# Advances in practical optimal coalition structure algorithms

Chattrakul Sombattheera
University of Wollongong


# Advances in Practical Optimal Coalition Structure Algorithms

A thesis submitted in fulfillment of the
requirements for the award of the degree

**Doctor of Philosophy**

from

UNIVERSITY OF WOLLONGONG

by

**Chattrakul Sombattheera**

School of CS and SE.
June 2010

*Dedicated to*

*My Parents*

# Declaration

This is to certify that the work reported in this thesis was done by the author, unless specified otherwise, and that no part of it has been submitted in a thesis to any other university or similar institution.

_____

Chattrakul Sombattheera
June 20, 2010

# Abstract

This thesis presents a number of algorithms for forming coalitions among cooperative agents in pragmatic domains where traditional cooperative game theory solution concepts do not apply due to bounded rationality of agents. While previous work in coalition formation in multi-agent systems research operated on relatively small number of agents, e.g. less than 30 agents, this work explores coalition formation among 100 agents, this is due to limited computational resources not the performance of the our algorithms. We explore a best-first search centralized algorithm for optimal coalition structures which is based on a novel idea of deciding what is the best coalition to put into coalition structure being generated. Empirical results show that the solution reaches optimality quickly and terminates quickly in pragmatic domains. We further explore on optimal coalition structures with distributed algorithms in linear and non-linear domains. For the linear domains, we explore linear production and integer programming. For the non-linear domains we explore logistic providers. Based on existing algorithms, we explore a novel environment of forming coalitions in supply networks involving buyers, sellers and logistics providers agents. In this setting, buyers form coalitions to increase their negotiation power while sellers and logistics providers form coalitions to aggregate their supply power and optimize their resources usage.

# List of Publications

The material of this thesis is based on the following publications:

1. Chattrakul Sombattheera, Aditya Ghose: A best-first anytime algorithm for computing optimal coalition structures. AAMAS (3) 2008: 1425-1428

2. Chattrakul Sombattheera, Aditya Ghose: A Pruning-Based Algorithm for Computing Optimal Coalition Structures in Linear Production Domains. Canadian Conference on AI 2006: 13-24

3. Chattrakul Sombattheera, Aditya Ghose: A Distributed Algorithm for Coalition Formation in Linear Production Domain. ICEIS (2) 2006: 17-22

4. Chattrakul Sombattheera, Aditya Ghose: Supporting Dynamic Supply Networks with Agent-Based Coalitions. IEA/AIE 2006: 1127-1137

5. Chattrakul Sombattheera, Aditya K. Ghose: A Distributed Branch-and-Bound Algorithm for Computing Optimal Coalition Structures. SETN 2006: 334-344

6. Chattrakul Sombattheera and Aditya Ghose: Agent-based Coalitions in Dynamic Supply Chains. the international conference 9th Pacific Asia Conference on Information Systems (PACIS 2005).

7. Chattrakul Sombattheera, Aditya Ghose, Peter Hyland: A Framework to Support Coalition Formation in Supply Chain Collaboration. ICEB 2004: 1-6

# Acknowledgements

First and foremost, I thank my supervisors: Professor Aditya Ghose, who introduced me to the area of coalition formation, guided me to produce most of my publications, etc. There are too many to mention here and I just want to say that I owe him a great deal. Associate Professor Peter Hyland, who brought me to do this PhD in Wollongong, supported me throughout the difficult times, and most importantly, help me to reach the end of this thesis.

I also would like to thank, Dr. Richard Booth and Mr. Evan Morrison, who have been very sincere and very helpful in proofreading most of the content of this thesis. I thank my old DSL colleagues, Chee Fon "Chief", Peter Harvey, Duc, Aneesh (and many others) who helped train me via so many presentations. Of course, new members of DSL colleagues, George Koliadis and Evan, are very generous. I thank my Thai fiends (as far as I can remember): Noi, Mam, Tip, P' Suay, Savanid, etc. for their friendship and help.

I thank my bosses (Deans of Faculty of Informatics, Mahasarakham University, Thailand): Associate Prof Wanchai Rievpaiboon, Assistant Professor Sangkom Pumpan, Assistant Professor Sujin Butdeesuwan and Associate Professor Wirat Pongsiri, for being supportive throughout my study leave. I thank my colleagues in Management Information System Department, Ajarn Preecha, Ajarn Boy, Ajarn Chumsak, Ajarn Namphon, for being supportive and taking care of my teaching during my absence.

To my family, I thank my father, Khun Por Prayoon Sombattheera, who contributed his saving to my education. It has always be emotional whenever I think about what it means to him and to me. I owe him a lot. I thank my mother, Khun Mae Pannee Sombattheera, who has been the best of the best supporter throughout this very long journey. I can never come this far without her support. My thanks go to my brothers, Piboonseth and Paripat Sombattheera for their support throughout these years.

The last but not the least, I thank Dr. Sujanya Sombattheera, my cousin sister who inspired me to come to Australia and helped me a lot during the first half of my 12 years in down under.

# Contents

# List of Tables

# List of Figures

# Chapter 1

# Introduction

## 1.1 Introduction

For over a decade, software agents have been used to solve a wide range of problems in which agents represent the goals or behaviours of independent human agents. For the most part, these problems have been intentionally limited in scope, using only small numbers of agents with a limited degree of interaction between the agents themselves. Agents have often been used to model interactions between groups of human agents but, once again, these interactions have typically focused on solving problems under only a single constraint. Historically, research has been limited to these simpler problems due to the lack of computer processing power and the lack of efficient algorithms to deal with the enormous increase in complexity when more and more agents are allowed to interact freely with one another to solve problems under heterogeneous constraints i.e. constraints on more than one variable.

However, real-world problems, which could be solved using agents, are unlike these simple problems because they often involve large numbers of agents, each representing stakeholders who may be organized into different groups, each of which may be operating under a different sets of constraints. Even more challenging are real-world problems in which members of a single stakeholder group may form a coalition to try to improve their performance e.g. a group of manufacturers sharing raw materials to meet a large order which no single manufacturer could supply. In even more complex problems, stakeholders may form coalitions with stakeholders from other groups as well as from within their own group e.g. a group of manufacturers who collaborate with logistics providers to distribute a product to multiple buyers. All of these different stakeholders may have heterogeneous constraints e.g. a logistic provider (LP) may want to minimize traveling time, minimize fuel use and maximize the utilization of each truck.

Most of the methods used to solve such coalition formation problems have used deterministic methods, in which every possible coalition is identified and the outcome for each of

the members of a coalition is calculated for every coalition. These exhaustive approaches are computationally very demanding and will only work for small numbers of agents, typically less than 30. However, an alternative approach would be to find an "anytime" solution i.e. a method which identifies and stores good coalitions while it continues to search for the optimal coalition. This method can be stopped at anytime and, while it may not have reached the optimal coalition, it will have found a coalition which adequately meets the needs of the stakeholders. This research aims to develop an anytime method for Optimal Coalition Structure (OCS) problems.

## 1.2 Background

Software agents can be used for a wide range of purposes such as monitoring the status of a system or a device, recording user interactions, or customizing a search on the Internet. These agents may be resident on a single computer or be deployed across networks, particularly the World Wide Web. More advanced agents, or intelligent agents, are able to communicate with other agents and to make decisions based on their initial parameters plus information gathered from the environment or from other agents. When a number of intelligent agents share a common environment, they are a multi-agent system (MAS).

One area of growing interest to MAS researchers are applications in which agents not only communicate with one another but actually collaborate with one another to solve a problem which they could not solve individually. This may be because the agent does not have the functionality to carry out the whole task or because an agent lacks access to computing resources e.g. CPU time, RAM, storage etc. which another agent on a different computer does have access to. When agents try to collaborate they must commit their own resources to the task.

When several agents make a binding agreement to cooperate, we say a coalition has been formed. The point of forming a coalition is to have every member of a coalition perform better than each one might do individually. For example, out of 10 local plumbers, 6 may decide to band together to buy materials in bulk at a cheaper price than each might do individually. The savings made by this coalition is called the coalition value. The 4 remaining plumbers may have been excluded because they do not buy enough materials to make it worthwhile for the original 6 to include them. This may seem odd, because 10 plumbers would almost certainly get a better deal than 6 i.e. the coalition value would be slightly higher.

However, coalitions come at a cost. When buying in bulk, it may be necessary for all the plumbers to meet to distribute their bulk purchase. Travel costs or the amount of time taken

to arrange the meetings or to divide up the order are all costs of being in the coalition which might not have existed at all if the plumbers had all bought independently. The costs of including the additional 4 plumbers may exceed the benefit of having their extra buying power, in which case the 6 plumbers are actually better off if they exclude the other 4 plumbers. So, the value of any given coalition depends on the benefit that the coalition provides i.e. the coalition value, and on the coalition costs.

However, when considering the outcome for a coalition, we need to take into account the effect of other agents which are not part of the coalition. For example, given three agents 1, 2 and 3, possible coalitions might be $\{1, 2\}$ or $\{1, 2, 3\}$ or $\{1\}$. In the first example we have not accounted for agent $\{3\}$ which is, effectively, a coalition of 1 but may still have an effect on the outcome. So the "complete" set of coalitions corresponding to the first case is actually $\{\{1, 2\}, \{3\}\}$. This "complete" set of coalitions is known as a coalition structure. The second example, $\{1, 2, 3\}$, is itself a coalition structure because no agents are unaccounted for. In the third example, $\{1\}$, there are 2 possible coalition structures, each of which may produce a different outcome. Possible coalition structures for the coalition $\{1\}$ are $\{\{1\}, \{2, 3\}\}$ and $\{\{1\},\{2\},\{3\}\}$. Clearly, the outcome for agent 1 might be different if it was competing with each of the other two agents singly or both the other two agents working together. So, we can only determine the outcome of a coalition if we consider separately all the coalition structures in which that agent can exist. For the 3 agents above, the identification of all coalition structures is trivial. However with 20 agents, the task of even identifying all the coalitions is not trivial. Calculating the outcome for all these coalitions is computationally intensive.

Given all of the coalition structures for a set of agents, it is typical for one of those coalition structures to give the best outcome i.e. to have the highest coalition value. This is called the Optimal Coalition Structure (OCS) and the benefit gained by the participants in this OCS is called the Optimal Coalition Structure value (OCS value). The definition of "best outcome" depends on the context and is left deliberately vague here. It will be defined more formally in Chapter 2.

In much of the previous research, an exhaustive or deterministic approach has been used i.e. the algorithm meticulously generates every possible coalition structure and calculates the coalition structure value for each coalition structure. This approach works quite well for small numbers of agents but as the number of agents increase the computational demands rapidly become intractable. The time taken to generate every possible coalition structure may be unacceptable in any practical application. For example, it would be pointless to find that an optimal route for a delivery truck was five minutes faster than the next best option,

if it took 3 hours to find that optimal solution. Moreover, as the number of agents increase, the amount of RAM used to carry out the calculations becomes unacceptably large. Practical restrictions on time or computing resources create a rational bound on the size of problems which exhaustive approaches can solve.

However, one approach which may be useful in solving OCS problems is the "anytime" approach, in which an algorithm keeps track of the "best solution so far" as it continues to search for the optimal solution. Thus the algorithm can be stopped at "anytime" and will be able to return its "best solution so far". While this is intuitively appealing, it relies on the ability of the algorithm to find at least a "good solution" quite quickly. If the algorithm laboriously plodded through millions of poor solutions, for some reason, then its "best solution so far" could be completely unacceptable in practice.

A number of search algorithms may do away with these "poor solutions". A branch and bound search, for example, uses some heuristic to eliminate or prune large numbers of "poor solutions" quickly. Consequently, a search of the remaining search space is likely to produce at least a "good solution" quite quickly. Such searches are at the heart of a successful "anytime" approach to solving problems involving large search spaces. Given the potentially huge search spaces involved in finding Optimal Coalition Structures, an anytime approach may offer a viable solution.

## 1.3 Structure of the thesis

Chapter 1 provides a brief introduction to Optimal Coalition Structure Problems (OCSP) and proposes a possible solution based on an anytime- Best First Search (BFS) approach.

Chapter 2 presents a review of the literature on OCSP, the best current solutions to OCSP and benchmarks against which to test the algorithms proposed in the remainder of the thesis.

The following 4 chapters each proposes a new anytime-BFS algorithm and demonstrates that this algorithm is better at providing solutions than the current stat-of-the-art approaches.

Chapter 3 presents an anytime-BFS algorithm to solve a general OCSP i.e. one without any specific domain requirements. As in most previous research, this algorithm assumes that the coalition value is known a priori. We propose a set of heuristics that help us identify nearly Optimal Coalition Structures (OCS) quickly. The heuristics work in three steps: setting configuration upper bounds, estimating the optimal coalition structure value, and searching for that optimal value using a branch-and-bound BFS strategy. Empirical results show that this algorithm approaches the optimal coalition structure value in a reasonably short period of time.

Chapter 4 presents an adaptation of the algorithm in Chapter 3 to a real-world setting, namely, a linear production domain, in which each coalition value is not known a priori. The common goal of the agents is to maximize the system's profit. In our algorithm, agents perform two tasks: i) identify profitable coalitions, and ii) compute OCS. We show that our algorithm outperforms exhaustive search in generating OCS in terms of elapsed time and number of coalition structures generated.

