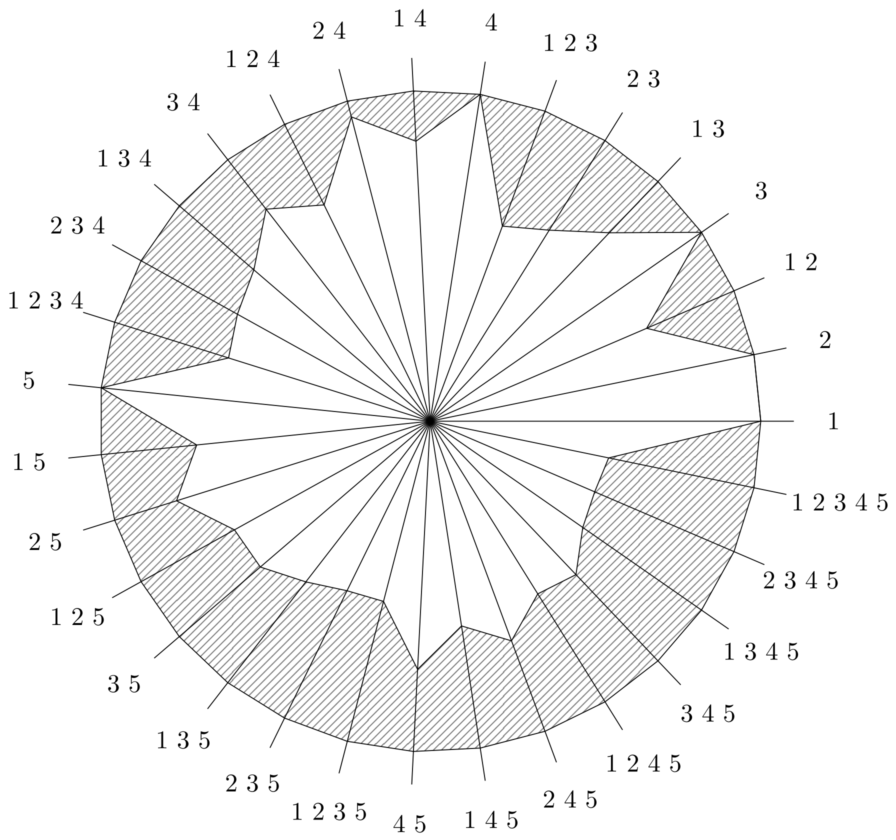


Figure 7.3: The optimal costs (inner-polygon) when collaborating versus the sum of stand-alone costs (outer-polygon) of the players belonging to each coalition.

From the figure it is apparent that for each coalition consisting of multiple players, the optimal cost is lower than the stand-alone cost of each customer. The reason that all coalitions yield significant cost savings is due to the fact that the locations of each player are not clustered. As such, much is to be gained when joining forces. How much one can gain is determined by the allocation methods. An overview of the costs allocated to each player versus the stand-alone costs is presented in Figure 7.4. The grey bar indicates the stand-alone cost whereas the white bar indicates the cost after collaboration with all other players.
For this instance it holds that the Star and Shapley allocations are not in the core. As the core is non-empty, the Lorenz, EPM and Nucleolus allocations are in the core. Here the differences between the allocation methods become visible. The star method allocates the cost relative to the stand-alone cost of each player. This can clearly be seen as players 1 and 2 have similar stand-alone costs and are allocated about equal costs by the star method. As this does not include any information on the underlying coalitions, the allocation is unlikely to be in the core, as is the case in this example. The Shapley allocation does include information about all coalitions, but is just like the star allocation, not in the core. The Lorenz allocation searches for a core allocation such that every player is allocated an equal amount, this is clearly visible in the allocation for this instance. The Equal profit method searches an allocation in the core such that each player is allocated an equal share of the costs proportional to their stand-alone costs.

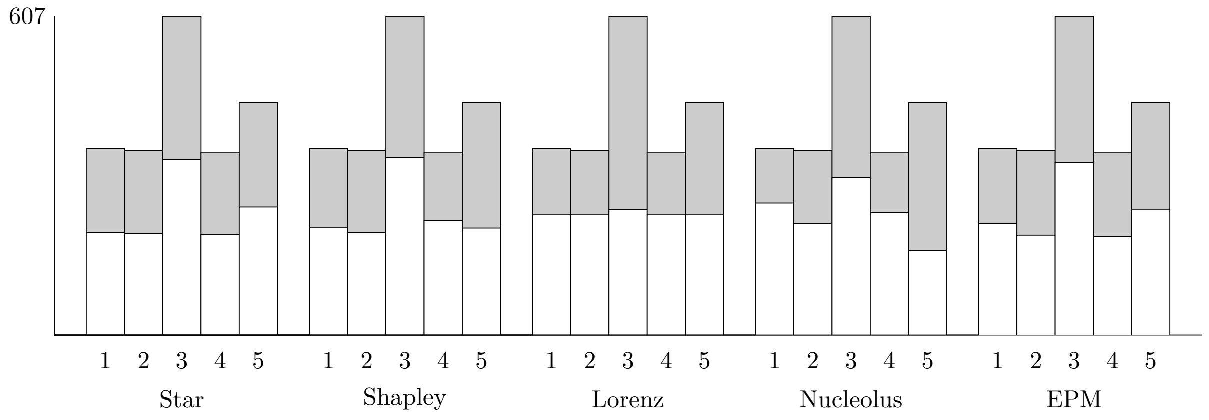
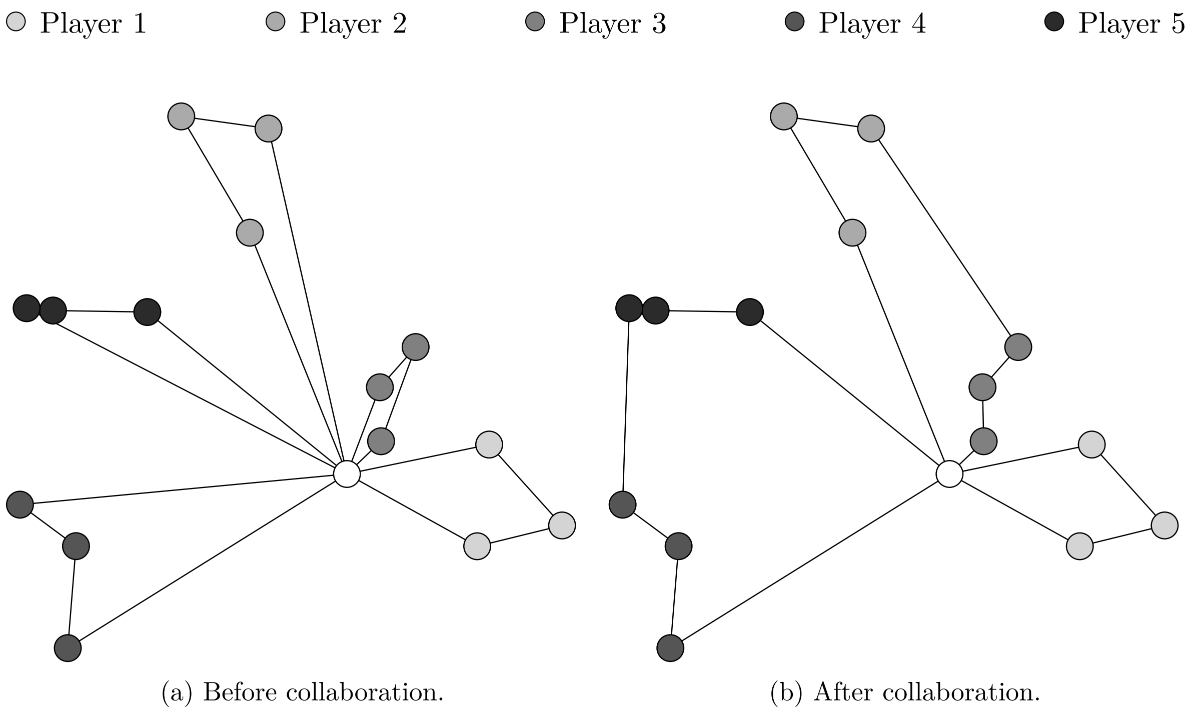


Figure 7.4: Allocated costs for the player in instance B3. The white area indicates the allocated cost whereas the white and grey area combined indicate the stand-alone cost of each player.

In the next example, an Instance of type C is considered. For these instances the locations are clustered. As such, it is to be expected that the grey areas in the coalitional cost visualisation become smaller as there is less room for improvement. First, the solution of the instance before and after collaboration is presented in Figure 7.5.

For this instance it can be seen that the optimal solution does not differ a lot from the initial solution. However, two less trucks are required for transportation and the total costs decrease from 895.26 to 692.52 a cost reduction of 22.6%. This shows that even if the locations of each player are clustered, significant savings can be achieved through collaboration.



(a) Before collaboration.

(b) After collaboration.

Figure 7.5: A comparison of the situation before and after collaboration. This example corresponds to an instance of type C. The total cost before collaboration equals 895.26 and after collaboration the total cost equals 692.52. In this example, the core is non-empty.

Player one, despite cooperating is not included in the optimal solution. This means that for any allocation within the core, player one is allocated his stand-alone costs and as such does not benefit from cooperative transportation. This effect is noticeable in the visualisation of the optimal and stand-alone costs of each coalition in Figure 7.6. When looking at coalitions 1 2, 1 3, 1 4 and 1 5 it is apparent that no cost savings are achieved by letting player one cooperate with any of the other players. One would expect this to lead to an empty core for the instance. Surprisingly enough, this is not the case. It turns out that no coalition at all profits from adding player one. As such, an allocation in which player one is allocated his stand-alone costs is still in the core. Notice the difference between this example and the previously discussed example of an instance of type A in which the core was empty. A player that is not included in the final optimal solution does not necessarily cause an empty core.

Another interesting fact is that the grey area in Figure 7.6 is a lot smaller than the grey area in Figure 7.3. This shows that cooperative transportation is less fruitful in clustered instances compared to non-clustered instances.

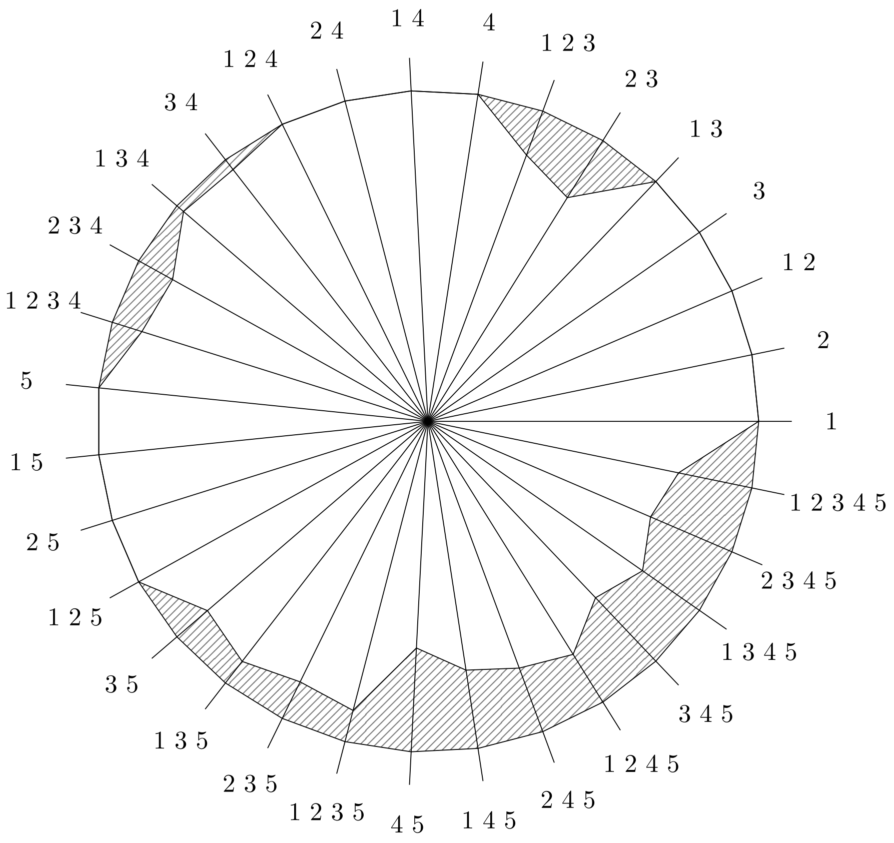


Figure 7.6: The optimal costs (inner-polygon) when collaborating versus the sum of stand-alone costs (outer-polygon) of the players belonging to each coalition.

The costs allocated to each player by each of the allocation methods is presented in Figure 7.7. All of the allocations, except for the allocation by the star method, are in the core. It can be seen that all methods except the Star method correctly identify player 1 as dummy player. Besides this, it can be seen that the Lorenz method allocated player 3 his stand-alone cost despite the fact that he contributes to a lower total cost. This can be seen in Figure 7.6 when looking at coalitions 2 3, 3 4 and 3 5. However, the Lorenz method gives players 3, 4 and 5 the full benefit of these cost reductions. In this manner player 3 is not stimulated to collaborate making the Lorenz allocation not favourable in this case.

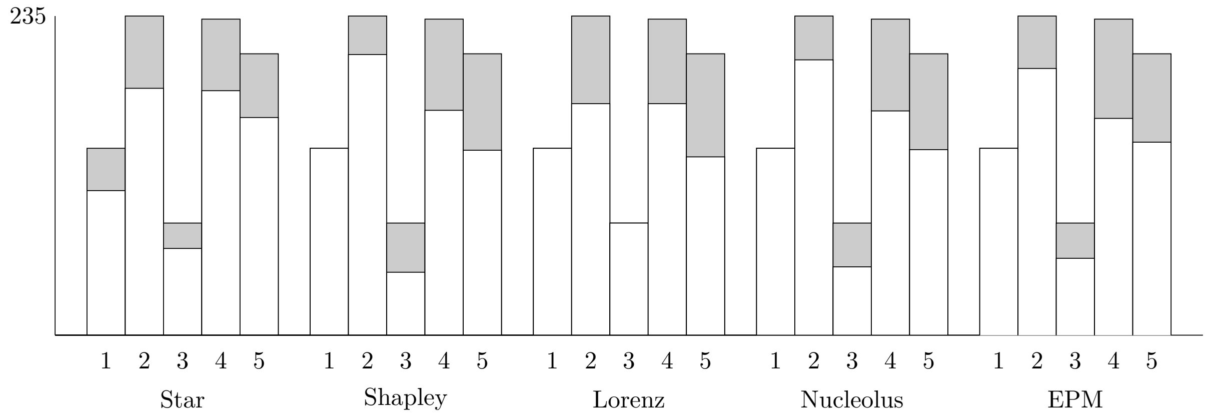


Figure 7.7: Allocated costs for the player in instance C2. The white area indicates the allocated cost whereas the white and grey area combined indicate the stand-alone cost of each player.

7.2 Allocation methods

In this section the five different allocation methods are compared and discussed. To this end, the different characteristics of the methods are judged using the results obtained on the generated instances. Each instance yields up to 12 player allocations per allocation method. This data has been used to observe the consistency of each method and to observe how each method spreads the profit over the customers.

First, the stability of the methods is discussed. Thereafter, the consistency of the different methods is observed and the savings of an average player are visualised for each of the allocation methods. Finally, the computation time required to calculate each of the allocation methods is compared.

7.2.1 Stability

The stability of an allocation method indicates if the method is guaranteed to yield an allocation in the core, if the core exists. Table 7.2 provides an overview of the amount of instances which have non-empty cores. Besides this, the table also indicates the amount of times an allocation method yielded an allocation in the core. Of all instances about 20% had an empty core. It should be noted that instances of type A and type G often have an empty core. One might use the reasoning as presented for the example of instance A to explain this assertion. For each instance, the total demand of all players in the instance is likely to be close to a multiple of the capacity of the vehicles. This, as a result, causes 'abandoned' players in the optimal solution whereas these players could potentially benefit from cooperating with other coalitions.

For instances of the joint network vehicle routing game with multiple non-clustered locations assigned to each player the core is rarely empty. This shows that a lot can be gained in cooperative transport between logistics service providers for everyone involved.

Table 7.2: An overview of the the amount of allocation in the core for each of the different types of instances.

Instance	Core		Allocation				
type	Empty	Non-empty	Star	Shapley	Lorenz	EPM	Nucleolus
A	24	26	0	0	26	26	26
B	2	48	22	37	48	48	48
C	3	47	1	7	47	47	47
D	10	40	1	8	40	40	40
E	11	9	0	0	9	9	9
F	0	50	26	45	50	50	50
G	6	4	0	0	4	4	4

The Lorenz method, Equal Profit Method and Nucleolus guarantee an allocation in the core if the core exists, the Shapley and Star allocation do not have this property. As such, these allocations do not necessarily need to be in the core, even if it exists. This assertion is supported by the results as presented in Table 7.2. It can be seen that for instances of type A, E and G the Star and Shapley allocations are never inside the core. On average the Star and Shapley allocation yielded respectively 17.9% and 34.6% allocations inside the core. Furthermore, the other stable methods perform as expected: always yielding a solution in the core if it is non-empty. It should also be noted that the Lorenz and Equal Profit Method do not yield any allocation if the core is empty, as an allocation is desirable this can be seen as a downside of the methods. Naber et al. [34] propose a modification to both methods to ensure that they always yield an allocation. They assign the Nucleolus in the case of an empty core. However, to accurately judge the consistency of each method, this adjustment has not been utilised in this thesis.

7.2.2 Consistency

In the following, the consistency of the allocation methods is discussed. To this end a regression analysis has been performed on the allocated cost versus the factors as presented in Section 6.2. The results of the linear regression are presented in Table 7.3.

Table 7.3: The results of the linear regression analysis. Both the coefficient and the p-values are presented for each allocation method.

	Star		Shapley		Lorenz		EPM		Nucleolus	
Explanatory variable	coeff.	p	coeff.	p	coeff.	p	coeff.	p	coeff.	p
Intercept	56.03	0.000	2.68	0.580	59.05	0.000	41.78	0.000	-3.98	0.414
Average distance depot	0.97	0.000	0.99	0.000	0.49	0.000	0.78	0.000	0.82	0.000
Average distance others	-0.94	0.000	-0.51	0.000	-0.59	0.000	-0.69	0.000	-0.32	0.000
Average distance own	0.16	0.000	0.19	0.000	0.087	0.015	0.17	0.000	0.17	0.000
Total demand	1.76	0.000	2.32	0.000	1.93	0.000	2.03	0.000	2.45	0.000
R-squared	0.61		0.67		0.66		0.62		0.68	

All methods show that the allocated cost and the four discussed factors are related. For all methods it holds that an increase in average distance to the depot yields an increase in allocated costs. This is to be expected as a larger distance to the depot also results in a higher stand-alone cost. A more surprising result is that a decrease in distance to other players' locations tends to increase the allocated cost. One would expect the total optimal costs to decrease as the average distance to other location decreases. However, one could also argue that as the distance to other players their locations increases the player stands to gain most from cooperation. As the player introduces a large reduction in the total costs, he receives the benefits belonging to this cost reduction. This could be seen as a flaw in the allocation methods as this classifies as unwanted behaviour. Unfortunately, the p-values indicate that all methods show this behaviour. The effect is the least strong for the nucleolus.

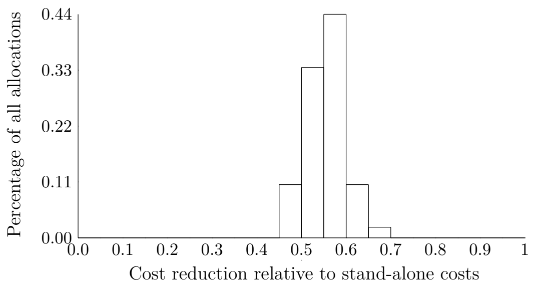
The average distance among the locations of a player also influences the allocated costs. It can be seen that as the average distance among the locations of a player increases, the allocated costs increase as well. This is to be expected as the increase in average distance is also likely to yield an increase in stand-alone costs. Lastly, it can be seen that the total demand of a player is positively related to the allocated cost. If the total demand of a player increases, his allocated costs are expected to increase as well. This does not come as a surprise since an increased total demand is also likely to increase the stand-alone cost and to reduce the flexibility of a player for cooperative transport.

The relatively high R-squared values indicate that the explanatory variables provide a lot of information about the final allocation. This means that most of the variation in the allocated cost can be attributed to differences in the five presented allocation methods. Besides this, most p-values are small, indicating that the allocated costs are dependent on the present factors. This shows that all allocation methods considered in this thesis are somewhat consistent. Furthermore, this also suggests that small changes to these parameters do not drastically alter the allocated costs for a logistics service provider, which is key in stimulating cooperative transport.

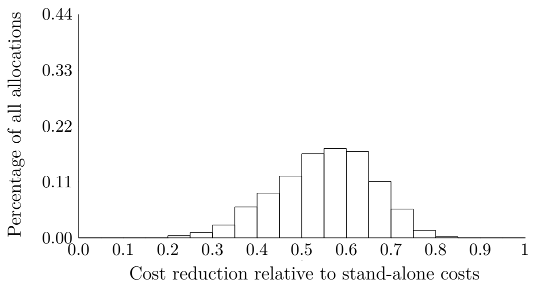
7.2.3 Achieved savings

In this section, the achieved average savings for each type of instance and allocation method are discussed. For instances D and E the results are presented in respectively Figures 7.8 and 7.9. The results for the other types of instances can be found in Appendix E.

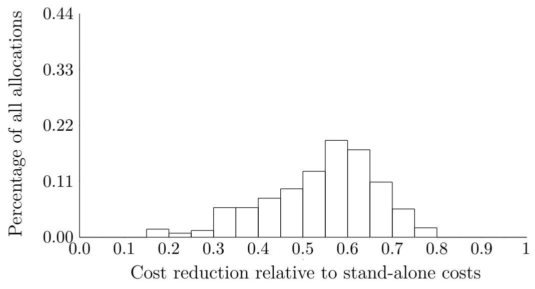
From the general behaviour in the graphs it can be observed that the average savings for each player was significantly higher for instances D compared to instance E. On average each player in instance D received a 55.3 percentage cost reduction compared to a 38.4 percentage cost reduction for players in instances E. Once more, this is related to the fact that the initial stand-alone routes for clustered instances are better compared to the initial stand-alone routes for non-clustered instances. This assertion is supported by the results for the non-clustered instances B (43.8%) and clustered instances C (16.7%).