Chapter 5 adapts the general algorithm in Chapter 3 to the real-world domain of supply chain networks. The agents take on the roles of three distinctly different sets of stakeholders, namely, buyers, sellers and logistics providers (LPs). Agents take two steps to form coalitions: i) agents in each stakeholder "sector" sequentially form primary coalitions in order to increase bargaining power, selling capacity or service efficiency, and ii) agents form secondary coalitions across sectors in order to finalize the deal and deliver goods to buyers. We propose a negotiation protocol and deliberation mechanism. The negotiation protocol allows thorough communication among agents within and across sectors. The deliberation mechanism allows agents to consider potential coalition members and attractive payoffs for them. We provide examples of how they can help agents form coalitions successfully.

Finally, Chapter 6 studies coalition formation in qualitative preference games in which agents carry out a number of different tasks. Each task is specified by a requirement which is a number of constraints over a set of attributes. These constraints are hard and soft ones. The attributes include price to be paid to agents. The price for each task is negotiable. Hence agents have to trade off between their resources and expected payoff. This kind of problems exist in many real-world domains including electronic commerce, composite web services, supply chains and logistics, grid computing, etc. The algorithm proposed in this chapter is a combination of the algorithms in Chapters 4 and 5.

# Chapter 2

# Background

## 2.1 Introduction to Coalition Formation

This chapter surveys related work in coalition formation conducted by other researchers that provides the background for the research presented in this dissertation. We firstly introduce the concept of coalition formation in cooperative game theory including a discussion on relevant solution concepts. We introduce coalition formation research in multi-agent systems including negotiation and deliberation. We discuss pragmatic aspects that fail to be handled adequately by both cooperative game theory and present research in coalition formation, and which are the motivation of this research. Lastly, we explore related work to our research topics including centralized algorithms for computing optimal coalition structures, distributed algorithms for computing optimal coalition structures, coalition formation in dynamic supply networks, and qualitative coalition formation.

Coalition formation is the process that leads to cooperation among agents within a multi-agent system. Coalition formation was first studied in cooperative game theory by Von Neumann and Morgensternčite89. With the growth and maturity of multi-agent system research, coalition formation has gained more attention from researchers and has been regarded as an important area in multi-agent systems. We shall informally explore both cooperative game theory and coalition formation in multi-agent systems below.

### 2.1.1 Cooperative Game Theory

Game theory studies decision making by multiple decision making units, which we shall refer to as **agents** henceforth, and whose decisions are inter-related. The goal of game theory is to find stable states in which none of the agents will want to change their decisions. Such a stable state is called an *equilibrium*. A principle that describes reasons which lead agents to an equilibrium is a *solution concept*. Game theory can be divided into *non-cooperative* games and *cooperative* games. In the non-cooperative game environment, agents are not

allowed to collaborate or communicate with each other. A real world example is the anti-trust law that prohibits agents from forming cartels. Research in non-cooperative game theory seeks to identify strategies which are the best possible response to other agents' strategies A well known solution concept in this area is Nash Equilibrium [59] in which none of the agents can benefit from changing their strategies.

In contrast to non-cooperative game theory, cooperative game theory allows for agents to communicate that leads them to cooperation [42] from which they can benefit more individually. Agents communicate in order to negotiate with regard to whom they can cooperate and how the joint benefits will be distributed among them. When several agents make a binding agreement to cooperate, we say a *coalition* has been formed. Hence, the cooperative game theories are also known as the theories of *coalition formation* [42]. Mathematically, given set $N$ of $n$ agents, a coalition is a non-empty subset $S$ of $N$, $S \subseteq N, S \neq \emptyset$. The set $N$ itself is called the *grand coalition* while a coalition of one agent is called *singleton coalition*. Let **S** be the set of all coalitions, whose size of $S$ is $2^n - 1$. Given a set of 3 agents, $N = \{1, 2, 3\}$, all the 7 coalitions are $\{1\}, \{2\}, \{3\}, \{1, 2\}, \{1, 3\}, \{2, 3\}$ and $\{1, 2, 3\}$. As in set theory, the *cardinality*, $|S|$, of $S$ is the size of (the number of agents in) $S$. Once agents have formed coalitions, they can be viewed as if they have divided themselves into a mutually exclusive and exhaustive partitions. We define a *coalition structure*, $CS$, as a partition of $N$. A $CS$ can be described by $CS = \{S_1, S_2, \ldots, S_m\}$. The set of all $CS$ is denoted by $\mathcal{CS}$. For example, given $N = \{1, 2, 3\}$, all $CS$ in $\mathcal{S}$ are $\{\{1\}, \{2\}, \{3\}\}, \{\{1, 2\}, \{3\}\}, \{\{1, 3\}, \{2\}\}, \{\{2, 3\}, \{1\}\}$, $\{\{1\}, \{2\}, \{3\}\}$.

Mathematically, a $CS$ has to satisfy three conditions [42]:

$$\begin{aligned} &1)\ S_j \neq \emptyset, j = 1, 2, \ldots, m, \\ &2)\ S_i \cap S_j = \emptyset \text{ for all } i \neq j, \text{ and} \\ &3)\ \bigcup S_j = N. \end{aligned}$$

The joint benefit of a coalition is call the *coalition value*, which is a numeric value that usually represents the utility which accrues from their cooperation. It is quite common in cooperative game theory that the coalition value is money value, e.g., dollars. Cooperative game theory assumes that there is a *characteristic function* [42], $V$ that assigns a real number to each $S$, $V : 2^n \rightarrow \mathbb{R}$. We shall denote the coalition value of $S$ with $V_S$. Hence, a cooperative $n$-person game in characteristic function form is defined by the pair $(N; V)$ [42]. The portion of $V_S$ given to agent $i$ is the *payoff*, $U_i$, of the agent for which the agent plays the game. The collection of payoffs to each agent is the payoff vector, $U = (U_1, U_2, \ldots, U_n)$, which specifies the payoff for each respective agent. Putting together a coalition structure and a payoff vector is the *payoff configuration* [42], $(U; CS)$, which describes a possible

outcome of the game. For example,the payoff configuration $(5, 10, 5; \{1, 3\}, \{2\})$ means agents 1 and 3 have formed a coalition and receive payoff for 5 dollars each while agent 2 remains a singleton coalition and receives 10 dollars payoff on its own.

**Games Environments**

Classical research in cooperative game theory considers games within the *superadditive* [42] environment in which the value of a coalition is at least as much as the sum of the values of each pair of its subcoalitions, e.g.,

$$V_{S\cup T} \geq V_S + V_T \text{ for all } S, T \subseteq N \text{ such that } S \cap T = \emptyset.$$

In contrast to superadditive, the *subadditive* environment is one in which the coalition value of a given coalition is strictly less than the sums of the coalition value of each pair of its subcoalitions, e.g.,

$$V_{S\cup T} < V_S + V_T \text{ for all } S, T \subseteq N \text{ such that } S \cap T = \emptyset.$$

In both environments, there is monotonicity in coalition value based on the size of coalitions. However, a *non-superadditive* [89] environment is one in which coalition values have no relationship to the size of coalitions at all. They are arbitrarily random. This environment is similar to the real world. It is less explored in cooperative game theory but has recently received more attention in multi-agent systems research recently [77, 48, 49, 84, 89, 86, 13, 88, 91].

### 2.1.2 Example of Cooperative Game

We now consider a classic game, the Sandal Maker game [42, 57, 81]. In this game, there are five agents (sandal makers). Agent 1 and 2 make only left sandals, while agents 3, 4, and 5 make right sandals. (Although no sandal maker in the world would utilize such a special way of production, this convincing example provides a fruitful model of cooperative game.) In one cycle, a left sandal maker can produce 17 sandals, while a right sandal maker can produce 10 sandals. Obviously, a single sandal is worth nothing, only a pair of left and right sandals can be sold for 20 dollars. All the scrap leather and unused sandals are thrown away at the end of each cycle. In this game, agents need to form coalitions of left and right sandal makers in order to create value to their coalitions then divide the payoffs. Given this simple information, we can determine coalition values below. Since an agent alone cannot sell its sandals, the value of a singleton coalition is 0. Also, a coalition of agents that produces the

same side sandals is worth 0. Apart from this, a coalition consisting of both agents capabled of producing each moiety (side) will be basically constrained by the smallest number of either moiety. For example, the value of a coalition of two agents, each of a moiety, is limited by the smaller number of right sandals. The value of a coalition of three agents, one of which is of the left moiety, is limited by the number of left sandals. The value of a coalition of four members, one of which of the right moiety, is limited by the number of right sandals. Finally the coalition value of the grand coalition is limited by the number of left sandals. Note that we will denote a coalition $S$ with the list of its member to designate the coalition value as well. For example, $V_1$ refers to the value of coalition $S = \{1\}$ while $V_{1,2}$ refers to the value of coalition $S = \{1, 2\}$. Below is the characteristic function that summarizes coalition values:

$$
\begin{aligned}
&V_1 = V_2 = V_3 = V_4 = V_5 = 0 \\
&V_{1,2} = V_{3,4} = V_{3,5} = V_{4,5} = V_{3,4,5} = 0 \\
&V_{1,3} = V_{1,4} = V_{1,5} = V_{2,3} = V_{2,4} = V_{2,5} = 200 \\
&V_{1,3,4} = V_{1,3,5} = V_{1,4,5} = V_{2,3,4} = V_{2,3,5} = V_{2,4,5} = 340 \\
&V_{1,2,3} = V_{1,2,4} = V_{1,2,5} = 200 \\
&V_{1,3,4,5} = V_{2,3,4,5} = 340 \\
&V_{1,2,3,4} = V_{1,2,3,5} = V_{1,2,4,5} = 400 \\
&V_{1,2,3,4,5} = 600
\end{aligned}
$$

If agent 1 and 3 agree to make a deal, while player 2 and 4 agree on another deal, a payoff configuration could be $(100, 50, 100, 150, 0; \{1, 3\}, \{2, 4\}, \{5\})$. Is this, however, a solution of the game? Since agents are self-interested, reaching such agreement may not always be this easy because there may be a chance that some agents are still looking to increase their payoffs. In the following, we shall explore solution concepts that bring stability to the game.

### 2.1.3 Solution Concepts in Cooperative Games

The study of coalition formation seeks to find solutions that lead to a stable state in which every agent is satisfied with its payoff and has no incentive to deviate from its coalition. Solution concepts established include Bargaining Set [8], Stable Set [60], Kernel [24], the Core [53], Nucleolus [79] and Shapley value [82]. Among them, (although many of them are not directly involved in our research) we briefly describe the following as they have also been studied in multi-agent systems research as well.

### The Core

Von Neumann and Morgenstern [60] consider that searching for stable states in a cooperative game is actually searching for payoff vectors that satisfy all agents. Hence, there is no incentive for any agent to deviate. They propose the idea of *individual rationality* that states that an agent in a coalition will never accept any payoff less than what it could receive from its singleton coalition, $U_i \geq V_i$ for all $i$. This individual rationality is virtually part of every stable state otherwise there will be at least one agent who deviates in order to satisfy this condition. Von Neumann et al. [60] also propose that agents should form coalitions such that the sum of coalition values is maximal. This is referred to as *group rationality* [60]. Since the environment they study is superadditive, $V_N$ is the largest coalition value. It can be claimed that agents should refuse any payoff configuration such that $\sum U_{i,i\in N} < V_N$. Von Neumann et al. [60] define group rationality as "the sum of every agent's payoff in the grand coalition is equal to the the grand coalition's value", $\sum U_{i,i\in N} = V_N$. The implication of this assumption into the general case is that agents should try to maximize the system's utility. A payoff vector that satisfies individual and group rationality is called an *imputation*.

Based on these rationalities, Gillies [30] defines the last level of rationality, i.e., *coalitional rationality* which requires that the sum of payoffs of agents in any coalition is not less than the coalition value, $\sum U_T \geq V_T$ for every $T \subseteq N$. There is no incentive for any agent to leave its coalition. Hence, it brings stability to the system. Gillies [30] names the set of payoff vectors that satisfy all three levels of rationality as the *Core*. Of all existing solution concepts in cooperative game theory, the core is simply the most beneficial concept that brings the most wealth to individual agents, coalitions, and the system as a whole. However, it is the hardest to satisfy as we shall cover in detail later.

Let us consider a simple game of 3 agents in a superadditive environment, explained in [103]. The game is defined by the characteristic function below:

$$
\begin{aligned}
&V_1 = V_2 = V_3 = 0 \\
&V_{1,2} = 0.25, V_{1,3} = 0.5, V_{2,3} = 0.75 \\
&V_{1,2,3} = 1
\end{aligned}
$$

A payoff vector $(U_1, U_2, U_3)$ is in the core of the game if

$$
\begin{aligned}
U_1 + U_2 &\geq V_{1,2} = 0.25 \\
U_1 + U_3 &\geq V_{1,3} = 0.5 \\
U_2 + U_3 &\geq V_{2,3} = 0.75
\end{aligned}
$$

Payoff vector $(0.25, 0.5, 0.25)$, for example, meets all three conditions, so it is in the core.