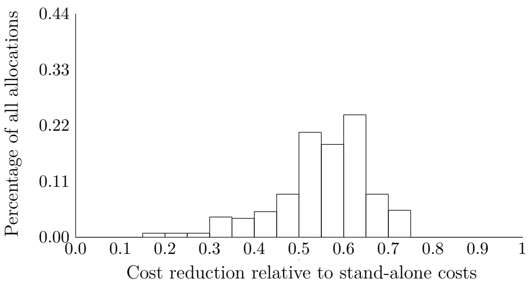
(a) Star



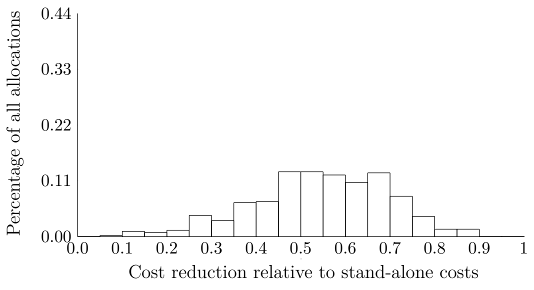
(b) Shapley



(c) Lorenz



(d) Equal Profit Method

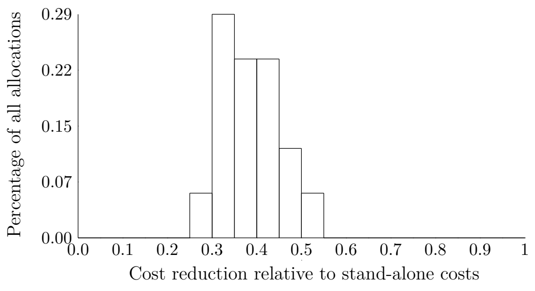


(e) Nucleolus

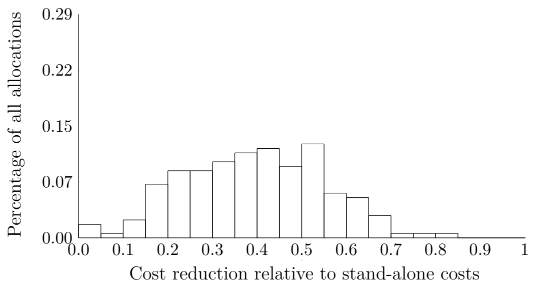
Figure 7.8: Percentage savings for instances of type D. The average cost reduction is 55.3%.

For all instances the Star method provides a small range in which most percentages of the cost-reductions fall. Simply put, for each instance, the total percentage of the cost reduction relative to the sum of the stand-alone cost equals the relative cost-reduction of each player. This creates narrow graphs with high peaks centred around the average cost reduction.

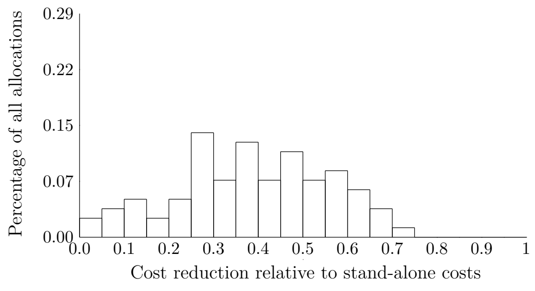
For instances of type D, the Shapley, Lorenz, EPM and Nucleolus allocations all show similar behaviour. A broad spectrum in relative savings with its peak around the average savings is visible. The most noticeable difference is that the graph belonging to the EPM allocation is less broad than compared to the Shapley, Lorenz and Nucleolus allocations. It can be seen that the method tries to provide equal relative cost reductions for each of the customers and as such, the graph has similar peaks as the graph of the star allocations.



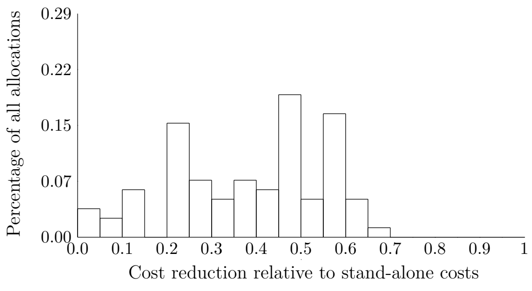
(a) Star



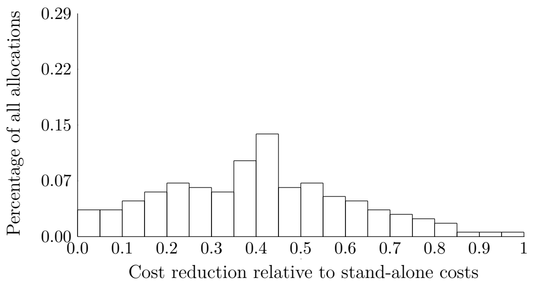
(b) Shapley



(c) Lorenz



(d) Equal Profit Method



(e) Nucleolus

Figure 7.9: Percentage savings for instances of type E. The average cost reduction is 38.4%.

For instances of type E, the star method performs as expected as can be seen by the narrow graph with high peaks. However, the spread in average cost allocations over the instances of type E is slightly larger compared to the spread seen for instances of type D. For the other allocation methods the spread in average cost reduction is fairly large. As an example, the Nucleolus provides cost reduction in the full spectrum compared to the 25 to 55 percent range of the star allocation.

7.2.4 Computational performance of the methods

In Table 7.6 the computational time needed to determine the optimal cost of all required coalitions is denoted. For the Star method, the optimal costs of the required coalitions were determined independently using the branch-and-price approach. Besides this, each of the allocation methods also requires some time to solve. The average time required to solve each of the allocation methods, based on the optimal costs of all coalitions, is denoted in Table 7.4

Table 7.4: An overview of the computational performance of the algorithms for the different allocation methods on the different types of instances. For each type of instance and algorithm, the average time to determine the cost allocation is given in seconds. Furthermore, the last two columns denote the time (in seconds) required to determine all prerequisite costs. All computations have been performed on an Intel core i5 2500K @ 3.96 GHz.

Instance			Allocation			All Costs	
Type	Star	Shapley	Lorenz	EPM	Nucleolus	Star	Others
A	$3.19 \cdot 10^{-5}$	$2.34 \cdot 10^{-3}$	$7.43 \cdot 10^{-3}$	$6.52 \cdot 10^{-3}$	0.751	0.21	11.68
B	$7.96 \cdot 10^{-6}$	$9.59 \cdot 10^{-5}$	$9.14 \cdot 10^{-4}$	$6.85 \cdot 10^{-4}$	$3.44 \cdot 10^{-3}$	1.40	4.01
C	$8.39 \cdot 10^{-6}$	$1.53 \cdot 10^{-4}$	$9.90 \cdot 10^{-4}$	$6.47 \cdot 10^{-4}$	$3.53 \cdot 10^{-3}$	3.31	15.92
D	$1.17 \cdot 10^{-5}$	$2.76 \cdot 10^{-3}$	$7.06 \cdot 10^{-3}$	$6.98 \cdot 10^{-3}$	0.796	10.97	320.28
E	$1.20 \cdot 10^{-5}$	$5.07 \cdot 10^{-3}$	$1.08 \cdot 10^{-2}$	$7.76 \cdot 10^{-3}$	0.924	62.45	1094.29
F	$5.54 \cdot 10^{-6}$	$8.65 \cdot 10^{-5}$	$8.61 \cdot 10^{-4}$	$6.62 \cdot 10^{-4}$	$3.81 \cdot 10^{-3}$	64.48	110.32
G	$3.16 \cdot 10^{-5}$	$8.18 \cdot 10^{-3}$	$3.06 \cdot 10^{-2}$	$2.99 \cdot 10^{-2}$	31.9	0.17	118.73

It can be seen that the time to compute each cost allocation is in most cases negligible compared to the time it takes to calculate the prerequisite information. The only exception to this is the Nucleolus. It can be seen that for instances with larger players, such as instances of type G, the time required to calculate the Nucleolus is no longer negligible. For instances of type G an average of 31.9 seconds was required to determine the Nucleolus.

Furthermore, the time to calculate the prerequisite information for the Star allocation is significantly smaller than the time required to calculate this information for the other allocations. This is due to the amount of vehicle routing problems that have to be solved. This makes the Star method a viable option for instances with a high amount of players and locations.

One could argue that utilising the Shapley value causes an increase in computation time as the method requires the optimal costs of all coalitions. In comparison, the Star value requires only a small amount of all optimal costs, of which most are easy to calculate. Besides this, the Lorenz method, Equal Profit Method and Nucleolus can be determined using constraint generation. As such it is advisable not to utilise the Shapley value for larger instances of the joint network vehicle routing game.

7.3 Computational performance of the algorithms

In this section the computational performance of the algorithms is discussed. Each of the algorithms have been discussed in Section 5.2. The basic (`BSC`) and advanced (`ADV`) algorithms are presented in Subsection 5.2.1. Furthermore, four constraint generation approaches have been introduced in Subsection 5.2.2. These four are the constraint generation approaches with the use of lower bounds (`CGL`), joint price-collecting vehicle routing problem (`CGP`), both the lower bounds and a relaxation of the joint price-collecting problem (`MIX`) and the constraint generation using column generation approach denoted as (`CGC`).

The numerical results have been summarised in Table 7.6. The results represent the time required to determine the optimal costs for all coalitions such that a solution in the core can be guaranteed if it exists. This means that the basic and advanced algorithm do calculate the optimal cost for all coalitions whereas this does not necessarily hold for the other algorithms. Note that each branch-and-price procedure was terminated after 900 seconds, whereas the entire procedure to determine all required costs was terminated after 3600 seconds. An overview of the amount of games that had to be terminated is presented in Table 7.5.

Table 7.5 shows that the `CGC` algorithm performed very unreliable on instances of type C, D, E and F. This can be accounted for by the fact that the set partitioning formulation of the price-collecting problem provides bad lower bounds, which cause the branch-and-bound procedure to be less efficient. The optimality gap for this formulation turned out to be significantly worse than the optimality gap for the set partitioning formulation for the vehicle routing problem. This issue becomes more apparent as the difficulty of the instances increase as can be seen from the amount of terminated instances.

Table 7.5: An overview of the amount of instances of each type that were fully terminated for each algorithm.

Instance	Allocation					
type	`BSC`	`ADV`	`CGL`	`CGP`	`MIX`	`CGC`
A	0	0	0	0	0	0
B	0	0	0	0	0	1
C	0	0	0	0	0	16
D	2	0	2	1	0	22
E	3	1	5	2	1	12
F	0	0	1	1	0	37
G	0	0	0	0	0	0

From these results it is apparent that especially on the clustered instances more games had to be terminated. Besides this, a positive correlation exists between the amount of terminated games and amount of players.

Table 7.6: An overview of the computational performance of the algorithms on the different types of instances. For each type of instance and Algorithm, the minimum, maximum and average time to solve an instance are given in seconds. A * means that instances that have not been solved in time have been omitted for the calculation of the average computation times. In brackets are the amount of instances that have been excluded because they were terminated. All computations have been performed on an Intel core i5 2500K @ 3.96 GHz.

Instance type		Algorithm											
		`BSC`		`ADV`		`CGL`		`CGP`		`MIX`		`CGC`	
	min	4.05		3.71		2.61		0.24		0.24		0.04	
A	*max*	101.70		93.13		32.47		10.29		25.13		4.23	
	average	11.68		10.56		4.51		1.57		4.41		**0.38**	
	min	0.58		0.52		0.16		0.32		0.10		0.04	
B	*max*	45.96		47.99		30.89		31.87		33.31		900.00	
	*average**	4.01		3.47		**1.84**		3.81		2.90		10.44	(1)
	min	0.33		0.25		0.25		0.68		1.14		0.05	
C	*max*	114.63		104.35		117.50		96.33		123.84		3600.00	
	*average**	15.92		**14.20**		15.18		22.25		17.49		50.22	(16)
	min	88.24		65.96		9.58		13.43		40.10		0.34	
D	*max*	3600.00		1408.12		3600.00		3600.00		1079.28		3600.00	
	*average**	320.28	(2)	255.01		**66.70**	(2)	307.41	(1)	213.86		138.22	(22)
	min	351.78		125.06		17.16		44.39		22.20		1.15	
E	*max*	3600.00		3600.00		3600.00		3600.00		3600.00		3600.00	
	*average**	1094.29	(3)	734.56	(1)	417.35	(5)	503.15	(2)	366.46	(1)	**131.37**	(12)
	min	6.60		7.34		0.65		15.25		4.81		2.79	
F	*max*	1199.40		1300.81		3600.00		3600.00		1234.62		3600.00	
	*average**	110.32		100.27		**73.83**	(1)	418.40	(1)	102.69		570.72	(37)
	min	76.35		68.30		15.60		0.09		0.59		0.07	
G	*max*	175.34		160.09		73.50		17.02		210.63		1.08	
	average	118.73		107.47		22.72		7.42		32.36		**0.27**	

From the table it is apparent that overall, the basic algorithm performs the worst. The only exception to this is for instances of type C. For instances of this type, the basic algorithm provides a very comparable performance to the advanced algorithm, only performing 12.1% worse on average. Furthermore, for instances of type C both constraint generation approaches utilising the flow formulation of the pricing-problem perform significantly worse than the basic algorithm. Besides this, it should be noted that for instances of type C the constraint generation using lower bounds performs 6.9% worse on average compared to the advanced algorithm. In comparison with the results for the other instance types, the constraint generating algorithms perform worse than expected. This can be accounted for by the fact that instances of type C only have 5 players, causing the constraint generation approach to be utilised on only 26 constraints. As such, finding the violated constraints may in this case be more computationally intensive than actually calculating all optimal costs.

This relation is also visible in the other instances, whereas the constraint generation approaches perform very well, they perform best in instances with more players. In instances A and G, all constraint generation approaches were at least a factor 2 faster than the other algorithms. In these instances the constraint generation using column generation also showed very promising results, outperforming all other methods by a large margin.

For instances D and E the performance gains of the constraint generation approaches becomes more noticeable. Compared to instances B and C not 32 but 1024 coalitions have to be considered. This means that there is a huge potential for constraint generating algorithms significantly reduce the amount of computational effort required. This effect is clearly seen as the constraint generation using lower bounds outperforms all other algorithms by about a factor 3.

However, looking at instances D, it can be seen that the constraint generation using lower bounds does not prove to be the most reliable method. Only the advanced and mixed approach were able to solve all instances within the set time limits. Besides this, it can be seen that both constraint generation approaches using the flow formulation of the pricing problem perform worse compared to the constraint generation using lower bounds. This shows that the flow formulation is less efficient in solving these types of instances. For instances of type E it can be seen that, in contrast to the results for instances of type D, the mixed approach does perform very well. The method solved the most instances with the lowest average time. No clear explanation for this results is known. For instances of type E, the `CGC` algorithm proved to be the fasted algorithm. However, it also proved to be the least reliable algorithm.

Another interesting assertion is that players with clustered locations already have better optimised transportation when compared to players without clustered locations. This suggests that these players would benefit less from cooperation and this could translate into a smaller core for the corresponding joint network vehicle routing game. A smaller core, in turn, could result in having to generate more constraints, penalising the performance of the constraint generating algorithms even further. For instances A, B, D and F, the locations of each players are not clustered. In all of these instances the constraint generation approach using lower bounds performs exceptionally well when compared to the basic and advanced algorithms. This supports the statement that in clustered instances the core might be more tight and as such, more constraints have to be generated compared to non-clustered instances.

In general the advanced algorithm outperforms the basic algorithm. This is to be expected as the advanced algorithm utilises a better initialisation without requiring any significant computation time. The advanced algorithm yields an increase between 9.1% and 32.87% in average performance over the basic algorithm. As both the basic and advanced algorithm have to solve the same amount of problems, any performance gains are generated in the branch-and-price procedure. Besides this, the advanced algorithm is generally outperformed by both the constraint generation using lower bounds and mixed approach. From the results it can be seen that the constraint generation using lower bounds shows the most promising results. However, these results do not show how often each method was the fastest. In this case, worse performance in some instances could heavily influence the averages. As such, an overview of the amount of time that each algorithm was the fastest on an instance is provided in Table 7.7.

Table 7.7: An overview of the amount of times an algorithm was the fastest on a type of instances.

Instance	Allocation					
type	`BSC`	`ADV`	`CGL`	`CGP`	`MIX`	`CGC`
A	0	0	0	2	2	46
B	0	2	22	1	0	25
C	1	9	8	8	1	23
D	0	0	21	8	0	21
E	0	0	4	5	3	7
F	2	4	44	0	0	0
G	0	0	0	0	0	10

These results show that in most cases the `CGL` and `CGC` algorithms outperform the other algorithms. For instances of type A, C and G the exact constraint generation using column generation for the joint price-collecting problem formulation (`CGC`) performs the best. This can be attributed to the efficient manner in which each of the sub-problems is solved using column generation. However, this method does not perform well on any other instances of the joint network vehicle routing game. For these instances, the constraint generation using lower bounds performs best, taking into account the reliability of the `CGC` algorithm.

8 Concluding remarks

In the previous chapter, a lot of results have been presented and discussed. The aim of this chapter is to provide a comprehensive overview of the most important features of the joint network vehicle routing game and its solution procedures. To this end, a short overview of the properties of the new game is provided. Next, the numerical results are summarised. At the end of this chapter some suggestions for future research are provided.

The newly joint network vehicle routing game has been shown to be a generalisation of the vehicle routing game as well as the travelling salesman game. Furthermore, it has been proven that the joint network-vehicle routing game is monotonic, sub-additive and under some simplistic assumptions even essential.

In order to observe the performance of the game, instances have been randomly generated. First, instances of the vehicle routing game are considered. The core was empty for about 50% of the generated vehicle routing game instances. These instances were shown to be very suitable for collaborative transport. An average cost reduction of 54% is given to each player. As these instances consist of 10 to 12 players they turned out to be a good match for the constraint generating approaches. It has been shown that these approaches outperform the enumerative approaches on average by at least a factor of 2. Furthermore, the algorithm combining constraint generation with column generation on the joint price-collecting vehicle routing problem showed very promising results. It proved to be the fastest algorithm in 93% of the instances of the vehicle routing game. The algorithm was significantly faster than any other algorithms on these instances. Unfortunately, the algorithm did not perform well for any of the other instances.

The other instances of the joint network vehicle routing game showed similar behaviour. For the non-clustered instances the highest potential savings were achieved. The non-clustered instances yielded an average cost reduction of 28% versus 46% for the clustered instances. This difference can be attributed to the fact that the optimal costs in both cases are comparable while the stand-alone costs are significantly higher for players with non-clustered locations. Either way, substantial savings can be achieved in all cases. It was shown that for more than five players, the constraint generating algorithms performed better compared to the non-constraint generating algorithms. This can be accounted for by the exponential increase of coalitions in the number of players. For most instances the `CGC` algorithm performed the fastest. However, the algorithm also was the least reliable. It was unable to solve 132 out of the total 280 instances. The advanced and `MIX` algorithms proved to be the most reliable. Both methods were unable to solve only one instance within the time limits.

The allocation methods have also been compared. All methods showed to be highly related to the presented dependent variables. Any difference in behaviour among the methods did not surface from the results of the linear regression. Differences do exist for the stability of the methods. On average the Star and Shapley allocation yielded respectively 17.9% and 34.6% allocations inside the core. The other methods, being stable, always produced an allocation in the core if it was non-empty.

8.1 Recommendations for future research

The current implementation of the joint network vehicle routing game utilises the capacitated vehicle routing problem. For future research it might be interesting to consider the vehicle routing problem with time-windows. Not only is this problem more general, but the problem also limits the space of feasible routes. As such, the label setting algorithm is expected to generate less routes and as such use significantly less computation time. This could speed up the overall computation time of the algorithms.

In the preliminary results it was shown that the basic algorithm did not benefit from the savings construction heuristic. This may be due to the fact that sorting all arcs is an exponential procedure. As such, one or more executions of the pricing problem tend to be faster in generating profitable routes than executing the savings algorithm to generate a better initial solution. It may be beneficial to search for potentially faster and more advanced algorithms. Furthermore, Golden et al. [24] discuss the viability of meta-heuristics. An example of such an algorithm is the heuristic proposed by Pisinger and Ropke [38]. The results shown by this meta-heuristic are very promising and might be worth taking into consideration as part of a solution procedure for the joint network vehicle routing game.

The largest advantage of the set partitioning formulation is the tight bounds it provides for the capacitated vehicle routing problem. The same cannot be said of the flow formulation of the vehicle routing problem, the formulation is known for the bad bounds it provides. However, recent studies show [24] that the gap can be tightened. This study shows that the flow formulation as proposed by Yaman [49] yields an average optimal gap of 2.48% versus the branch-and-price approach which yielded an average optimal gap of 1.62%. The approach by Yaman [49] was partially implemented in this thesis, however, more advanced formulations may exist. Such formulations could significantly increase the performance of the proposed joint price-collecting algorithms. One should carefully weigh the benefit of a reduced gap versus the potential increase in computation time.

Besides this, the comparison between the solution algorithms for the joint network vehicle routing game may not have been entirely fair. Whereas CPLEX is optimised for multiple processor cores, the implemented branch-and-bound procedure was not. This procedure relied heavily on serial calculations, which resulted in not fully utilising the CPU. For future research it might be a good idea to optimise the label setting algorithm by Range [40] for multi-core processing. This could easily be implemented by letting each core handle the extension of labels from a node for a certain label cardinality. These processes can be computed independently and do no necessarily interfere. However, one would have to restructure the procedure to remove any dominated label. This process can not be performed in a parallel fashion.

The Advanced algorithm showed a performance increase over the basic algorithm. This can solely be attributed to the initialisation of the routes as that is the only difference between the algorithms. Besides the currently proposed algorithm, many ways of initialising the routes for each coalition exist. It might be interesting to investigate the effect of not just simply initialising the larger problems with the optimal solution to the smaller problems. For example, one could insert any non-considered locations into the optimal solutions of the smaller instances. This may create better initialisations for the vehicle routing problem. Besides this, one could also first solve the largest problem and initialise the smaller problems with the optimal solution of the larger problem. One would simple have to remove any location from the optimal routes that are no longer present in the smaller coalition. In a similar fashion many different initialisations can be determined.

As promising results for the constraint generation of the Equal Profit Method were found, it might be beneficial to develop a similar procedure for the Nucleolus. For the Nucleolus the same constraints can be generated as was done for the Equal Profit Method. However, the Nucleolus requires each of the constraints to be generated. Future research may investigate whether the Nucleolus can be solved without generating each and every constraint. It should be noted that the proposed procedure can easily be adapted to the Lorenz method due to the similarities in constraints or to a method to determine whether the core is non-empty.

The constraint generation method using the branch-and-price procedures shows a significantly better computational performance on instances of the vehicle routing game compared to the other proposed algorithms. Unfortunately, something similar can not be said for most instances of the joint network vehicle routing game. For instances in which players do have more than one location, the method was outperformed by all other algorithms. Looking somewhat deeper into the algorithm it was shown that the lower bounds of the set partitioning formulation of the joint price-collecting vehicle routing problem were significantly worse than the bounds of the set partitioning formulation of the vehicle routing problem. This causes the tree within the branch-and-price procedure to become larger as less nodes can be pruned. This in turn causes an increase in computational performance required to solve the joint price-collecting problem. For future research one may want to include the joint price-collecting problem in a branch and price and cut procedure. An example of cuts that can be utilised are the capacity cuts a proposed for the flow formulation of the joint price-collecting problem. Besides this, other beneficial cuts may exist to decrease the optimality gap and improve the overall performance.