Let us consider another example, the House Selling game discussed in [60, 103] which was analyzed by Von Neumann et al. [60]. Agent 1 has a house which it values at $100,000 and wants to sell it. Agents 2 and 3 are potential buyers, who each has $200,000 in cash and values the house at $200,000. The coalition value, in this case, is actually the difference between the amount that buyers and seller value the house. Because any singleton coalition and a coalition of buyers cannot make any deal, hence their values are 0. A coalition of agent 1 and one of the buyers is $100,000. The grand coalition also (theoretically) has the value $100,000 (although the house can not be divided). Hence, the characteristic function is shown below:

$$V_1 = V_2 = V_3 = 0$$
$$V_{1,2} = V_{1,3} = 100,000, V_{2,3} = 0$$
$$V_{1,2,3} = 100,000$$

The core of this game is the set of payoff vectors $(U_1, U_2, U_3)$ with $U_1 + U_2 \geq 100,000$ $U_1 + U_3 \geq 100,000$, and $U_2 + U_3 \geq 0$. The only payoff vector which satisfies these conditions is $(100000, 0, 0)$. It implies that agent 1 sells the house to either agent 1 or 2 with the maximum possible price of $200,000. The only agent who gets all the benefit of this economic cooperation is agent 1. The negotiation that may lead to this conclusion could be agent 2 offers $150,000 to agent 1. Agent 3 then raises the bid by offering agent 1 $175,000, and so on. As long as the offer is below $200,000, there is a chance both agents can raise the bids.

Now, let us consider the Sandal game. Since the largest amount possible is $600, the proposed payoff vector $(100, 50, 100, 150, 0)$ is definitely not in the core because it is not group rational. Payoff Vector $(120, 120, 120, 120, 120)$ is in the core because it satisfies individual, group and coalitional rationality. No agent can deviate and be better off.

### Shapley Value

Since agents are self-interested, a fair distribution of a coalition value among coalition members should satisfy agents very easily. Shapley [82] seeks for an imputation that represents a fair distribution of payoffs taking into account the contribution each agent made into the coalition value. This fair distribution is known as the *Shapley value*. In a game $(N; V)$, Shapley's concept of a fair distribution of imputation $U = (U_1, U_2, ..., U_n)$ is based on three axioms.

**Axiom 1**: $U$ should only depend on $V$, and should respect symmetry in $V$. If agents $i$ and $j$ have symmetric roles then $U_i = U_j$.

**Axiom 2**: If $V_S = V_{S-i}$, a coalition without $i$, for all coalitions $S \subseteq N$, then $U_i = 0$, e.g, the payoff to dummy agent $i$ who has no contribution to any coalition is 0. Also, adding such a dummy agent does not change the value $U_j$ for other agents $j$ in the game.

Based on the first two axioms, Shapley proposes that any game can be broken down into a sum of symmetric games (a symmetric game is one where an agent's payoff can be transposed as the other agent's payoff [60]) with dummies added. Suppose that $(N; V)$ and $(N; W)$ are two different games with the same set of agents. The sum game $V + W$ can be defined as $(V+W)_S = V_S+W_S$ for all coalitions $S$. Now, we have 3 games, the imputations to be assigned to them should be related as below:

**Axiom 3**: $U[V + W] = U[V] + U[W]$

The idea is that if it is fair for some agent $i$ to get $U_i[V]$ in $V$ and $U_i[W]$ in $W$ (both symmetric games), it would seem fair for the agent to get the sum of these two payoffs in the game $V + W$.

Shapley [82] proposes a theorem that states that there is only one imputation that satisfies the three axioms. To calculate the imputation, consider the agents forming the grand coalition step by step. Start by one agent and add one additional agent at a time. As each agent joins, award the new agent an additional value he contributes to the coalition. By saying an agent *contributes* a value to a coalition, we mean the agent increases the coalition value by the difference between the coalition value with the presence of the agent and that of the coalition value without the presence of the agent. Once this is done for each of the $n!$ grand coalitions' permutation, divide the accumulated awards to each player by $n!$ to give the fair imputation. Consider the following game [103].

$$V_1 = V_2 = V_3 = 0$$
$$V_{1,2} = 2, V_{1,3} = 4, V_{2,3} = 6$$
$$V_{1,2,3} = 7$$

The 6 ordered (3! of orderings) grand coalitions are: $\{1, 2, 3\}, \{1, 3, 2\}, \{2, 1, 3\}, \{2, 3, 1\}, \{3, 1, 2\}$ and $\{3, 2, 1\}$. Given $S = \{1, 2, 3\}$, the first member is agent 1, who contributes the value 0 to the empty coalition( whose value is also 0). The second member is agent 2, who contributes the value $V_{1,2} - V_1 = 2 - 0 = 2$ to coalition $\{1, 2\}$. The third member is agent 3, who contributes the value $V_{1,2,3} - V1, 2 = 7 - 2 = 5$ to coalition $\{1, 2, 3\}$. Consider coalition $\{3, 2, 1\}$, the value added by each player to the coalition is:

Agent 3: $V_3 - V_\emptyset = 0 - 0 = 0$,
Agent 2: $V_{3,2} - V_2 = 6 - 0 = 6$,
Agent 1: $V_{3,2,1} - V_{3,2} = 7 - 6 = 1$

Repeat the same process for each of the ordered coalitions such that we can obtain the value each player contributes to each of the ordered coalitions as shown below. The column "Order" contains all the permutations of the grand coalition. The column "Contribution of Agents" contains all the contributions each individual agent made to the existing coalition. Note that the contribution of each agent illustrated below depends on the order the agent joins the coalition. Given $S = \{1, 3, 2\}$, agent $1$ contributes the value $0$ to the existing coalition (empty), agent $3$ contributes the value $4$ to the existing coalition, and agent $2$ contributes the value $3$ to the coalition.

| | Contribution of Agents | | |
|---|---|---|---|
| Order | 1 | 2 | 3 |
| {1,2,3} | 0 | 2 | 5 |
| {1,3,2} | 0 | 3 | 4 |
| {2,1,3} | 2 | 0 | 5 |
| {2,3,1} | 1 | 0 | 6 |
| {3,1,2} | 4 | 1 | 2 |
| {3,2,1} | 1 | 6 | 0 |
| Total | 8 | 14 | 20 |

Then we find the average contribution of each player over the six ordered coalitions for the Shapley value, i.e. $= \frac{1}{6}(8, 14, 20) = (1.33, 2.33, 3.33)$. Although the Shapley value guarantees a "fair" distribution of payoffs to agents, it does not tell which coalitions are formed.

**Kernel Solution Concept**

While Shapley value finds a fair distribution by taking into account every permutation of the grand coalition, in many cases some agents' contribution is really low and affects their average contribution. It can be argued that this is not really fair to them because they are not obliged to be part of those low valued coalitions. How can the payoffs for these agents be really fair?

To answer this question, Davis and Maschler [24] propose another solution concept, namely, the *Kernel*. The kernel stabilizes coalition structure by balancing each pair of agents payoffs in every coalition. For a game $(N; V)$ and a payoff configuration $(U; CS)$, there may be a subset of $N$ (group of agents), which contemplate leaving their respective coalitions to form a new coalition $R$. The difference between the value of $R$ and the sums of their

collective payoffs, $U_R$, is defined by Davis et al. [24] as the *excess*: $e(R;U) = V_R - U_R$. Hence, the excess of $R$ is the amount that the prospective coalition members of $R$ may gain (or lose, depending on the sign of the excess value) if they were to leave their current coalitions in the given $(U;CS)$ to form $R$. Now, consider any two agents $i$ and $j$ in a coalition $S \in CS$. Agent $i$ may join any alternative coalition alone, $R \notin CS$ and $j \notin R$. Each of these coalitions will have an excess with respect to $(U, CS)$as defined above. The largest of these excesses is called the *maximum surplus* of agent $i$ over agent $j$ with respect to $(U;CS)$, e.g.,

$$\mathfrak{S}_{i,j} = max_{R|i\in R, j\notin R} \quad e(R;U).$$

This maximum surplus is the maximum amount agent $i$ can gain (or minimum amount it must lose) if it deviates from $(U;CS)$ to any alternative $R$. Hence, agent $i$ can claim that it potentially could gain that much payoff. Similarly, agent $j$ can do the same and claim that its maximum surplus is $\mathfrak{S}_{j,i}$. Now, if both

$$\mathfrak{S}_{i,j} > \mathfrak{S}_{j,i} \text{ and } U_j > V_j,$$

e.g., agent $i$'s maximum is greater than that of agent $j$ and agent $j$'s payoff is greater than its singleton coalition's value, then it is said that agent $i$ **outweighs** agent $j$, with respect to $(U;CS)$ [24]. In other words, agent $i$ has greater maximum surplus and can ask for compensation from agent $j$ otherwise it may leave for better payoff in $R$. However, agent $i$ cannot ask for all, it is bounded by individual rationality, $V_j$, e.g., agent $j$ would not accept anything lower than this value.

If neither agent outweighs the other, then they are in equilibrium. An equilibrium between the two agents exists when a payoff configuration satisfies one of the following conditions:

- $\mathfrak{S}_{i,j} = \mathfrak{S}_{j,i}$; their maximum surpluses are equal and nullify each other's claim for compensation.
- $\mathfrak{S}_{i,j} > \mathfrak{S}_{j,i}$ and $U_j = V_j$; although agent $i$'s maximum surplus is greater than that of agent $j$, it cannot claim compensation from agent $j$ because agent $j$ may deviate to its own singleton coalition.
- $\mathfrak{S}_{j,i} < \mathfrak{S}_{i,j}$ and $U_i = V_i$; this is the reverse case of the above condition.

For the global equilibrium, Davis et al. [24] define the set **K** of all payoff configurations $(U;CS)$ such that every pair of agents $i, j \in S \in CS$ are in equilibrium. Here, the sense of fairness is different from Shapley value in that it considers potential coalitions that might

raise agents' payoffs. If agent $i$ claims that it can raise a higher payoff without agent $j$ than vice versa, then agent $j$'s payoff in $(U; CS)$ is too much. Hence, some of it should compensate agent $i$.

Let us consider the game [42],

$$V_1 = V_2 = V_3 = 0;$$
$$V_{1,2} = 90; V_{1,3} = 80; V_{2,3} = 70;$$
$$V_{1,2,3} = 105.$$

Suppose we have a configuration payoff $(U; CS) = (45, 0, 35; \{1, 3\}, \{2\})$ on offer. In order to find if it is in the Kernel, we find its excesses, maximum surpluses, then equilibria as follows. Suppose agent $1$ and $3$ are considering their coalition $\{1, 3\}$. There are two coalitions that include 1 but exclude 3, e.g., $\{1\}$ and $\{1, 2\}$. The excesses and maximum surplus are

$$e(1; U) = V_1 - U_1 = 0 - 45 = -45.$$
$$e(\{1, 2\}; U) = V_{1,2} - U_1 = 90 - 45 = 45.$$
$$\mathfrak{S}_{1,3} = max(-45, 45) = 45.$$

Similarly, $\mathfrak{S}_{3,1} = 35$. Since $\mathfrak{S}_{1,3} = 45 > \mathfrak{S}_{3,1} = 35$, agent 1 outweighs agent 3, i.e. $(U; C) = (45, 0, 35 : \{1, 3\}, 2)$ is not in the Kernel. What about the $(U; CS) = (50, 30, 25; \{1, 2, 3\})$? The excesses and maximum surplus between agent 2 and 3 are:

$$\mathfrak{S}_{2,3} = max(V_2 - U_2, V_{1,2} - U_2) = max(0 - 30, 90 - 30) = 60.$$
$$\mathfrak{S}_{3,2} = max(V_3 - U_3, V_{1,3} - U_3) = max(0 - 25, 80 - 25) = 55.$$

Agent 2 outweighs agent 3, i.e. $(50, 30, 25; \{1, 2, 3\})$ is not in Kernel. What about the $(U; C) = (45, 35, 25; \{1, 2, 3\})$? The excesses and maximum surpluses are:

$$\mathfrak{S}_{1,2} = max(0 - 45, 80 - 70) = 10 = max(0 - 35, 70 - 60) = \mathfrak{S}_{2,1},$$
i.e. player 1 and 2 are in equilibrium.
$$\mathfrak{S}_{1,3} = max(0 - 45, 90 - 80) = 10 = max(0 - 25, 70 - 60) = \mathfrak{S}_{3,1},$$
i.e. player 1 and 3 are in equilibrium.
$$\mathfrak{S}_{2,3} = max(0 - 35, 90 - 80) = 10 = max(0 - 25, 80 - 70) = \mathfrak{S}_{3,2},$$
i.e. player 2 and 3 are in equilibrium.

Hence, the $(U; C) = (45, 35, 25; \{1, 2, 3\})$is in the Kernel.

## 2.2 Coalition Formation in Multi-agent Systems

Taking into account the above mentioned points, coalition formation in multi-agent systems deals with more realistic environments. Instead of just considering the outcomes, we consider coalition formation as a process, consisting of inter-related activities, i.e., deliberation and negotiation, that eventually helps agents reach agreement to form coalitions. In deliberation, agents deal with necessary calculations, including computing coalition values, choosing potential coalition members, and computing reasonable payoffs. In negotiation, agents follow a protocol to exchange information, which was computed during their individual deliberation, among each other to convince potential coalition members to make a decision. Note that coalition formation requires simultaneous multilateral rather than bilateral negotiation [78].