Finally, to better observe the applicability of the game, one might want to vary the types of instances. Right now, each player has the exact same amount of locations. This might not be a reflection of reality. One could, for example, randomly generate the amount of locations for each player. The results obtained by such experiments could give additional insight into the joint network vehicle routing game.

References

[1] IBM ILOG CPLEX Optimizer 12.7.0, 2016. URL https://www-01.ibm.com/software/commerce/optimization/cplex-optimizer/.

[2] J. Arin. Egalitarian distributions in coalitional models: The lorenz criterion. IKERLANAK 2003-02, Universidad del País Vasco - Departamento de Fundamentos del Análisis Económico I, 2003. URL http://EconPapers.repec.org/RePEc:ehu:ikerla:200302.

[3] M. L. Balinski and R. E. Quandt. On an integer program for a delivery problem. *Operations Research*, 12(2):300–304, 1964. ISSN 0030364X, 15265463. URL http://www.jstor.org/stable/167930.

[4] K. Braekers, K. Ramaekers, and I. V. Nieuwenhuyse. The vehicle routing problem: State of the art classification and review. *Computers & Industrial Engineering*, 99:300 – 313, 2016. ISSN 0360-8352. doi: http://doi.org/10.1016/j.cie.2015.12.007.

[5] R. Branzei, D. Dimitrov, and S. Tijs. *Models in Cooperative Game Theory*. Lecture notes in economics and mathematical systems. Springer Berlin Heidelberg, 2008. ISBN 9783540779544. URL https://books.google.nl/books?id=TywLRIiEvSoC.

[6] S. E. Butt and D. M. Ryan. An optimal solution procedure for the multiple tour maximum collection problem using column generation. *Computers & Operations Research*, 26(4):427 – 441, 1999. ISSN 0305-0548. doi: http://dx.doi.org/10.1016/S0305-0548(98)00071-9. URL http://www.sciencedirect.com/science/article/pii/S0305054898000719.

[7] A. Chabrier. Vehicle routing problem with elementary shortest path based column generation. *Computers & Operations Research*, 33(10):2972 – 2990, 2006. ISSN 0305-0548. doi: https://doi.org/10.1016/j.cor.2005.02.029. URL http://www.sciencedirect.com/science/article/pii/S0305054805000857. Part Special Issue: Constraint Programming.

[8] G. Clarke and J. W. Wright. Scheduling of vehicles from a central depot to a number of delivery points. *Operations Research*, 12(4):568–581, 1964. ISSN 0030364X, 15265463. URL http://www.jstor.org/stable/167703.

[9] G. B. Dantzig and J. H. Ramser. The truck dispatching problem. *Management Science*, 6(1):80–91, 1959. ISSN 00251909, 15265501. URL http://www.jstor.org/stable/2627477.

[10] G. Desaulniers, J. Desrosiers, and M. Solomon. *Column Generation*. Springer US, 2005. ISBN 978-0-387-25486-9.

[11] M. Desrochers, J. Desrosiers, and M. Solomon. A new optimization algorithm for the vehicle routing problem with time windows. *Oper. Res.*, 40(2):342–354, Mar. 1992. ISSN 0030-364X. doi: 10.1287/opre.40.2.342. URL http://dx.doi.org/10.1287/opre.40.2.342.

[12] S. Engevall, M. Göthe-Lundgren, and P. Värbrand. The traveling salesman game: An application ofcost allocation in a gas and oil company. *Annals of Operations Research*, 82 (0):203–218, 1998. ISSN 1572-9338. doi: 10.1023/A:1018935324969.

[13] S. Engevall, M. Göthe-Lundgren, and P. Värbrand. The heterogeneous vehicle-routing game. *Transportation Science*, 38(1):71–85, 2004. doi: 10.1287/trsc.1030.0035.

[14] D. Feillet. A tutorial on column generation and branch-and-price for vehicle routing problems. *4OR*, 8(4):407–424, 2010. ISSN 1614-2411. doi: 10.1007/s10288-010-0130-z. URL http://dx.doi.org/10.1007/s10288-010-0130-z.

[15] D. Feillet, P. Dejax, M. Gendreau, and C. Gueguen. An exact algorithm for the elementary shortest path problem with resource constraints: Application to some vehicle routing problems. *Networks*, 44(3):216–229, 2004. ISSN 1097-0037. doi: 10.1002/net.20033. URL `http://dx.doi.org/10.1002/net.20033`.

[16] D. Feillet, P. Dejax, and M. Gendreau. Traveling salesman problems with profits. *Transportation Science*, 39(2):188–205, May 2005. ISSN 1526-5447. doi: 10.1287/trsc.1030.0079. URL `http://dx.doi.org/10.1287/trsc.1030.0079`.

[17] P. C. Fishburn and H. O. Pollak. Fixed-route cost allocation. *The American Mathematical Monthly*, 90(6):366–378, 1983. ISSN 00029890, 19300972. URL `http://www.jstor.org/stable/2975572`.

[18] M. Frisk, M. Göthe-Lundgren, K. Jörnsten, and M. Rönnqvist. Cost allocation in collaborative forest transportation. *European Journal of Operational Research*, 205(2):448 – 458, 2010. ISSN 0377-2217. doi: http://dx.doi.org/10.1016/j.ejor.2010.01.015.

[19] W. W. Garvin, H. W. Crandall, J. B. John, and R. A. Spellman. Applications of linear programming in the oil industry. *Management Science*, 3(4):407–430, 1957. ISSN 00251909, 15265501. URL `http://www.jstor.org/stable/2627037`.

[20] B. Gavish and S. Graves. Scheduling and routing in transportation and distribution systems: Formulations and new relaxations. IKERLANAK 8202, University of Rochester, 1981. URL `http://hdl.handle.net/1802/4883`.

[21] F. Gheysens, B. Golden, and A. Assad. A comparison of techniques for solving the fleet size and mix vehicle routing problem. *Operations-Research-Spektrum*, 6(4):207–216, 1984. ISSN 1436-6304. doi: 10.1007/BF01720070. URL `http://dx.doi.org/10.1007/BF01720070`.

[22] D. B. Gillies. Solutions to general non-zero-sum games. (40):47–85, 1959.

[23] B. Golden, A. Assad, L. Levy, and F. Gheysens. The fleet size and mix vehicle routing problem. *Computers & Operations Research*, 11(1):49 – 66, 1984. ISSN 0305-0548. doi: http://dx.doi.org/10.1016/0305-0548(84)90007-8. URL `http://www.sciencedirect.com/science/article/pii/0305054884900078`.

[24] B. L. Golden, S. Raghavan, and E. A. Wasil. *The Vehicle Routing Problem: Latest Advances and New Challenges*. Springer US, 2008. ISBN 978-0-387-77777-1.

[25] M. Göthe-Lundgren, K. Jörnsten, and P. Värbrand. On the nucleolus of the basic vehicle routing game. *Mathematical Programming*, 72(1):83–100, 1996. ISSN 1436-4646. doi: 10.1007/BF02592333.

[26] J. C. Harsanyi. A general theory of rational behavior in game situations. *Econometrica*, 34(3):613–634, 1966. ISSN 00129682, 14680262. URL `http://www.jstor.org/stable/1909772`.

[27] O. G. Haywood. Military decision and game theory. *Journal of the Operations Research Society of America*, 2(4):365–385, 1954. ISSN 00963984. URL `http://www.jstor.org/stable/166693`.

[28] J. Kuipers. A note on the 5-person traveling salesman game. *Zeitschrift für Operations Research*, 38(2):131–139, 1993. ISSN 1432-5217. doi: 10.1007/BF01414209. URL `http://dx.doi.org/10.1007/BF01414209`.

[29] J. K. Lenstra and A. H. G. R. Kan. Complexity of vehicle routing and scheduling problems. *Networks*, 11(2):221–227, 1981. ISSN 1097-0037. doi: 10.1002/net.3230110211. URL `http://dx.doi.org/10.1002/net.3230110211`.

[30] F. Li, B. Golden, and E. Wasil. A record-to-record travel algorithm for solving the heterogeneous fleet vehicle routing problem. *Computers & Operations Research*, 34(9): 2734 – 2742, 2007. ISSN 0305-0548. doi: http://doi.org/10.1016/j.cor.2005.10.015. URL `http://www.sciencedirect.com/science/article/pii/S0305054805003382`.

[31] M. Maschler, B. Peleg, and L. S. Shapley. Geometric properties of the kernel, nucleolus, and related solution concepts. *Mathematics of Operations Research*, 4(4):303–338, 1979. ISSN 0364765X, 15265471. URL `http://www.jstor.org/stable/3689220`.

[32] C. E. Miller, A. W. Tucker, and R. A. Zemlin. Integer programming formulation of traveling salesman problems. *J. ACM*, 7(4):326–329, Oct. 1960. ISSN 0004-5411. doi: 10.1145/321043.321046. URL `http://doi.acm.org/10.1145/321043.321046`.

[33] R. B. Myerson. *Game theory - Analysis of Conflict.* Harvard University Press, 1997. ISBN 978-0-674-34116-6.

[34] S. Naber, D. de Ree, R. Spliet, and W. van den Heuvel. Allocating co2 emission to customers on a distribution route. *Omega*, 54:191–199, July 2015. doi: 10.1016/j.omega.2015.01.017.

[35] J. Nash. Non-cooperative games. *Annals of Mathematics*, 54(2):286–295, 1951. ISSN 0003486X. URL `http://www.jstor.org/stable/1969529`.

[36] J. V. Neumann and O. Morgenstern. *Theory of Games and Economic Behavior.* Princeton University Press, 1944. ISBN 0691119937.

[37] C. Papadimitriou and K. Steiglitz. *Combinatorial Optimization: Algorithms and Complexity.* Dover Books on Computer Science. Dover Publications, 1982. ISBN 9780486402581. URL `https://books.google.nl/books?id=u1RmDoJqkF4C`.

[38] D. Pisinger and S. Ropke. A general heuristic for vehicle routing problems. *Computers & Operations Research*, 34(8):2403 – 2435, 2007. ISSN 0305-0548. doi: http://doi.org/10.1016/j.cor.2005.09.012. URL `http://www.sciencedirect.com/science/article/pii/S0305054805003023`.

[39] J. A. M. Potters, I. J. Curiel, and S. H. Tijs. Traveling salesman games. *Mathematical Programming*, 53(1):199–211, 1992. ISSN 1436-4646. doi: 10.1007/BF01585702. URL `http://dx.doi.org/10.1007/BF01585702`.

[40] T. M. Range. Exploiting set-based structures to accelerate dynamic programming algorithms for the elementary shortest path problem with resource constraints. Discussion Papers of Business and Economics 17/2013, Department of Business and Economics, University of Southern Denmark, 2013. URL `http://EconPapers.repec.org/RePEc:hhs:sdueko:2013_017`.

[41] D. Schmeidler. The nucleolus of a characteristic function game. *SIAM Journal on Applied Mathematics*, 17(6):1163–1170, 1969.

[42] L. S. Shapley. A value for n-person games. *Contributions to the theory of games*, 2:307–317, 1953.

[43] L. S. Shapley. Cores of convex games. *International Journal of Game Theory*, 1(1):11–26, 1971. ISSN 1432-1270.

[44] M. Shubik. Incentives, decentralized control, the assignment of joint costs and internal pricing. *Management Science*, 8(3):325–343, 1962. ISSN 00251909, 15265501. URL `http://www.jstor.org/stable/2627389`.

[45] H. Stellingwerf, A. Kanellopoulos, J. van der Vorst, and J. Bloemhof. The load dependent vehicle routing problem for temperature controlled road transportation. *Under review*, 2017.

[46] A. Stenger, M. Schneider, and D. Goeke. The prize-collecting vehicle routing problem with single and multiple depots and non-linear cost. Publications of darmstadt technical university, institute for business studies (bwl), Darmstadt Technical University, Department of Business Administration, Economics and Law, Institute for Business Studies (BWL), 2013. URL `http://EconPapers.repec.org/RePEc:dar:wpaper:62372`.

[47] A. Tamir. On the core of a traveling salesman cost allocation game. *Operations Research Letters*, 8(1):31 – 34, 1989. ISSN 0167-6377. doi: http://dx.doi.org/10.1016/0167-6377(89)90030-8. URL `http://www.sciencedirect.com/science/article/pii/0167637789900308`.

[48] Y. Wang, X. Ma, Z. Li, Y. Liu, M. Xu, and Y. Wang. Profit distribution in collaborative multiple centers vehicle routing problem. *Journal of Cleaner Production*, 144:203 – 219, 2017. ISSN 0959-6526. doi: http://dx.doi.org/10.1016/j.jclepro.2017.01.001. URL `http://www.sciencedirect.com/science/article/pii/S095965261730001X`.

[49] H. Yaman. Formulations and valid inequalities for the heterogeneous vehicle routing problem. *Mathematical Programming*, 106(2):365–390, 2006. ISSN 1436-4646. doi: 10.1007/s10107-005-0611-6. URL `http://dx.doi.org/10.1007/s10107-005-0611-6`.

Appendix

A Derivations cost allocation example

Below some of the derivations belonging to Example 2.3 are presented.

Nucleolus

To determine the Nucleolus the iterative procedure is initialised with $F_1 = \emptyset$. This yields the following problem:

$$\begin{aligned}
\textbf{Objective:} \quad & z_1 = \max z \\
\\
\textbf{Subject to:} \quad & x_1 \leq 10 \\
& x_2 \leq 10 \\
& x_3 \leq 6 \\
& x_1 + z \leq 10 \\
& x_2 + z \leq 10 \\
& x_3 + z \leq 6 \\
& x_1 + x_2 + z \leq 13 \\
& x_2 + x_3 + z \leq 15 \\
& x_3 + x_1 + z \leq 15 \\
& x_1 + x_2 + x_3 = 18 \\
\\
& x_1, x_2, x_3 \geq 0 \\
& z \in \mathbb{R}
\end{aligned}$$

An optimal solution to the problem is given by $x_1 = 9, x_2 = 3.5, x_3 = 5.5$ with $z_1 = 0.5$. Now, this is the exact excess corresponding to coalition $\{3\}$,$\{1, 2\}$ and $\{1, 3\}$, note that lifting the constraint on $\{3, 1\}$ does not increase the objective value. As such, only coalitions $\{3\}$ and $\{1, 2\}$ are limiting the objective. These coalitions are added to F_2. Now the following simplified linear programming problem has to be solved:

Objective: $z_2 = \max z$

Subject to:

$$\begin{aligned}
&x_1 \leq 10\\
&x_2 \leq 10\\
&x_1 + z \leq 10\\
&x_2 + z \leq 10\\
&x_3 + 0.5 = 6\\
&x_1 + x_2 + 0.5 = 13\\
&x_2 + x_3 + z \leq 15\\
&x_3 + x_1 + z \leq 15\\
&x_1 + x_2 + x_3 = 18\\
&\\
&x_1, x_2, x_3 \geq 0\\
&z \in \mathbb{R}
\end{aligned}$$

An optimal solution to this problem is given by $x_1 = 6.25, x_2 = 6.25, x_3 = 5.5$ with $z_2 = 3.25$. This solution yields an equation in the constraints for coalitions $\{1, 3\}$ and $\{2, 3\}$ both limiting the objective value and hence added to F_3. This leads to the following linear programming problem:

Objective: $z_3 = \max z$

Subject to:

$$\begin{aligned}
&x_1 \leq 10\\
&x_2 \leq 10\\
&x_1 + z \leq 10\\
&x_2 + z \leq 10\\
&x_3 + 0.5 = 6\\
&x_1 + x_2 + 0.5 = 13\\
&x_2 + x_3 + 3.25 = 15\\
&x_3 + x_1 + 3.25 = 15\\
&x_1 + x_2 + x_3 = 18\\
&\\
&x_1, x_2, x_3 \geq 0\\
&z \in \mathbb{R}
\end{aligned}$$

After solving the problem the iterations terminate with optimal solution $x_1 = 6.25, x_2 = 6.25,\ x_3 = 5.5$ and $z_3 = 3.75$. The unique Nucleolus has been determined.

Equal profit method

In order to determine the equal profit method allocation the following linear programming problem has to be solved:

$$
\begin{aligned}
\textbf{Objective:} \quad & \min z \\
\\
\textbf{Subject to:} \quad & \frac{x_1}{10} - \frac{x_2}{10} \leq z \\
& \frac{x_1}{10} - \frac{x_3}{6} \leq z \\
& \frac{x_2}{10} - \frac{x_1}{10} \leq z \\
& \frac{x_3}{6} - \frac{x_1}{10} \leq z \\
& \frac{x_2}{10} - \frac{x_3}{6} \leq z \\
& \frac{x_3}{6} - \frac{x_2}{10} \leq z \\
& x_1 \leq 10 \\
& x_2 \leq 10 \\
& x_3 \leq 6 \\
& x_1 + x_2 \leq 13 \\
& x_2 + x_3 \leq 15 \\
& x_3 + x_1 \leq 15 \\
& x_1 + x_2 + x_3 = 18 \\
\\
& x_1, x_2, x_3 \geq 0 \\
& z \in \mathbb{R}^+
\end{aligned}
$$

An optimal solution is given by $x_1 = 6.5, x_2 = 6.5, x_3 = 5$ and $z = 0.18$. The method tries to allocate the costs relative to the stand-alone emissions. It is however limited by the coalitional rationality constraint of coalition $\{1, 2\}$. In this solution customer 1 and 2 would have an equal profit if they form a coalition together, without 3. As such, despite the solution being in the core it is unlikely that each player accepts the allocation.

Lorenz

In order to determine the optimal allocation by the Lorenz method the following linear programming problem has to be solved:

$$
\begin{array}{rl}
\textbf{Objective:} & \min z \\
\\
\textbf{Subject to:} & x_1 - x_2 \leq z \\
& x_1 - x_3 \leq z \\
& x_2 - x_1 \leq z \\
& x_3 - x_1 \leq z \\
& x_2 - x_3 \leq z \\
& x_3 - x_2 \leq z \\
& x_1 \leq 10 \\
& x_2 \leq 10 \\
& x_3 \leq 6 \\
& x_1 + x_2 \leq 13 \\
& x_2 + x_3 \leq 15 \\
& x_3 + x_1 \leq 15 \\
& x_1 + x_2 + x_3 = 18 \\
\\
& x_1, x_2, x_3 \geq 0 \\
& z \in \mathbb{R}^+
\end{array}
$$

An optimal solution is given by $x_1 = 6, x_2 = 6, x_3 = 6$ and $z = 0.18$. In contrast to the equal profit method the Lorenz method tries to distribute the costs equal without violating any coalitional rationality constraints. The solution may be in the core but is very unlikely to be accepted by customer 3.

B Example of elementary shortest path algorithm

In the following a small example of the algorithm is shown. Consider the following instance of the elementary shortest path problem:

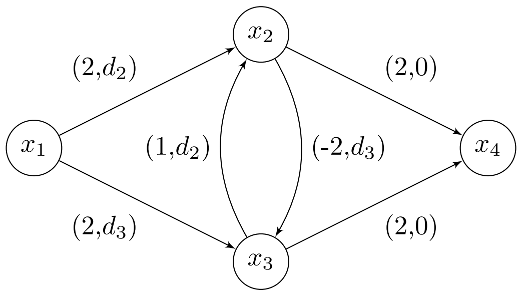


Figure 8.1: An exemplary instance of the elementary shortest path problem with resource constraints.

The example displayed in Figure 8.1 covers a total of four nodes. Node x_1 represents the origin whereas node x_4 is the destination. x_2 and x_3 each have a demand of respectively d_2 and d_3. For each arc the costs of using that arc and the demand of the endpoint of the arc is displayed between brackets. The goal is to find the minimum length path from v_1 to v_4 To each node labels are assigned during the iterative process. First the algorithm is initialised in Figure 8.2a by assigning the label $[q_i, s_i, V_1^1, V_1^2, V_1^3, V_1^4, C] = [0, 1, 1, 0, 0, 0, 0]$ to the first depot node. In Figure 8.2b node x_1 is considered, in this first iteration the label corresponding to x_1 is extended to the two successors x_2 and x_3. In the second iteration node x_2 is considered. As such, labels are added to x_3 and x_4. In the third and final iteration node x_3 is considered. Note that an additional label is not added to x_2 as the extension from x_3 to x_2 is dominated by the already existing label for x_2. Now all non-dominated paths are shown next to node x_4. It can now be concluded that the shortest path $x_1 - x_2 - x_3 - x_4$ has cost 2.

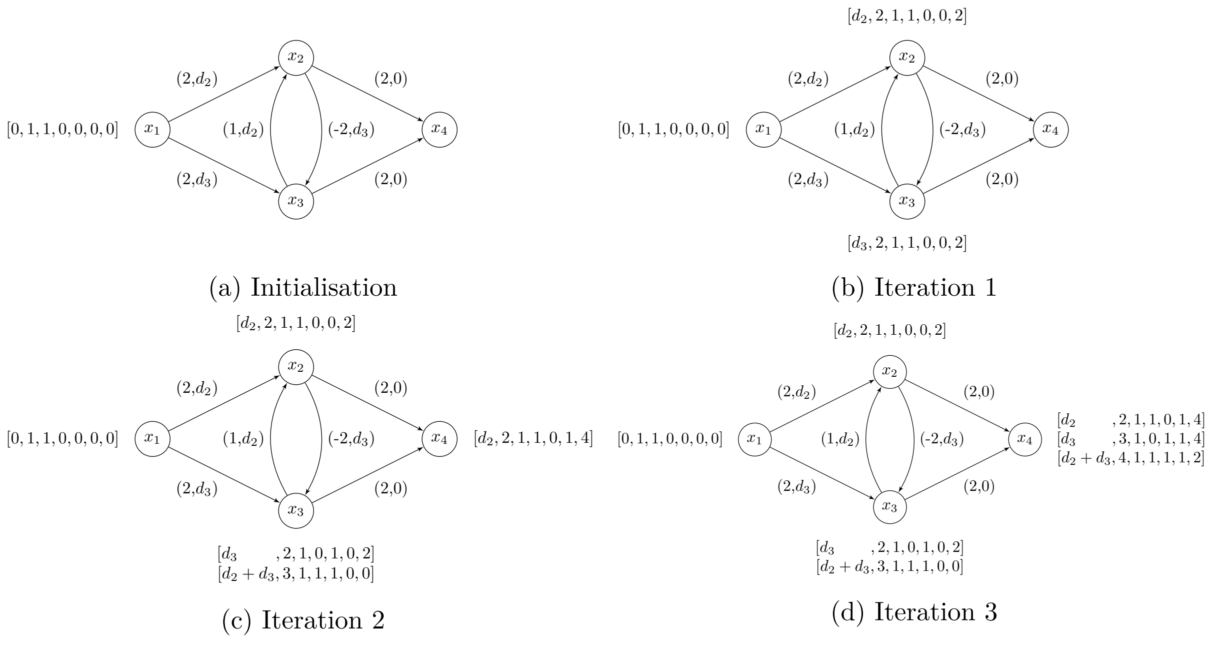

$[0, 1, 1, 0, 0, 0, 0]$
(a) Initialisation
$[d_2, 2, 1, 1, 0, 0, 2]$
$[0, 1, 1, 0, 0, 0, 0]$
$[d_3, 2, 1, 1, 0, 0, 2]$
(b) Iteration 1
$[d_2, 2, 1, 1, 0, 0, 2]$
$[0, 1, 1, 0, 0, 0, 0]$
$[d_2, 2, 1, 1, 0, 1, 4]$
$[d_3, 2, 1, 0, 1, 0, 2]$
$[d_2 + d_3, 3, 1, 1, 1, 0, 0]$
(c) Iteration 2
$[d_2, 2, 1, 1, 0, 0, 2]$
$[0, 1, 1, 0, 0, 0, 0]$
$[d_2, 2, 1, 1, 0, 1, 4]$
$[d_3, 3, 1, 0, 1, 1, 4]$
$[d_2 + d_3, 4, 1, 1, 1, 1, 2]$
$[d_3, 2, 1, 0, 1, 0, 2]$
$[d_2 + d_3, 3, 1, 1, 1, 0, 0]$
(d) Iteration 3

Figure 8.2: An exemplary instance of the elementary shortest path problem with resource constraints.