### 2.2.1 Impractical Issues in Cooperative Game Theory

As we have covered so far, cooperative game theory ignores multiple pragmatic aspects. First of all, it involves merely a very small number of agents (usually less than 10) whereas multi-agent systems can involve a larger number of agents (can be as large as 1000 agents [21]). The larger number of agents causes a lot of consequential pragmatic issues. While the number of agents grows linearly, the number of coalitions and coalition structures grow exponentially (as we shall discuss in greater detail later in 2.3). Practically, in order to calculate them repeatedly and rapidly retrieve coalition values for some 44 agents ($2^{44}$ or $4 \times 10^{12}$ coalitions) would require the most powerful computer in the world.[1] Note that this figure would leave no memory left for any other operation at all. Let alone the number of coalition structures (see 2.3.1). The fastest algorithm for generating coalition structures [69, 71] merely deals with 27 agents on typical personal computers. The problem becomes intractable for even a slightly larger number of agents.

Cooperative game theory assumes a characteristic function to instantly yield optimal [42] coalition values. Combinatorial games that are studied in game theory assume coalition values are given and focus on the aspect of coalition value distribution. However, it is known that optimizing coalition values is also necessary in pragmatic coalition formations [75, 76]. The complexity of computing these values can be as complex as linear (Linear Production game [63]) or non-linear (Traveling Salesman game [66]). Although the computation of a

[1] IBM's BlueGene has memory of 32768 GB or $2^{44}$. Ref: http://www.top500.org/system/7747, accessed on 18 June 2009.

coalition value in linear games can be done in polynomial time, it is not the case for non-linear games which require exponential time. The computation of all coalition values can become intractable very easily due to the exponential number of coalitions.

By considering a very small number of agents and assuming perfect rationality and complete information, important aspects such as dynamicity of coalition, deliberation of individual agents and negotiation among agents are not considered. Von Neumann et al. [60](page 44) admitted that

> "... our theory is thoroughly static. A Dynamic theory would unquestionably be more complete and therefore preferable."

Von Neumann et al. [60] focus on mathematical analysis of the game and the consequential possible outcomes. Note that in many cases, the outcomes are a set (the Core or Kernel) not a single value. Furthermore, since the study focuses on the payoffs, it is not known exactly what coalitions are formed. On the other hand, given a large number of agents who are realistically of bounded rationality and are often under time constraint, deliberation may not be optimal. Negotiation can be important in that it can disclose useful information derived from other agents' deliberation and help agents make decision appropriately.

By taking into account the above mentioned pragmatic aspects, coalition formation in multi-agent systems research proposes algorithms and protocols that allow agents to deliberate and negotiate. Most importantly, the outcome is a payoff configuration, which describes the payoff for each agent and what coalitions are formed. Below, we discuss the overview of coalition formation in multi-agent systems. Detailed discussion on related topics will be presented in later sections.

### 2.2.2 Early Dynamic Coalition Formation

Dynamic coalition formation dates back to the study of *Transfer Schemes* by Stearns [102]. It is a mathematical analysis of the dynamic process of coalition that leads to a final payoff configuration (a payoff vector for all agents and a coalition structure of all agents) for a given game with respect to a solution concept, such as the Kernel. Stearns provides algorithms that begin with a payoff vector, repeatedly compute excesses of each pair of agents, balance the differences, and eventually converges to an equilibrium. The algorithms can be implemented in a "largest-excess-first" fashion or in a "round-robin" fashion. The study assumes perfect rationality and complete information. Given that this work is an algorithmic analysis, Stearns [102] shows that agents can eventually converge to the Kernel with the cost of exponential time.

The real implementation of dynamic coalition formation among (computer software) agents took place between the late 80's and the early 90's. One such work was Zlotkin and Rosenschien [119] which studies coalition formation of $n$ agents in subadditive task-oriented domains. Agents are allocated parcels to deliver in a grid environment and they have to finish their tasks within limited steps. Agents are allowed to cooperate. Hence, their coalition values are the costs they can save by exchanging tasks. The payoffs to agents are calculated based on the Shapley value. Because of the procedure of computing the Shapley value, Zlotkin et al. [119] argue that agents may be reluctant to be the first in the permutation because they will get payoff 0 by definition. Zlotkin et al. [119] propose to deploy a simple combination of private and public key (a cryptographic approach) to guarantee random permutations of any coalition. Each agent chooses a permutation of a coalition in which it is interested. The permutation is then encrypted with the agent's private key and is broadcast with its public key. Having received all messages, each agent chooses a message randomly and decrypts it. The combination of all the permutations will be used as the initial permutation. This work is a good example of applying a cooperative game theory into a multi-agent system. Agents can decide to form coalitions by applying one of the proposed schemes which will suggest the most appropriate coalition for each agent.

Ketchpel [43] proposes a "two-agent auction" mechanism for coalition formation that prescribes an algorithm for agents to negotiate. The model which Ketchpel studies is a basic economic model where rational agents join auctions to win contracts from which they earn guaranteed payments. In this model, agents play similar roles in the bargaining process. Firstly, agents broadcast their individual offers to other agents. These offers are ranked in preference orders by interested agents. Each interested agent then chooses the most attractive agent, as a potential coalition member, in order to form a coalition of 2 agents. These two agents then enter the "two agent auction" phase where an agent will act as the manager. The manager will propose to its potential coalition member a payoff using the Shapley value. The member agent receives a fixed payment for its role in the coalition. The principle of ranking potential coalition members for negotiation seems to be a mainstream of coalition formation algorithms derived in later research [90, 89, 86, 88, 91, 85, 83, 48, 49]. Note that only a small number of coalitions are formed due to the ranking.

Similar to Ketchpel [43], Shehory and Kraus [84] explore coalition formation among cooperative agents. These agents use resources, which may include materials, energy, information, communications, etc. to fulfill their tasks. The tasks are to deliver parcels, but the coalitions can be made of more than two agents at a time. They propose two ways of forming coalitions: computing-oriented and negotiation-oriented. With computing-oriented

coalitions, an agent will be appointed as the computing agent, who will compute the payoffs for coalition members. With negotiation-oriented coalitions, agents can negotiate about their payoffs. In this environment, they define "strong" coalitions as ones whose potential coalition values are high, thus is preferred by agents. On the other hand, "weak" coalitions are ones whose potential coalition values are low. Strong coalitions are the ones which other coalitions join to form larger ones. Each coalition will have a representative, who values the potential coalition higher and takes care of negotiation with weaker coalitions. The negotiation is to distribute relevant information, i.e., payoff and resource vectors. Shehory et al. [84] also analyze the complexity of negotiation which is $2(n-1)^2$ operation for their algorithm. The payoffs to agents in each coalition are based on the "common extra" payoff, i.e., the increased value of the joint pair of coalitions. This common extra payoff will be distributed equally among agents in the new coalition. Since, agents operate in superadditive environment, they eventually form the grand coalition.

### 2.2.3 Kernel in Multi-Agent Systems

So far, we have discussed some early work in coalition formation in multi-agent systems that are more pragmatic than cooperative game theory. With respect to stability, many of them deploy the Shapley value. Here, we discuss the application of the Kernel in dynamic coalition formation.

It has been shown by Stearns [102] that the complexity of computing the Kernel is exponential. However, Klusch and Shehory [47] show that the Kernel can be reached in polynomial time in some settings. They study coalition formation among information agents. An information agent is an active intelligent database front end trying to satisfy its own application specific tasks alone or in cooperation with other agent to gather information. In their setting, information agents try to rationally cooperate to discover intentionally relevant information in non-local domains. The utility which each agent achieves is derived from the search on the set of discovered interdatabase dependencies which satisfy its own and/or received requests. The utility achieved from the search on its own request is the singleton coalition value of each agent. The coalition value of composite coalitions is the sum of the singleton coalition values. They propose a detailed protocol for forming coalitions under a predefined duration. Since the formation is done in a distributed fashion, they claim that the complexity of computing Kernel is, in the worst case, of order $O(n^{|S_{max}|})$, where $S_{max}$ is the largest coalition being considered.

Another attempt to reduce the complexity of reaching the Kernel is conducted by Shehory and Kraus [87]. They study coalition formation in general environments, i.e., non-superadditive, and propose an algorithm which can reach th Kernel in polynomial time. The coalition formation is partitioned into two levels: the social level and the strategic level. The social level is composed of the coordination regulation protocols which must be agreed upon by different designers of agents in advance. The strategic level consists of strategies for the individual agent to act in the environment for maximization of its own expected payoff, given the social level, and can be decided upon by individual agents during the coalition formation process. At the strategy level, a coalition will calculate a joint value of a new coalition if its value will be increased. Given the increased value, if at least the payoff vector of a component coalition is increased, that coalition should attempt to make an offer to the other coalition. This strategy is not enforced but is recommended.

Whereas other works assume that coalition values are known for certain, Blankenburg, Klusch and Shehory [13] consider coalition formation within real-world environments where coalition values are known only to some degree of certainty. They apply the concept of fuzziness to the Kernel to allow specification of uncertain coalition values. The fuzzified Kernel extends the classical one to contain information about the degree of certainty to which a fuzzy configuration is Kernel [24] stable. An example of such a configuration is when information agents have a task to deliver some information to their users. The agents' evaluation of the relevance of the available information to the task is uncertain. This uncertainty is represented by fuzzy numbers. In their example, however, they assume the maximum coalition size of 2, just half of the number of agents 4. The coalition protocol is the same as Shehory et al. [87]. The complexity of the method to reach stability in general cases is exponential. However, polynomial time can be achieved by limiting the size of coalitions being considered as in [47].

Blankenburg and Klusch [12] also extend their work to the security domain. They consider real world settings like health care and $m$-commerce which require high preservation of the privacy of user information. By using their algorithm in [47], they show that such a requirement of privacy preservation can be obtained. More interestingly, they show that, in principle, fraudulent agents may try to $a$) unreasonably strengthen their bargaining power, or $b$) play some cheating strategies but doing so is very computationally costly in practice.

### 2.2.4 Bounded Rational and Time-Constrained Coalition Formation

Real world settings usually involve complex optimization, which raises two closely related issues: bounded rationality and time constraints. Given a complex problem, solving it optimally is hard to achieve because the available computational power is limited. Furthermore, a solution is needed by a certain point of time. In the following, we shall explore previous work in coalition formation research that also takes into account bounded rationality and time constraint (which our work principally follows).

Sandholm and Lesser [75, 74] analyze coalition formation among self-interested agents in real world settings where agents are logistics dispatch centers who try to cooperate, i.e. forming coalitions, in order to reduce their costs. This work raises the importance of solving combinatorial optimization problems as part of the coalition formation because it yields the coalition value, i.e., cost reduction achieved by the optimization of appropriate routes for trucks belonging to cooperative dispatch centers. Given the real world setting described in [75, 74], computing coalition values is a very complex problem. Note that the coalition value is needed in timely fashion, i.e., the optimization is time constrained. Sandholm et al. [75, 74] also take into account the cost of computation because optimizing such complex problems requires intensive computation power. Sandholm et al. [75, 74] state that the solution quality is a trade off against computation cost. The performance of the algorithm deployed by the agents in the optimization affects significantly the stability of coalitions among agents due to their bounded rationality. Sandholm et al. [75, 74] assume agents have the same computation power and algorithm, hence agents would achieve the same result for a given problem. Agents form coalitions according to the optimization result without real negotiation. Based on the empirical results [75, 74], agents can often reach a stable state, i.e., the "rational-bounded" Core. However, the final coalition structure (see 2.3) may not be the grand coalition.

Further work with respect to pragmatic constraints is studied by Tohm and Sandholm [107]. This work takes into account the costs of communication and deliberation of rationally bounded agents in order to model the dynamic process of coalition formation in superadditive environments. Agents are self-interested. They try to maximize their payoffs. They join coalitions if they have the chance to increase their payoffs. A novel idea deployed in this work is the exchange of beliefs of agents during their negotiation, i.e. each agent tries to convince other agents to join its coalition. The new information will be kept as agents' new beliefs replace the old ones. In each step of the negotiation, the costs of communication and deliberation are taken into account in order to find the right agent to negotiate with as well as when to decide to form coalitions before the time has run out. In their setting, agents can

reach a state of stability. They also note that the effect of bounded rationality depends on the computation in the negotiation process is also intractable.

Soh and Tsatsoulis [98] study coalition formation in a cooperative multiagent system in an incomplete information environment. While lacking knowledge of "noisy, dynamic, an uncertain world", agents need to respond to events under time constraints. Soh et al. [98] propose a model, which has two stages: (1) compiling a list of potential coalition members as the candidate agents, and (2) negotiating with these candidate agents to form coalitions. The first stage takes place when when an agent detects an event in the world that it needs to form a coalition and respond to the event. This stage is called "coalition initialization". The second stage allows for agents to negotiate with the candidates collected in the first stage. Agents negotiate in pairs by exchanging their information and constraints. The negotiation may or may not be successful. Each successful negotiation adds a new member to the agent's final coalition. This second stage is called "coalition finalization". Soh and Tsatsoulis [96] also study within a similar setting and consider coalition formation in a "dynamic, negotiation-based" model where agents cannot form rationally optimal coalitions. With the two-stage approach proposed in [98], the agent who initiates the coalition needs to determine the task distribution among the members of the coalition and design a strategy to successfully form the coalition. Having limited resources and incomplete knowledge about noisy and dynamic environments, agents need to form coalitions in order to react to events instantly.