C Derivation of the upper bound to the cardinality of K'

In the following an upper bound to the cardinality of K' is derived for a coalition S. The cardinality can be expressed as follows:

$$|K'| = \left| \bigcup_{S' \in \Omega_1^S} R_{S'} \cup \bigcup_{S' \in \Omega_{|S|-1}^S} R_{S'} \right| = \left| \bigcup_{S' \in \Omega_1^S} R_{S'} \right| + \left| \bigcup_{S' \in \Omega_{|S|-1}^S} R_{S'} \right| \tag{8.1}$$

Now, using the fact that each vehicle routing problem consisting of n locations is solved to optimality with a most n routes the following inequalities follow:

$$\left| \bigcup_{S' \in \Omega_1^S} R_{S'} \right| \leq \sum_{i \in S} |V_i| \leq |S| \max_{i \in S}\{|V_i|\} \tag{8.2}$$

$$\left| \bigcup_{S' \in \Omega_{|S|-1}^S} R_{S'} \right| \leq \sum_{S' \in \Omega_{|S|-1}^S} (|S|-1) \max_{i \in S}\{|V_i|\} \leq |S|(|S|-1) \max_{i \in S}\{|V_i|\}. \tag{8.3}$$

Finally, these equations can be combined to yield the final result:

$$|K'| = \left| \bigcup_{S' \in \Omega_1^S} R_{S'} \right| + \left| \bigcup_{S' \in \Omega_{|S|-1}^S} R_{S'} \right| \leq |S| \max_{i \in S}\{|V_i|\} + |S|(|S|-1) \max_{i \in S}\{|V_i|\} = |S|^2 \max_{i \in S}\{|V_i|\}. \tag{8.4}$$

D Preliminary results

Two types of instances were used to obtain the preliminary results. An instance of P10 contains of 10 players, each with one location, whereas an instance of P15 contains 5 players, each with three locations. A summary of the used instances is given in Table 8.1.

Table 8.1: An overview of the instances used in the preliminary tests. For each instance, the sum of the demand, optimal costs and sum of stand-alone costs of the players are displayed.

Instance	$\sum_{i\in V_N} d_i$	$C(N)$	$\sum_{i\in N} C(\{i\})$
P10_1	142.74	579.20	1206.51
P10_2	148.78	644.69	1566.70
P10_3	156.36	564.21	1258.31
P10_4	172.55	687.07	1494.26
P10_5	122.23	534.98	1185.49
P10_6	166.16	646.21	1376.42
P10_7	155.37	644.13	1635.34
P10_8	143.05	623.36	1366.51
P10_9	150.06	643.28	1568.79
P10_10	71.33	504.02	1396.28
P10_11	128.87	639.79	1459.35
P10_12	155.64	716.00	1525.14
P10_13	165.89	527.89	1201.44
P10_14	168.01	606.86	1176.83
P10_15	158.29	624.63	1302.08
P10_16	156.98	551.94	1328.72
P10_17	187.00	617.66	1332.09
P10_18	120.42	509.78	1291.25
P10_19	186.00	580.94	1344.89
P10_20	153.83	558.06	1368.13

Instance	$\sum_{i\in V_N} d_i$	$C(N)$	$\sum_{i\in N} C(\{i\})$
P15_1	224.30	858.74	1109.31
P15_2	174.59	556.84	772.23
P15_3	238.76	589.91	747.13
P15_4	249.16	587.18	812.21
P15_5	208.66	694.56	906.59
P15_6	215.06	634.86	891.49
P15_7	269.69	714.51	866.95
P15_8	253.84	661.80	864.76
P15_9	198.90	597.93	888.47
P15_10	174.64	643.78	998.31
P15_11	206.73	616.87	900.64
P15_12	241.29	516.03	617.50
P15_13	244.20	619.16	775.01
P15_14	252.91	703.11	885.58
P15_15	212.95	766.88	1030.23
P15_16	197.15	581.32	917.01
P15_17	288.93	772.14	936.66
P15_18	264.41	726.01	879.64
P15_19	190.54	535.80	702.54
P15_20	211.48	492.84	823.12

In the following the different experiments which have been executed are discussed. The efficiency of the parameters of the branch-and-price algorithm are solely judged on the average time to determine the optimal cost of all coalitions for all of the instances.

D.1 Initialisation method

The algorithm is initialised with the label setting algorithm proposed by Feillet et al. [15] without the heuristic adjustment. Furthermore, a last in first out node strategy is utilised for the branch-and-bound procedure. In this experiment the amount of columns added during the pricing problem is varied while comparing the initialisation methods. The first method (*trivial*) initialises each branch-and-price procedure with only the trivial routes, whereas the second method (*savings*) also uses the routes generated by the savings algorithm. The results of the tests on the preliminary instances are displayed in Table 8.2.

Table 8.2: The computational time in seconds. The first column denotes the maximum columns added at each iteration of the pricing problem. The second and third column, denote the time each algorithm required to solve the instances on average

(a) Instances of type P10

Columns	Trivial	Savings
50	11.77	12.05
100	12.97	12.54
1000	13.47	14.03

(b) Instances of type P15

Columns	Trivial	Savings
50	44.17	44.54
100	42.53	46.31
1000	60.80	60.39

From the results it follows that the savings algorithm actually does not improve the overall performance. This may be due to the fact that the algorithm requires the savings to be sorted with is a process of non-polynomial order. As such, the algorithm is utilised without the savings algorithm in the initialisation. Furthermore, after each algorithm of the pricing problem at most 100 columns are added to the restricted master problem.

D.2 ESPPRC algorithm

The algorithm is initialised with the trivial routes. Besides this, the last in first out node strategy is utilised for the branch-and-bound procedure and at most 100 columns are added in each iteration of the pricing problem. In the following both elementary shortest path problem with resource constraints algorithms as proposed by Feillet et al. [15] and Range [40] are compared with and without the heuristic adjustment based on the computational time required. The results are presented in Table 8.3.

Table 8.3: The computational time in seconds. The first column denotes the maximum columns added at each iteration of the pricing problem. The second and third column, denote the time each algorithm required to solve the instances on average

(a) Instances of type P10

Heuristic	Feillet et al. [15]	Range [40]
Yes	10.95	10.53
No	12.97	13.15

(b) Instances of type P15

Heuristic	Feillet et al. [15]	Range [40]
Yes	34.67	32.85
No	42.53	42.64

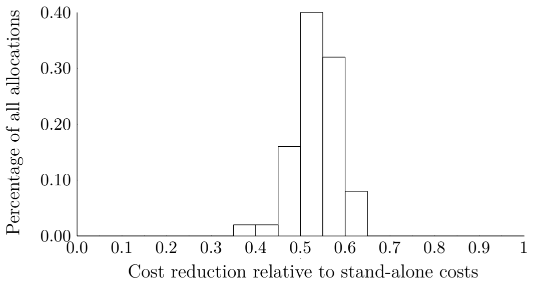
Overall, the algorithm as proposed by Range [40] with the heuristic adjustment outperforms the other options. As such, this option is included in the algorith.

D.3 Node strategy

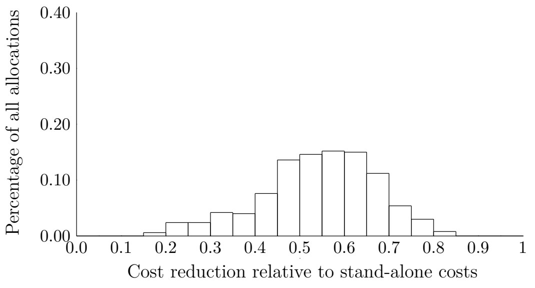
The algorithm is initialised with the label setting algorithm proposed by Range [40] with the heuristic adjustment. Furthermore, a last in first out node strategy is utilised for the branch-and-bound procedure and at most 100 columns are added in each iteration of the pricing problem. The final comparison compares the last in first out and first in first out strategies. For the P15 instances the algorithm required on average 32.85 seconds with LIFO and 31.79 seconds with FIFO. The P10 instances required on average 10.53 seconds with LIFO and 10.73 seconds with FIFO. Taking the scalability of the algorithm into account, the FIFO method is implemented.

E Achieved savings per method

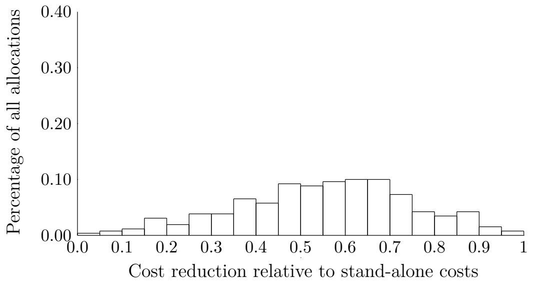
E.1 Type A



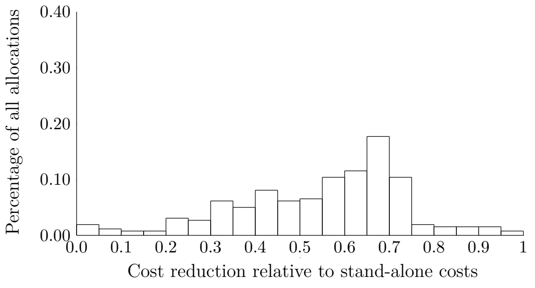
(a) Star



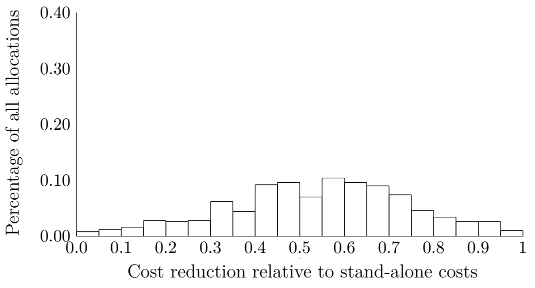
(b) Shapley



(c) Lorenz



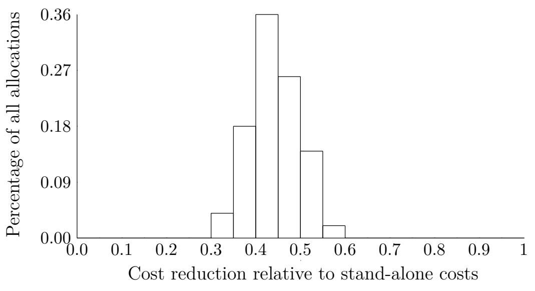
(d) Equal Profit Method



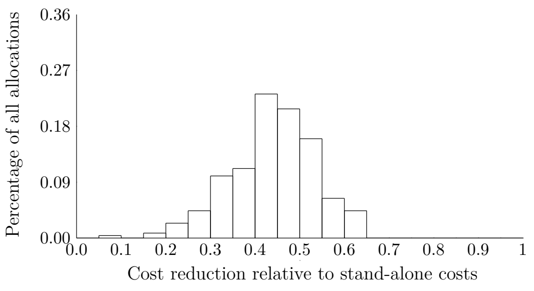
(e) Nucleolus

Figure 8.3: Percentage savings for instances of type A. The average cost reduction is 53.3%.

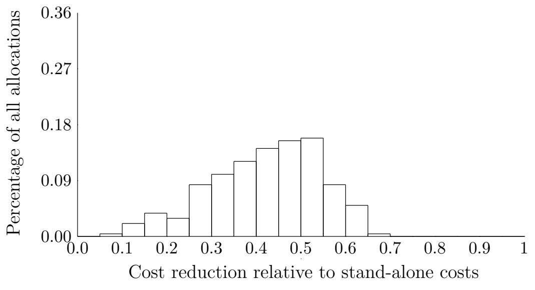
E.2 Type B



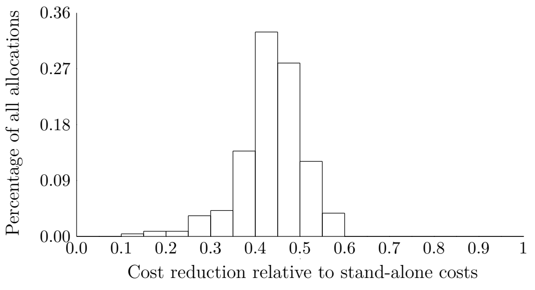
(a) Star



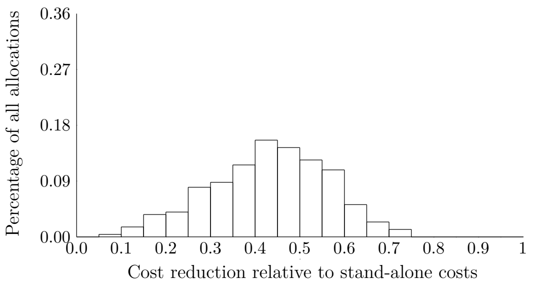
(b) Shapley



(c) Lorenz



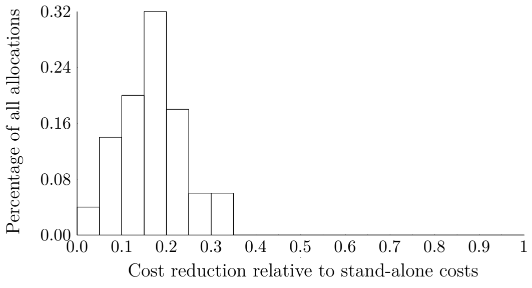
(d) Equal Profit Method



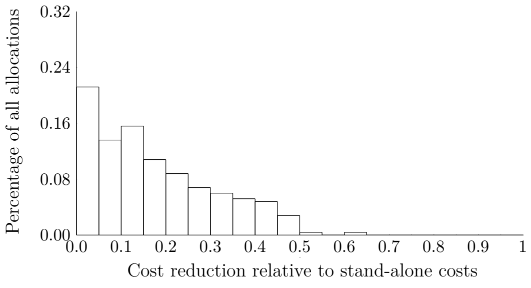
(e) Nucleolus

Figure 8.4: Percentage savings for instances of type B. The average cost reduction is 43.8%.

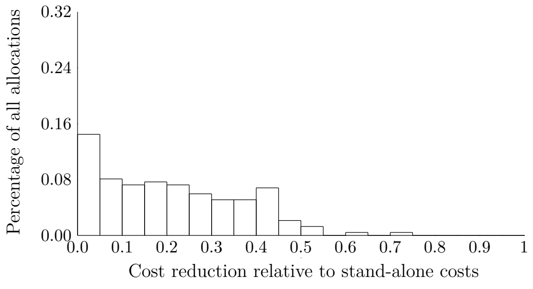
E.3 Type C



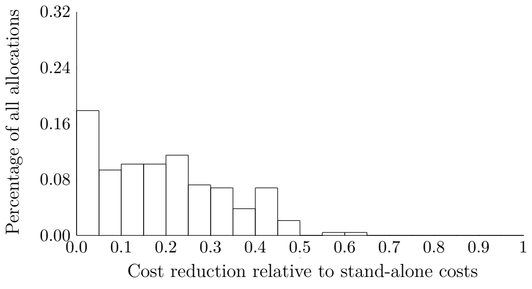
(a) Star



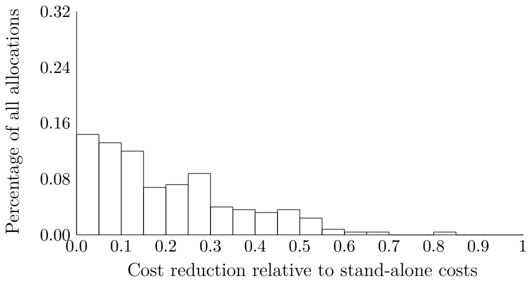
(b) Shapley



(c) Lorenz



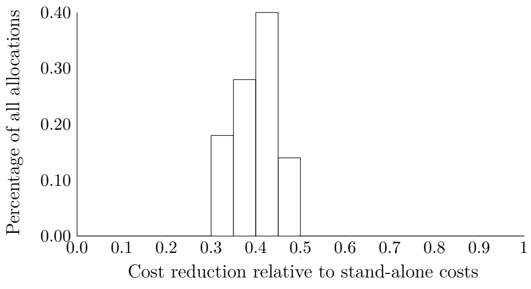
(d) Equal Profit Method



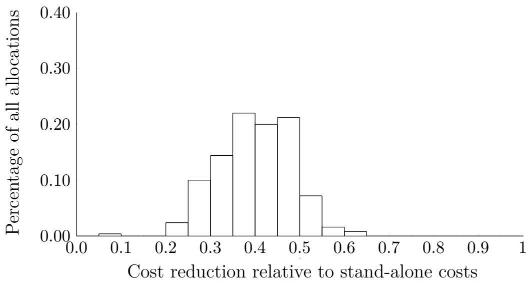
(e) Nucleolus

Figure 8.5: Percentage savings for instances of type C. The average cost reduction is 16.7%.

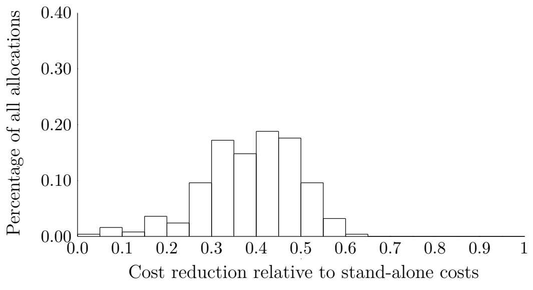
E.4 Type F



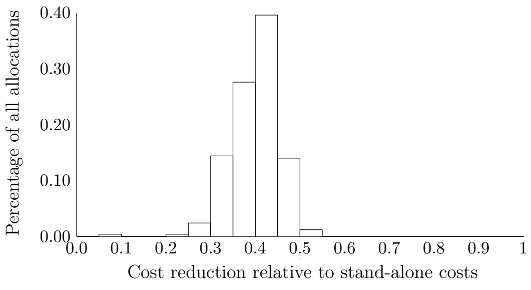
(a) Star



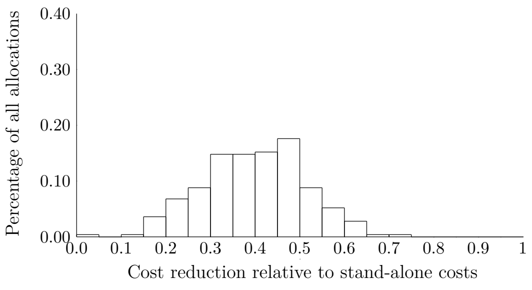
(b) Shapley



(c) Lorenz



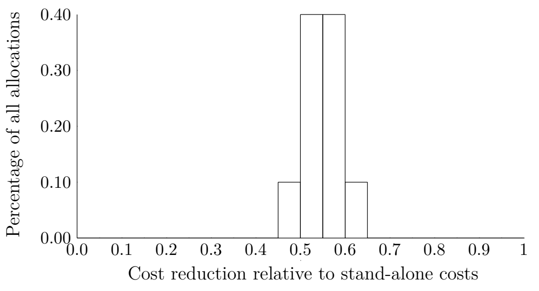
(d) Equal Profit Method



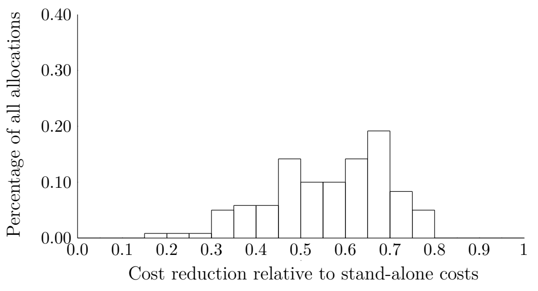
(e) Nucleolus

Figure 8.6: Percentage savings for instances of type F. The average cost reduction is 40.0%.

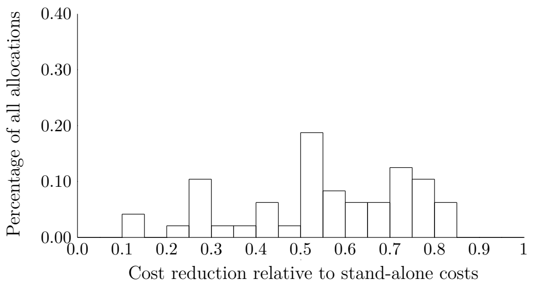
E.5 Type G



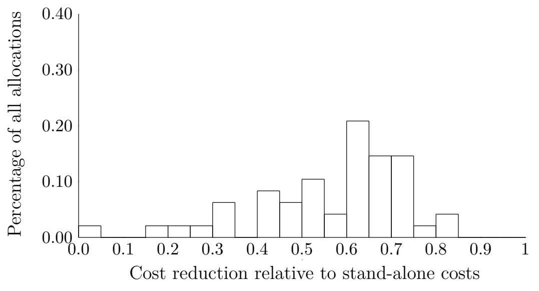
(a) Star



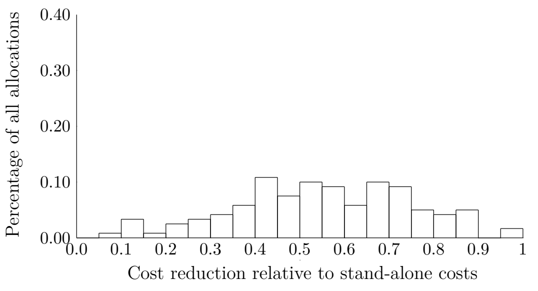
(b) Shapley



(c) Lorenz



(d) Equal Profit Method



(e) Nucleolus

Figure 8.7: Percentage savings for instances of type G. The average cost reduction is 55.3%.