### 2.2.5 Strategic Coalition Formation

Modern coalition formation research takes into account more pragmatic aspects such as bounded rationality and time constraint. These pragmatic issues, in turn, bring strategies into perspective. Kraus, Shehory and Taase [48] explore coalition formation in the Request For Proposal (RFP) domain. In such a domain, a requester business agent issues an RFP, i.e., a description of a complex task, composed of sub-tasks. The task requires "several service provider agents" to cooperatively address the RFP. Although each agent knows the value of the RFP and its private costs for performing a specific sub-task, it does not know the private costs of other agents. Additionally, agents have to address the RFP within a given time frame. Given these realistic constraints, Kraus et al. [48] argue that existing coalition formation approaches are inappropriate because they assume complete information and no time constraints. Kraus et al. [48] have developed a protocol that allows agents to form coalitions within this environment. With the protocol, agents will use some heuristics for choosing partners to form coalitions under this realistic setting. According to their results, the overall payoff for agents using their heuristics is very close to an experimentally measured optimal

value.

Based on [48], Kraus, Shehory and Taase [49] study further on the "advantages of compromising in coalition formation with incomplete information". Kraus et al. [49] study protocols and strategies for coalition formation with incomplete information under time constraints. In such a setting, Kraus et al. [49] suggest that the "strategies should preferably be stable, lead to a fair distribution, and maximize the social welfare of the agents. These properties are not fully supported by existing coalition formation approaches." Kraus et al. [49] argue that "stability and the maximization of social welfare are supported only in the case of complete information, and only at a high computational complexity." Furthermore, the payoff distribution in such a setting is done in a "naive manner". Under limited computational resources and incomplete information, Kraus et al. [49], investigate various strategies for payoff distribution, including the Kernel (balance in every coalition), self-interested (being greedy and try to maximize payoffs), compromising (trying to be friendly by giving away an agent's own payoff to persuade other agents), etc. The compromising strategy was "specifically examined". The empirical results show that, under their [49] setting, the compromise strategy brings about stability and also raises the social wealth (maximal utility) to the system. By emphasizing the importance of time constraint, the work in [48] suggests to us that achieving empirically good results from a reasonably good algorithm is more achievable and preferable than theoretical results. We shall follow this direction in this research.

Below, we will explore research works that are closely related to our research. They include computing coalition structure which will allow for the search for the maximal wealth of the system, coalition formation in combinatorial settings, coalition formation among buyers, sellers and logistics providers. Lastly, we also briefly explore works in coalition formation in various aspects although they are of related to our work.

## 2.3 Algorithms for Computing Optimal Coalition Structures

Searching for optimal coalition structures has gained much attention from researchers recently. It is so important for two reasons: $i$) it indicates the optimal solution of a given system, and $ii$) it helps determining the core of the system (collective rationality). In the following, we shall discuss the overview of the problem as it is presented in the literature [77].

Given a $CS$, we define its value,

$$V(CS) = \sum_{S \in CS} V_S,$$

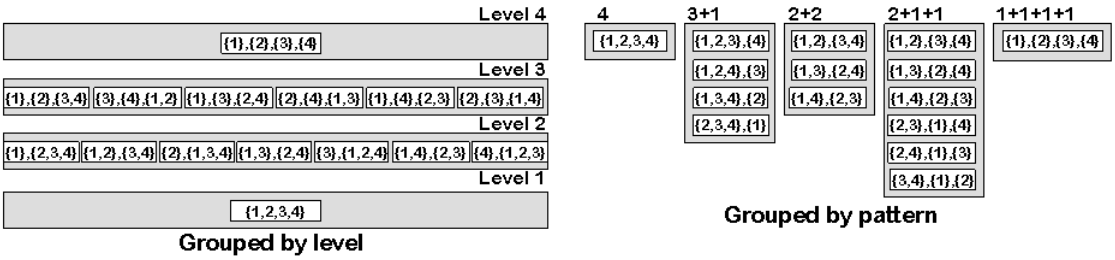


Figure 2.1: **Configuration Bounds**

which indicates the system's utility yielded by that partitioning. An optimal coalition structure is a $CS^*$ such that

$$CS^* = argmax_{CS \in L} V(CS)$$

The number of all coalition structures can be determined by $B_n$ [50], *Bell Number* which is the size of the whole search space. Since the value of $B_n$ can be very large for a small value of $n$, existing algorithms tend to divide the search space into small portions. There are two divisional methods. Firstly, we can categorize $CS$s by the number of coalitions within them [77]. We denote the set of $CS$s, whose number of coalitions of each $CS$ is $1 \leq i \leq n$, by $L_i$. Each $L_i$ is known as a layer. The number of $CS$s in $L_i$ is known as the *Stirling Number of the Second Kind* [50],

$$S(n,i) = \frac{1}{i!} \sum (-1)^{k i}_{k=0} \binom{i}{k} (i-k)^n, \text{where} \binom{i}{k} \equiv \frac{i!}{(i-k)!k!}$$

Hence, the set of all $CS$s is $L = \bigcup_{i=1}^{n} L_i$.

Alternatively, we can categorize $CS$s by the integer partition of $n$ that describes the number of coalitions and their cardinalities. Each instance $j$ of such a partition is known as a "pattern" [101, 100] or a "configuration" [62], $G_j$, which is usually written in the form $b_1 + \ldots + b_k$, where $\sum_{l=1}^{k} b_l = n$. Given a set of 4 agents, all the configurations are 4, 3+1, 2+2, 2+1+1 and 1+1+1+1. Figure 2.1 shows all coalition structures of 4 agents categorized by level and by pattern (configurations).

### 2.3.1 The Analysis of the Problem

Sandholm et al. [77] show that computing the optimal coalition structures in a non-superadditive environment is non-trivial; it is NP-hard because $B_n$ can be very large for a small $n$. Table 2.3.1 shows approximate numbers of coalition structures for $11 \leq n \leq 30$. For $n = 11$ agents, $B_{11}$ is relatively small but is very large for 30 agents, $B_{30} = 8.47 \times 10^{23}$. Let us

Table 2.1: Search Space in Coalition Structure where "$B_n$" is the number of coalition structures, "Largest $L_i$" is the largest layer $i$, "$S(n,i)$" is the number of $CS$ in that layer $i$, "# of Config." is the number of configuration, "Conf Max" is the configuration which has the largest number of $CS$s, "CS Max" is the number of $CS$s in "Conf Max".

| # of Agents | 11 | 12 | 13 | 14 | 15 | 16 | 17 | 18 | 19 | 20 |
|---|---|---|---|---|---|---|---|---|---|---|
| $B_n$ | 6.79E+5 | 4.21E+6 | 2.76E+7 | 1.91E+8 | 1.38E+9 | 1.05E+10 | 8.29E+10 | 6.82E+11 | 5.834E+12 | 5.17E+13 |
| Largest $L_i$ | 5 | 5 | 6 | 6 | 6 | 7 | 7 | 7 | 8 | 8 |
| $S(n,i)$ | 2.47E+5 | 1.38E+6 | 9.32E+6 | 6.34E+7 | 4.21E+8 | 3.28E+9 | 2.57E+10 | 1.97E+11 | 1.71E+12 | 1.52E+13 |
| # of Config. | 56 | 77 | 101 | 135 | 176 | 231 | 297 | 385 | 490 | 627 |
| Config Max | 3(3)+2 | 3(4) | 3(3)+2(2) | 3(4)+2 | 3(5) | 4+3(4) | 3(5)+2 | 3(6) | 4+3(5) | 3(6)+2 |
| CS Max | 3.69E+3 | 1.54E+4 | 5.64E+4 | 2.66E+5 | 1.40E+6 | 5.60E+6 | 3.00E+7 | 1.90E+8 | 9.05E+8 | 4.86E+9 |
| # Agents | 21 | 22 | 23 | 24 | 25 | 26 | 27 | 28 | 29 | 30 |
| $B_n$ | 4.75E+14 | 4.51E+15 | 4.42E+16 | 4.46E+17 | 4.64E+18 | 4.96E+19 | 5.46E+20 | 6.16E+21 | 7.13E+22 | 8.47E+23 |
| Largest $L_i$ | 8 | 9 | 9 | 9 | 10 | 10 | 10 | 10 | 11 | 11 |
| $S(n,i)$ | 1.33E+14 | 1.24E+15 | 1.23E+16 | 1.21E+17 | 1.20E+18 | 1.32E+19 | 1.43E+20 | 1.54E+21 | 1.81E+22 | 2.15E+23 |
| # of Config. | 792 | 1002 | 1255 | 1575 | 1958 | 2436 | 3010 | 3718 | 4565 | 5604 |
| Config Max | 3(7) | 4+3(6) | 4(2)+3(5) | 3(8) | 4+3(7) | 4+4+3(6) | 3(9) | 4+3(8) | 4+4+3(7) | 3(10) |
| CS Max | 3.62E+10 | 1.99E+11 | 1.15E+12 | 9.16E+12 | 5.73E+13 | 3.72E+14 | 2.98E+15 | 2.08E+16 | 1.51E+17 | 1.21E+18 |

consider the size of search space by levels. Given $n$ agents, the number of coalition structures in each level increases exponentially, around $6^n(roughly)$, as the level gets higher. It reaches the peak around the middle level and decreases exponentially towards the top level. For $n = 11$ agents, level 5, $L_5$, has the highest number of coalition structures, i.e., approximately $S(11,5) = 2.74 \times 10^5$ coalition structures. For $n = 30$ agents, the largest level is 11 whose search space is approximately $2.15 \times 10^{23}$. The whole search space for any given $n$ agents is slightly higher than that of largest level. In case of dividing the search space into configurations, each divided search space can be relatively small. For 11 agents, there are 56 configurations. The largest configuration is 3+3+3+2 whose number of coalition structures is approximately $3.69 \times 10^3$. For 30 agents, there are 5604 configurations. The largest configuration is 3+3+3+3+3+3+3+3+3+3 whose number of coalition structures is $1.21 \times 10^{18}$.

### 2.3.2 Coalition Value Distribution

In traditional algorithms of this problem, there are 4 four environments that are considered: normal, uniform, superadditive and subadditive. These environments involve the distribution of coalition values only. It is obvious that the structure of $CS^*$ depends on the distribution of coalition values and so does the performance of an algorithm. Small coalitions (e.g., of size 1 or 2, for example) tend to be in the optimal coalition structures if forming larger coalitions does not increase the value high enough. In this case, it is better for the system

if most agents remain singleton coalitions. However, many real world environments involve cooperation among agents. Small coalitions do have enough resources to perform tasks or create any coalition value. Composite web services is a simple example. Agents have to form coalitions in order to create joint values which will be divided among coalition members. Hence, coalitions of small cardinality tend to be useless in composition of optimal coalition structures. As shown in table 2.1, the number of coalition structures in certain areas, e.g. Largest $L_i$ and Config. Max, can be very large. Hence, it is important that the algorithm be consistently efficient compared with various distribution forms. In the following section, the previous algorithms for finding optimal coalition structures will be discussed.

## 2.4 Previous Centralized Algorithms in Optimal Coalition Structures

Due to the fact that the search space of the optimal coalition structure problem is very large and the search terrain is arbitrarily random, the algorithm to solve the problem needs to perform efficiently. Previous algorithms tend to divide the whole search space into smaller parts. The division is based on the structure of coalitions in $CS$s and a lexicographic order of agents within coalitions.

Sandholm, Larson, Andersson, Shehory and Tohm [77] propose an anytime algorithm (that can yield an approximate answer, whose quality depends on the computation executed, at any time) that guarantees improvement of the worst-case bound as the algorithm proceeds. The large search space of all coalition structures is divided into levels, each of which is $L_i, 1 \leq i \leq n$ (a level where each $CS$ has $i$ coalitions). The algorithm advances through levels $1, 2, n, n-1, \ldots, 3$ and search through all $CS$s in each $L_i$ in the breadth-first search manner. The algorithm can guarantee that the solutions that have been found after finishing the first two levels are within the bound $k = n$ from the optimal solution. Although this bounds drops as more levels have been completed, the search space in many levels is still large. Dang and Jennings [22] improve the performance of Sandholm et al.'s algorithm. Having finished the first three levels (1, 2 and $n$), Dang et al.'s algorithm then generates a list of indicators that will be used to determine various configurations across levels $n-1$ and $3$. The indicator is simply used to choose any configuration that contains at least a coalition of a certain cardinality. Given a set of 20 agents, for example, the next cycle after $L_n$ is to search through all coalition structures of any configuration that has at least one coalition of cardinality 16, 15, 13 and 10. Dang et al.'s algorithm guarantees that it reaches bounds closest to the optimal faster than Sandholm et al.'s algorithm. This is due to the greater

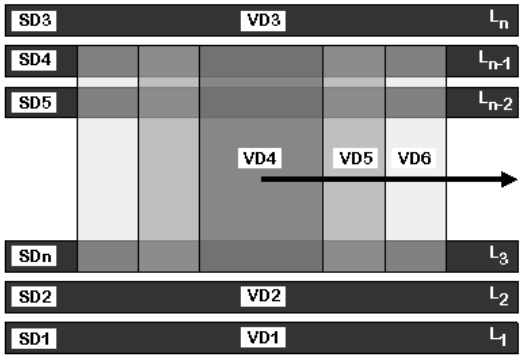


Figure 2.2: **Search Direction in Divided Search Space**
Sandholm et al. [77] divides search space into levels. The direction of search is $L_1, L_2, L_n, L_{n-1}, \ldots L_3$. Dang et al. [22] follow the first two steps of Sandholm et al. but search through portions of remaining levels.

selectivity in searching through $L_i$. Although these two algorithms can guarantee improving solutions as time elapses, they need to exhaustively search through the whole search space in each cycle to guarantee optimality. Figure 2.2 illustrates the search direction in the divided search space.

The common problem of Sandholm et al.'s and Dang et al.'s algorithms is that they rely on the completion of the search on the first two levels. For a small number of agents, e.g., 20-40 agents, the search space over the two levels is practically small, e.g., around $10^6$ and $10^{12}$ coalitions respectively. Once the number of agents grows larger, e.g., 60 agents or more, the search space of the first two levels becomes too large to complete, e.g., around $10^{18}$, let alone the remaining layers. Moreover, the solution obtained within the elapsed time can be bad. Furthermore, accessing the coalition values can be a problem. One can compute coalition values every time a coalition structure is generated but this can cost too much time because the same coalition value has to be computed again and again. An alternative to this is to keep the coalition values in memory. This, however, requires a large space of memory that no single computer can offer.

Sombattheera and Ghose [100] propose the idea of partitioning the search space into smaller sections, referred to as "pattern". Rahwan, Ramchurn, Dang and Jennings [70] follow this idea by proposing a near optimal coalition structures algorithm. To begin with, all coalitions in the first two levels will be examined. Along with this, the upper bound and the mean value for each configuration are established. The configuration to be searched first is either $i$) one whose ratio between the upper bound and the mean value is the lowest, or $ii$) one that is likely for the algorithm to prune the largest search space in its configuration. As the

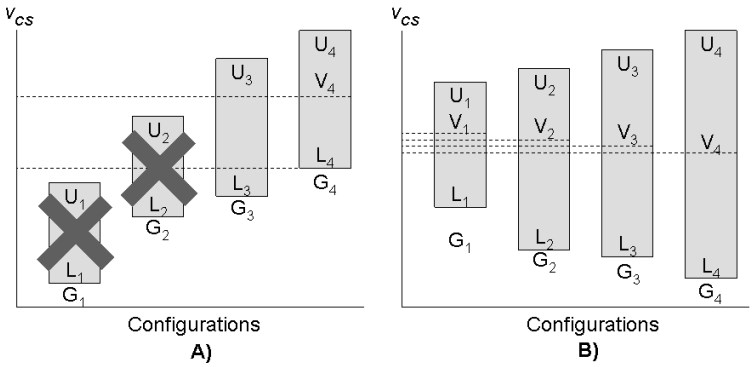


Figure 2.3: **Configuration Bounds**
In general Rahwan et al.'s algorithm can prune portions of the search space (A) when a present solution is better than that of remaining configurations. However, it may fail to do so due to misleading upper bounds(B).

algorithm proceeds, the upper bound of each configuration will be updated and the configuration will be eliminated if its upper bound is lower than the best solution found so far. Given enough time, the algorithm terminates when there is no configuration left to be searched and the solution is the optimal. Based on this work, Rahwan, Ramchurn, Dang, Giovannucci and Jennings [69, 71] develop an optimal algorithm that, to our best knowledge, is *the state of the art of anytime optimal coalition structure algorithm*. They apply pruning over configurations and within coalitions to cut the search space massively and generate optimal solutions rapidly under a number of data distributions [51]. However, this algorithm can be misled by miscalculation of the bound. As shown in figure 2.3, the search will take place in all configurations whose bounds are higher than the actual values.

Yun Yeh [117] proposes a deterministic algorithm to compute optimal coalition structure based on the integer programming technique. His solution is based on the bi-partitioning principle. A whole set of agents will be bi-partitioned into ${}^nC_2$ pairs. Each coalition in each pair will be recursively partitioned downward to the singleton level where all coalitions are of cardinality 1. The algorithm then works upwards for the optimal value of each coalition. At each level, the pair whose combined value is the highest will be the optimal value of the coalition. This algorithm guarantees optimal results with $3^n$ time and space complexities. However, this algorithm is not appropriate for a multi-agent systems environment because the problem becomes intractable for even a small number of agents. On the other hand, Sen and Dutta [80] use an order-based genetic algorithm (an evolution algorithm, which deploys the biological evolution concept, i.e., inheritance, mutation, selection, and crossover, to constitute the search for approximate answer in large combinatorial problems) as a stochastic

search process to identify the optimal coalition structure. Although their algorithm has no performance guarantees, they claim that it is found to dominate the deterministic algorithm in a significant number of problem settings. Due to the nature of genetic algorithms, an additional advantage of their algorithm is its scalability to larger problem sizes and to problems where performance of a coalition depends on other coalitions in the environment. Larson and Sandholm [51] present experimental results for three anytime algorithms that search the space of coalition structures. They show that, in the average case, all three algorithms do much better than the recently established theoretical worst case results in [77]. They also show that no one algorithm is dominant. Each algorithm's performance is influenced by the particular instance distribution, with each algorithm outperforming the others for different instances. They present a possible explanation for the behavior of the algorithms and support their hypothesis with data collected from a controlled experimental run.

As we have discussed before, these algorithms have to scan the set of all coalitions, which is of size $2^n$, in order to observe their values. The size of the input can be very large for a small $n$, let alone the size of coalition structures. To this end, Rahwan and Jennings [67, 68] develop an algorithm for computing coalition values in a distributed manner. The task of computing coalition values is distributed evenly among cooperative agents, who seem to be involved in computing $CS^*$, with respect to communication and computation redundancy. They claim to have massively reduced the number of of messages sent among agents and memory usage. However, they have tested their algorithm against 25 agents only, which is practically small for multi-agent environments. As the number of agents increases linearly, the size of the problem, $2^n$, increases exponentially. Even though the algorithm can divide the task among agents evenly and efficiently, the workload of each agent becomes unmanageable for even a small number of agents. Even with 40, 60 and 80 agents, the size of the task is approximately $2.45 \times 10^{10}, 1.92 \times 10^{16}$and$1.51 \times 10^{22}$, respectively, let alone realistic environments where the number of agents is much more than this.

In real world operation, computing $CS^*$ involves two steps: $i$) computing coalition values, and $ii$) generating coalition structures. Since the existing algorithms for computing $CS^*$ need to scan all coalition values, they can precompute coalition values and store them in memory, if the size of $2^n$ is not too large. Alternatively, the precomputed values may be stored in a database or other storage. This will definitely slow down computing the optimal coalition structure because of the relatively slow access to storage device. The last alternative can be recomputing the coalition value every time a $CS$ is being generated. This seems to be a very low performance approach since computing coalition values itself can be a complex optimization problem. In addition to this, we also have to take into account the need for

an on-time solution, which is not necessarily optimal. It is very doubtful that existing algorithms can guarantee such a quick response because they have to scan all the inputs under limitation of resources. Hence, it is important that we provide a solution in a timely fashion.

## 2.5 Coalition Formation in Combinatorial Settings

Optimization in combinatorial problems (or operations research) has some degree of relationship with early studies of cooperative game theory [106] (although it does not focus on optimizing the combinatorial problem directly). In game theory, as in optimization, decision makers want to act in an optimal way. The main difference is that there is one decision maker (agent) in optimization, while in game theory there are multiple inter-related decision makers. In the following, we will briefly explore works that are related to combinatorial settings. Although many of these works are in cooperative game theory, they provide motivating examples of how important optimization is to coalition formation.

### 2.5.1 Linear Production Games

A simple classic sample is the Sandal Maker game [42, 57, 81] where agents possesses partial resources (either left or right sandals) and need to cooperate to produce salable goods (pairs of sandals). Obviously, the optimization in such a game is trivial but it reflects the underlying fact that gathered resources of a coalition requires complex optimization. In a more complex setting, Owen [63] considers coalition formation in the linear production game. This work is derived from the linear production problem where an agent has to maximize its profit by producing appropriate goods in the right amount. Owen [63] takes this problem one step towards cooperative game theory. Here, each agent possesses a number of resources and produces goods to sell to the market to maximize its profit. Agents can benefit more if they pool their resources together. Owen [63] studies coalition in superadditive domain, i.e., there was no cost involved in the process. As the name suggests, to find out what is the most profitable productions, given available resources, the optimization is basically linear programming. Hence, the coalition values can be obtained by solving linear programs. Owen [63] uses “duality theory of linear programming to obtain equilibrium price vectors and to prove the non-emptiness of the Core”. However, Owen [63] assumes a superadditive environment and ignores other real world costs such as communication, transportation, etc. In such a setting, the most profitable coalition is obviously the grand coalition. Although the payoffs to agents are Owen’s [63] interest, we are convinced that applying a little more realistic constraints to the problem will lead us to another direction of research, i.e., to find the

optimal coalition structures will need a novel approach, which involves complex algorithms. Thus it is our motivation for work in chapter 4, where we redefine his setting as the basis of our work. Note that Owen's [63] goal is to analyze the outcome of the game whereas our work is to invent an algorithm in order to search for optimal coalition structures.

Another example of combinatorial domain in cooperative game theory is the study of the transportation game [73], which is derived from the *transportation problem* [23]. In such a problem, it is important to the management that they need to minimize the transportation cost of a certain product. The product is available at several origins and is to be transported to several destinations according to their individual needs, given that the transportation cost per unit of the product is fixed. (Note that although this problem is also a linear programming one, optimizing for the optimal route to minimize the cost of transportation is not, which we shall discuss below.) Sanchez-Soriano, Lopez and Garcia-Jurado [73] further study the transportation game in the cooperative game theory context. The main purpose of the research is to study the Core of the game which they [73] prove does exist. This game is reversibly also a superadditive game because reducing cost is actually increasing profit, given that the revenue is the same. A larger coalition can allow greater reduction of the cost as in the case of linear production game.

A more complex problem that is similar to the transportation problem is the traveling salesman problem [4] where a salesman has to visit a number of cities connected by a road network. He has to make a route from a city, go through each of other cities once only and go back to the city where he started. The challenge for the salesman is to find a route that will minimize his traveling distance, i.e., cost. Potters, Curiel and Tijs [66] study the problem in cooperative game theory context. In their setting, an academic is to visit sponsoring institutes who will cover the academic for the cost to visit them. The goal is to find a stable allocation of traveling costs among the sponsoring institutes. Potters et al. [66] manage to find the core of the game, however, this work is merely an analysis of a small instance of a potentially complex problem.

Research on coalition formation among transportation agents was conducted by Shehory and Kraus [89]. Instead of having all the costs of traveling known a priori, this work applies a more realistic setting. Shehory et al. [89] study a scenario where agents are taxis traveling on a road network to pick up and drive passengers to various locations. This work considers a non-superadditive environment where the revenue of the agents is the price they charge their customers and the cost includes taxes, fuel, etc. Each agent will have a number of customers awaiting their services. It may be better off for the agents if they can cooperate, this is similar to the postman problem in which agents exchange their letters. Here, agents may exchange

their customers if they find any benefit, i.e., forming a coalition to reduce their costs while serving their customers. The revenue of the coalition is the collective prices charged to their customers. The coalition value is the difference between the collective revenue and the collective cost. To minimize cost in order to maximize the profit is a hard optimization problem which involves a combination of taxis in each coalition and the optimal route to find and deliver their respective customers. However, the number of agents being studied in this work is relatively small. Shehory et al. [89] can find the Kernel stable for the agents in reasonable time and find that forming more coalitions can benefit agents further.

It is widely known that ideas in operation research are widely adopted into cooperative game theory because they are very close to each other [106]. The only difference is optimization in operations research is done by a single agent whereas in cooperative game theory optimization involves multiples agents. However, the work we we have seen so far is more about analyzing the outcome of the games, which involve merely a small number of agents. Although Shehory et al. [89] have shown a good example, we want to push our research further by examining larger number of agents via an algorithmic approach.

## 2.6 Coalition Formation in Supply Networks

Supply networks can be viewed as sophisticated coalition formation. As stated in cooperative game theory [42], a simple trading transaction between a seller and a buyer is considered coalition formation, whose coalition value is the difference between the buyer's and the seller's reserve prices. Coalition formation can takes place among buyers in order to gain negotiation power. Their coalition values are the discount on the prices. Coalition formation can take place among sellers in order to increase their supply power (not to form cartels) and optimize the usage of their resources. Since transportation costs play such an important role in providing value to end customers, we need to consider logistic providers in supply networks. Coalition formation can help logistic providers minimize their costs which in turn can benefit end users in addition to benefiting themselves. In the following, we shall explore coalition formation that take place in different sectors of supply networks.

### 2.6.1 Coalitions of Buyers

Agents can increase their buying power to gain discounts from buying large volumes of goods and distributing discounts as payoffs among themselves. Tsvetovat, Sycara, Chen and Ying [109] propose coalition formation among buyers in an electronic market. The example in the real world being deployed in their work is a number of students, who enrolled in the

same subjects, gather together to buy books in large numbers. Agents, representing students, can form coalitions either before or after the negotiation of the prices with the suppliers. In the former case, any agent who wants to buy a book can start the process of forming a coalition by acting as a leader of a coalition. The agent sends requests, stating the item it wants to buy, for bids from suppliers through an auctioneer agent. Each interested supplier consults with online stores for competitive prices and constructs its bid. Each bid is proposed in a linear price function where price drops according to increasing quantity. The bid is sent to the auctioneer who, after waiting for the expiry of the auction time, broadcasts the bids to respective leaders. Each leader then applies its evaluation strategy to determine the winning bidder. The leader uses the best bid to advertise for coalition members. If the number of coalition members is lower than expected, the leader will suffer the loss. On the other hand, agents may form coalitions first, then negotiate with the suppliers. In order to decide which bid to be taken, agents in the coalition may vote on the bids. However, this can be impractical because agents may not reach agreement so easily. Alternatively, agents may decide to leave the decision to the leader. In this setting, agents have to trust the leader that it will do its best for the sake of the coalition.

Coalitions of buyers can also take place in online shops in the Internet. Hyodo, Tokuro and Ito [37, 36] point out that the allocation of goods available on the Internet can be much more efficient with the application of coalition formation. The motivation is that there are too many online shops available on the Internet, hence buyers are not aware of all of them that sell their desired goods. More importantly, they usually buy goods from some shops they are aware of individually. This is inefficient because they fail to utilize economy of scale, i.e., they lose money that could have been saved by gaining discount from bulk buying. The method proposed in this work is a centralized approach. Hyodo et al. [37, 36] apply genetic algorithm to form coalitions of buyers, each of which is composed of buyers seeking the same product. Hyodo et al. [37, 36] assume the price is a linear function. The algorithm tries to generate coalitions of buyers searching for the same product. However, if a member of a coalition has a reserve price lower than the selling price, such a coalition is invalid and cannot be considered. The unit of each product a seller stocks is finite. If the stock is lower than the requested quantity, the seller cannot sell its product. Hence, some goods may not be allocated to buyers. However, Hyodo et al. [37, 36] show that optimal coalitions for buyers can be found.

In the same spectrum, Ito, Ochi and Shintani [40] consider coalitions of buyers in a more pragmatic fashion. Instead of considering the formation as a static game where all buyers are present all at once, they consider the scenario where buyers are randomly present to the

market. In their setting, sellers possess multiple products. They maintain information on price, deadline and stock of each product. For each product, the price is based on discount rate. The deadline specifies the duration in which the price is effective. Furthermore, they also allow substitutability for buyers' preferences. A buyer, for example, wants to buy a unit of a good at $20. However, he will be indifferent if he can buy two units of another good at $15. The payoff to an agent is the difference between its reserve and selling prices times the quantity. During the negotiation on the price, a seller chooses the best coalition of buyers (one which offers the highest reserve price) and checks for the stock level of the product after the deadline has passed. If it has enough quantity, it informs the coalition if they want to buy the product at the quoted price. The coalition will accept the offer if the price is lower than the reserve price of every member in the coalition. Otherwise, the coalition will disband and enter the market in the next round. In addition to this, they also advance other work by considering cooperation among sellers as well.

Categories of product can be an important issue in coalition formation among buyers. Yamamoto and Sycara [115] consider coalitions of buyers who seek for products of the same categories, which are considered no different to buyers.. A real world example deployed in this work is when a number of buyers want to buy cameras. In the example given, a buyer may have a list of preference on the product. He, for example, values camera C1 for $100 while he values camera C2 for $150. He is indifferent between these two. Another buyer values camera C2 for $140 while values camera C3 for $150. He is indifferent between the two. Buyers of the same categories will be classified into coalitions of the same products. Hence, both buyers may form coalitions for buying C2 while they may form coalitions with other agents for buying C1 and C3. Since goods are indifferent to buyers, they can only buy one product, i.e., they belong to just one coalition. The coalition value is the sum of the difference between the reserve prices of all agents and selling cost (selling price times quantity) of the respective product. The coalition that has the highest value will be pooled for matching with the product. The largest coalition will be chosen from the pooled coalitions and the good will be allocated to its members. With regard to the payoff, those members whose reserve prices are lower than the selling prices receive nothing; they pay their reserve prices. Each of the agents whose reserve prices are higher than the selling price receives a payoff which is the proportion of the difference between their reserve prices, and the selling price. Yamamoto et al. [115] claim that their solution are in the Core.

Although large coalitions may offer higher payoffs (due to higher bargaining power), agents may sometimes prefer smaller coalitions. Asselin et al. [6, 5, 7] argue that waiting

for large coalitions may cost agents dearly because such a large coalition cannot form before the expiration of the offer. In real world environment such as the Internet, agents do not know beforehand how many agents are buying the same products. Some agents may prefer to form small coalitions quickly, execute transaction and gain their payoffs. Asselin et al. [6, 5, 7] propose a coalition formation protocol that consists of customer agents who represent human customers and a grouping agents who is responsible for forming coalitions of customer agents. In their setting, customers are limited to buy just 1 unit of each product they want to buy. Human customers create customer agents and provide the agents a list of wanted products. Each item is a combination of the products and the preferred size of buyer coalitions. Items in the list are ranked by preference of the customers. Customer agents then submit the list to the grouping agent. The grouping agent is the mediator who collects all the requests from customer agents, allocates matched items to coalitions and matches coalitions to sellers. Each coalition is composed of customer agents who are buying the same product and prefer the same coalition size. The payoff to the agent is its reserve price for the product which is exactly specified by the preferred coalition size.

So far we have seen coalitions of buyers who want to buy the same products only. When buyers want to buy a combination of products [55, 54, 35] (which is also common), they have to split their orders. Having seen the growth of combinatorial auction in which an agent can buy a combination of products and benefit from complementarity of its own order, Li and Sycara [55, 54] propose algorithms for combinatorial coalition formation and payoff division in an electronic marketplace. They combine combinatorial auction into coalition formation to help improve the efficiency of the market. Li et al. [55, 54] consider an e-market environment where a buyer buys a combination of various products, which is quite common in our daily life. The discount the seller can offer is, however, based on the quantity of a particular product. They call coalition formation under this condition a "Combinatorial Coalition Formation(CCF)" problem. This is because forming coalitions in this setting is motivated by price discounts on single goods whose volumes are increased by the size of the coalitions. They claim to construct "optimal coalition with respect to each item" which is the division of the reverse price of each buyer appropriately. The optimal coalitions are induced by the complementarity of the items by transferring cost among the coalition members. They present polynomial-time algorithms to find a semi-optimal solution of CCF. With linear price functions (the price drops at a constant level when the quantity raises by one unit), they can also derive a payoff division scheme that is in the Core of the coalition. They claim that that, empirically, the solution derived by the algorithms is "in a satisfactory ratio to the optimal value".

The domain of coalitions of buyers can expand to computational resources available on the Internet. Maheswaran and Bacar [56] study coalition formation in such an environment where agents form coalitions in order to bid for computational resources. They do not explain how exactly agents form coalitions but propose to use "weighted proportionally fair scheme" to divide payoff among agents. With their scheme, members of non-singleton coalitions cannot be better off by deviating to join singleton coalitions. The scheme ensures stable coalition structures. Similar works on coalitions of buyers, whose common idea about payoffs to agents are the share of discounted prices, include [36, 92, 83, 10].

We have covered coalitions of buyers that do split the discount on the prices, as the coalition values, to agents as their payoffs. However, these works seem to ignore one important factor, the cost of transportation. It is quite common in our daily life that we drive a distance to buy some products whose prices are discounted without really caring much whether it is a wise action, taking into account time, fuel, etc. We shall discuss the role of transportation (logistics) in coalition formation later in section 2.6.3.

### 2.6.2 Coalitions of Buyers and Sellers

Coalitions we have seen so far are among buyers. Agents form coalitions for a particular purpose, e.g., buying goods for one transaction, then disband after they have completed transaction. Breban et al. [14] consider coalitions among a number of seller and buyer agents, which may last longer. Their motivation is that in some settings agents usually trade on certain categories of products and their transactions are committed more often among a certain number of agents. The more they reach agreement to trade the more they trust each other, thus it is more likely to reach agreement again between the same parties. Breban et al. [14] introduce a model for long term coalitions between sellers and buyers. The coalition is driven by trust among agents. Trust is the belief that they have been successful in their previous transactions and are more likely to be successful in the future. It is better to form long term coalitions. At a point in time, an agent may or may not belong to a coalition. Once a buyer agent wants to buy a product, it looks for the most trusted seller, i.e., one in the same coalition. If there is none, it chooses one who offer the best price. These pairs of agents then negotiate on price. The negotiation may end successfully, i.e. they agree to trade at a certain price. Each agent keeps track of success rate between itself and other agents. After a successful transaction, the rate is increased. If the pair are not in the same coalition, they may decide to form a long term coalition. On the other hand, when they cannot reach an agreement, the success rate is decreased. This may lead to agents leaving their present coalitions if the rate is lowered to certain points. They empirically show that the model brings stability to the dynamic system

and customers are satisfied all the time. The trade between highest trusted agents are more beneficial to individual agents and market efficiency.

The encounter between sellers and buyers so far assume sellers have enough products in stock. What if sellers run low on their stocks and buyers' demand is still high. Goldman et al. [32, 31] search for appropriate strategies when the shortage on goods is common in the market. In such a situation, sellers may decide to choose particular buyers in order to maximize their individual utility instead of selling products to buyers in the first-come-first-serve fashion without being detected by buyers. At the same time, buyers may choose to approach particular sellers. In their experiment, sellers can choose buyers by $i$) random, $ii$) the size of of the order, and $iii$) the type of buyers. Buyers can choose sellers by $i$) random, $ii$) loyalty, and $iii$) probability. They set up three levels of stock: small, medium and large. The empirical results show that there are equilibria in medium stock size when buyers play loyal strategy and sellers play random strategy. There are also equilibria in large stock size when buyers play probability strategy and sellers play random strategy. Hence, sellers should choose their buyers randomly.

### 2.6.3 Coalitions of Logistics Providers

The idea of deploying intelligent agents in logistics and supply chains is widespread. Work in [61, 27, 28, 104, 118, 111] deploys agents as the representatives to the role of logistics/supply chain management. A management role can be to negotiate the allocation of the tasks to the trucks [110]. Work in [112, 20, 26, 105, 9] deploy agents, who are responsible on certain tasks, to cooperatively plan and schedule for particular systems. Coalition formation can also be be applied to solve complex problems in the logistics domain. Complex logistics in the military domain also deploy multi-agent systems technique as well [2]. Recent research [25] pays attention to the invention of efficient strategies for intelligent agents to manage supply chains on behalf of human beings. However, apart from [89] which we have covered and is our motivation on coalition formation among transportation agents, we explore a small number of related works on coalitions of logistics providers in the following.

Coalitions of real trucks traveling on a highway can also take place and be beneficial. Khan and Boloni [44] study the dynamicity of convoys of trucks using simple agent technology. Trucks can form a convoy in a random fashion, two trucks gathering together while traveling. A truck can join and leave the convoy both by incidence or at will. Forming convoys on highway is an important problem because it has economic effects, e.g., overtaking or influence of traffic signs can directly affect the fuel consumption of the trucks. In this research, simple agents, each of which is attached to a truck, are able to detect the vehicle

speed, location, and have limited communication ability with other agents. Each vehicle is still controlled by the driver, who will be advised by the agent what the person should do, such as accelerate, decelerate, join a convoy or leave the convoy. The results from [44] suggest that forming convoys can deliver safer and better coordinated traffic. Furthermore, their algorithm can also suggest how each convoy should be structured, i.e. the distance between vehicles, which is related to the traveling speed. This work pays more attention on the behavior of individual agents in the coalition rather than the behavior of the whole system.

The Contract Net protocol [94], described by Smith, is the foundation solution for cooperation among agents. It allows agents to negotiate by sending messages among them. Han, Gu, Li, Yin and Zhang [34] improve the performance of the protocol with global information and applying iterative optimization. The domain of study of [34] is the planing and task allocation, which is usually done by a human being, among agents. Their experiment [34] was done in a multi-agent system environment, in which there are shipping and truck agents. Shipping agents have to allocate tasks, with respect to constraints imposed by customers, to truck agents who will simply transport the allocated goods. Note that a truck agent will operate solely with a shipping agent. Once given a job request, a shipping agent will consider the most appropriate truck for the job. This operation is referred to as a vertical cooperation. Then the initial plan will be generated. Note that optimizing the plan at this point is a complex knapsack problem.[2] In order to solve this, Han, Gu, Li, Yin, and Zhang [34] extend the Contract Net protocol to have "temporary grant" and "temporary reject" messages. Then they apply an auction protocol to allocate tasks among truck agents. Providing that they have time remaining, the plans can be re-estimated. Furthermore, the algorithm allows for "horizontal cooperation", in which the cooperation among the shipping agents can be conducted.

Rehak, Volf and Pechoucek [72] study cooperative task allocation among self-interested agents in a transportation domain. The goal of the system is to deliver humanitarian aid from various sources to a destination. The resources are to be delivered by self-interested agents who possess trucks. These agent are reluctant to share their private information such as capacity, availability, location, etc. There is a human being manager who is responsible for dispatching the resources. Rehak et al. [72] argue that existing agent-based technology in transportation allocation is not efficient because they are only applicable to small problems and are not scalable. Their contribution is to accommodate optimization technology of operation research to multi-agent systems domain. Given a task of transportation, their

[2] "Given a set of items, each with a weight and a value, determine the number of each item to include in a collection so that the total weight is less than a given limit and the total value is as large as possible." Ref: http://en.wikipedia.org/wiki/Knapsack_problem, accessed on 8 May 2009.

algorithm will generate an initial plan, which will be further optimized during the abstract plan using advanced technology (such as a graph plan [34] algorithm). The available plans will be bidden by self-interested agents using Contract Net protocol [94]. After that the plans of each agent will be evaluated and verified before being executed. The results show that the overall performance is improved due to reduction in communication, increase in parallelism (of high performance optimization), etc.

At this point, we have seen coalitions among buyers, coalitions among sellers, coalitions between sellers and buyers, and coalitions among logistics providers. However, transactions in supply networks are committed by buyers, sellers and logistics providers. The forms of coalitions we have seen so far seem to be inadequate that they cannot be used to solve this combined problem. This is a direction which our research takes in chapter 6.

## 2.7 Qualitative Coalition Formation

Coalition formation discussed so far is based just on coalition value, which is regarded as "quantitative" coalition formation [114]. What about "qualitative" coalition formation? There is also another setting in the literature, where agents are assigned goals, which they have to complete. Forming coalitions can help agents to achieve more goals. Achieving more goals is considered, in this respect, achieving better quality. Although being far less studied, there are mechanisms to capture quality of the coalition, such as *effective function* [1]. In the following, we will explore how coalition formation works in this situation.

One of the early works in coalition formation among goal-oriented agents is the study of overlapping coalitions by Shehory and Kraus [86, 88]. Whereas typical coalition formation work allows each agent to be part of a single coalition only, Shehory et al. [86, 88] argue that task-oriented agents can be given a set of tasks, each of which requires cooperation among agents and needs to be executed in a certain order, to complete. Limitations to cooperation among agents can waste resources that agents in coalitions may be able to utilize. Thus, it makes sense for the agents to have overlapping coalitions. They propose algorithms that allows agents to form coalitions and jointly execute their tasks, disband the coalitions, and form coalitions for their remaining tasks again. Given that task allocation problem is of class non-polynomial complexity, they manage to propose a polynomial complex algorithm with sub-optimal results. The algorithm is also an anytime algorithm and yields good results.

Wooldridge et al. [114] propose the idea of "qualitative coalitional games". In this work, there is a set of goals, some of which will be assigned to each agent. Each agent will be "happy" if any goal can be achieved, i.e., the agent has "gained something". There is no

preference over the set of goals assigned to each agent. The agent is satisfied as long as a goal is achieved. In their setting, an agent does not aim at any goal in particular and there is no need for the agent to maximize the number of goals it needs to achieve. A coalition is a collection of agents and has a set of "choices", from which the coalition may decide to act. Taking a choice simply means a number of certain goals satisfy agents in the coalition. It is up to an agent which coalition it would like to form, with respect to its goals. Given such a setting, the goal which the research [114] pursues is to derive the computational complexity. They identify 14 decision problems, including "successful coalition", "selfish successful coalition", "minimal coalition" etc. They can assess the computational complexities of those decision problems, whose complexities are mostly of class non-polynomial. Note that the quality mentioned in this work is somewhat related to quantity, i.e. minimal number of goals to be achieve by each agent is one.

Based on the previous research, Wooldridge et al. [113], study "coalitional resource games", which is a variation of "qualitative coalitional games" [114]. In this new setting, agents require some resources in order to finish their tasks. Each task requires at least one resource and agents are given a set of resources. One real world of such a setting is a group of academics who set up goals to finish their research. They need necessary resources to conduct their experiments. These resources are limited, thus sharing them among researchers is inevitable. This raises the question of which agent should be given access to the resources. This is known widely in AI and multi-agent systems research as *resource allocation problem* [38]. In their research, Wooldridge et al. [113] have classified the complexity of ten decision problems related to "coalitional resource games". They have also shown the relationship between this work and the previous one. Again, most of the problems are of non-deterministic polynomial complexity.

Ieong and Shoham [39] study coalition formation among agents where attributes of the agents designate the values of coalitions. They argue that it is very common in the real world that each agent possess some qualities, referred to as attributes, necessary to execute its task. In other words, their attributes constitute the coalition value. Such settings include a football team which has a set of players who possess various skills needed for winning a game, or a mining company possesses a number of minerals which are to be optimally produced to maximize the company's profit. Ieong et al. [39] define a formal representation of their games which is very succinct and is applicable for other settings. They can also apply the Shapley value and the Core to their game. Furthermore, they can also propose a heuristic for computing Shapley value for a large problem.

## 2.8 Other Coalition Formation Work

We have covered coalition formation works that are closely related to this thesis. However, there are many other works in coalition formation that are interesting, although they are less related to this thesis. We shall explore them briefly in the following.

Rather than just deciding on the spot whether they should cooperate or not, research suggests that agents should learn from their cooperation in the past as well. Merida-Campos and Willmott [58] propose a model for forming coalitions over a period of time. The model allows for various strategies to be deployed by agents. Based on the repeated coalition formation, Gerber and Klusch [29, 45] propose a scheme for agents to dynamically form coalitions. They manage to apply a technique to ubiquitous computing such as m-commerce. Soh and Li [95, 97] propose a learning mechanism for agents while forming coalitions. Agents apply an enforced learning approach to interact more efficiently in a complete information environment. Chalkiadakis and Boutilier [16] study learning coalition formation under uncertainty by applying a Bayesian technique. They also introduce the ”Bayesian Core” as a new solution concept. Lastly, Pechoucek, Mark and Brta [65] apply a knowledge-based approach to coalition formation to reduce the complexity of the process as well as to retain the privacy of the agents involved.

Other aspects studied in coalition formation include economic analysis [15, 6], deception-free model [11] where agents may not reveal the whole truth, information retrieval system [46, 91], power transmission planning [18], security on e-commerce [116, 17], information sharing [41], sensor network [93], robotic soccer [3], physics-oriented system [90, 52] where principles of physics are applied, market modeling [19], and policy evaluation [64].

## 2.9 Motivation to the Thesis and Research Question

In the following, we will discuss the motivation that lead us to research in the thesis. We then raise research questions that we try to solve in later chapters of the thesis.

### 2.9.1 Motivation

We have covered coalition formation in considerable details. It is a reasonable assumption that applying coalition formation to any cooperative system, where agents may be self-interested, can benefit the system a lot. While many solution concepts, such as the Shapley value and the Kernel, can bring the system to a stable state, one in which agents do not change their strategies, by ensuring individual rationality, the Core can also bring the wealth

(in addition to stability) to the system as well. Hence, the Core seems the most appropriate solution concept be applied in any cooperative system.

However, bringing about the core to a system can be a difficult task due to several reasons. In a superadditive environment, although agents can form the grand coalition for maximizing the system's wealth, it has been shown that the Core may be empty [30]. In a non-superadditive environment (where coalition values are arbitrary and are independent of their respective coalition sizes), finding the maximal wealth for the system is to compute the maximal sum of coalition values from all coalition structures in the system. This has been proved that it is a hard problem [77] (as we will explore in greater details in section 2.3), let alone the optimization for coalition values which can be by itself a hard problem. Furthermore, agents are practically rationally-bounded and are under various real world constraints, such as time, resources, etc., it is such an important challenge to find a solution that can reach the closest state possible to the Core.

We take the challenge in this research–to deliver a solution which will lead to the closest state possible to the Core. Whereas previous studies in coalition formation merely deal with a small number of agents, assuming coalition values known a priori (with characteristic function), etc., we aim at conducting research in coalition formation under an environment which is as close to real world setting as possible. Rather than binding ourselves to classical theories, we are investigating for algorithms that can help coalition formation be more practicable and useful in real world settings. In particular, we want to explore coalition formation which the system's utility is maximal via optimal coalition structure algorithm in various settings.

### 2.9.2 Research Questions

Whereas the state of the art [69, 71] scan all the inputs and can handle only 27 agents, the question is how can we manage to find algorithm that is scalable to a larger number of agent and can yield reasonably good results in anytime fashion. This idea is not new at all in solving hard problems, such as traveling salesman or vehicle routing problems. In this environment, optimal results cannot be guaranteed but the reasonably good result achieved within the limited time is acceptable. The performance of algorithms for these hard problems are evaluated by benchmarking against a standard input set. Since, searching for OCS is a hard problem [77], we are interested in investigating for a solution to cope with the scalability issue of the problem and can yield reasonably good results in anytime fashion. Given the same setting which is quite generic, we want to test the solution as thorough as possible within various data distribution.

Given that the first problem has been solved, the next question remains how the solution can be applied to any problem in particular, given that coalition values are not know a priori? We adopt the classical game of linear production as our test bed for this question. Whereas the original problem deal with superadditive domain, we consider non-superadditive domain, which is more complex and raises another level of difficulty to the problem. Although linear production is considered not a hard problem by itself, solving it can be time consuming given a large number of agents involved. Note that existing algorithms in OCS problem require all the input be scanned for their values. When the number of agents become larger than 30 agents, it seems impossible for any typical computer to handle the problem. Many of previous work in coalition formation have deploy "best first" strategy to form coalitions, we follow this direction.

We go further in terms of raising complexity, which is realistic, into OCS problem. The next level is how can we handle OCS problem when optimizing coalition value itself is a hard problem, let alone the complexity of solving OCS. We investigate into OCS in the non-linear domain where agents have to solve for the (near) optimal solutions (values) of coalitions. We consider a domain where independent truck agents are to cooperatively distribute goods from sources to various destinations. There are two levels of complexities: $i$) solving the hard problem of distributing goods from each source, and $ii$) solving the the optimal allocation of trucks to each sources, which is a OCS problem, such that the total cost of the system is minimal.

We have seen that coalition formation has been explored separately in various parts of the supply chains domains, such as coalitions among buyers, coalitions between buyers and sellers, coalition among logistics providers. It is interesting to us that if we combine these parties together, how can we find a way to optimize the utility of the system? We want to address this problem via optimal coalition structures. The challenge in this problem is that it involves 3 different parties, which none of any work in coalition has attempted before.

Thus the objectives of this research are:

1. to develop a best-first, anytime algorithm that is an efficient solution for OCS problems in environments where coalition values are known a priori Then, to adapt the algorithm developed in objective 1:

2. to solve OCS problems in a linear environment where coalition values and coalition structure values are not known a priori but must be calculated

3. to solve OCS problems in an NP hard, non-linear environment where coalition costs and coalition structure values are not known a priori

4. to solve OCS problems in complex environments such as those in which coalitions

involve 3 types of stakeholders, such as in the supply chain domain

# Chapter 3

# Computing Optimal Coalition Structures

## 3.1 Introduction

The previous chapter has identified 5 objectives of this research. This chapter will address the first of those objectives, namely: 1. to develop a best-first, anytime algorithm that is an efficient solution for OCS problems in environments where coalition values are known a priori

Computing optimal coalition structures in multi-agent systems is an important research problem both from theoretical [42] and practical perspectives. The optimal coalition structure problem seeks to identify, given a set of agents and a value to each subset, the optimal partitioning of that set of agents (i.e., a partitioning for which the sum of the coalition values is maximized). The optimal coalition structure problem finds application in a variety of real world settings, including logistics and supply chains [75, 85, 99], virtual organizations, [33] team formation, [62] etc. This problem is proved to be NP-Hard [77].

The number of all coalition structures can be determined by *Bell Number*, $B_n$ [50]. Since the value of $B_n$ can be very large for a small value of $n$, existing algorithms tend to divide the search space into small portions. Algorithms for solving this problem can be divided into two main categories: $i$) anytime algorithms that perform exhaustive search to generate the optimal solution [77, 22], and $ii$) heuristic algorithms that do not provide a guarantee of generating an optimal solution, but in ideal settings, generate a near-optimal solution [100, 70]. Algorithms of the first kind guarantee that a solution obtained after completion of a certain part of search space would be within a bound of the optimal. Algorithms of the second kind do not provide any guarantees of optimality at all, but provide empirical evidence to suggest that the solutions obtained are indeed near-optimal.

In the research reported here, we have sought to develop an algorithm (that we shall refer to as the coalition bound heuristic or CH). This work presents a best-first anytime algorithm for computing optimal coalition structures. Our algorithm differs from others in

two ways. First, the approach is novel in that it generates coalition structures based on coalition values, while existing algorithms base their generation on the structure (members and configurations) of coalitions. With our algorithm, coalition structures are generated by repeatedly choosing the best coalition, as determined using a novel metric called *agent contribution ratio* that we define. Second, our algorithm can proceed towards the optimality beyond the boundary of the partitioning structure (such as configuration) of the whole search space. The agent contribution ratio will identify (independently of the partitioning structure) the best candidate coalition, which contributes the most value to $CS$ and will be placed in CS. As the consequence, the algorithm can converge to (reach) optimality quickly.

We have compared the performance of our algorithm against that of Rahwan et al [69, 71] using 20 data distributions. Our results show that our algorithm always converges on an optimal coalition structure faster. Although our algorithm terminates later (because of a simple prune mechanism being used) in some cases, our algorithm always yields a better, or, at least, as good solution as the algorithm of Rahwan et al.

The content of this chapter structured as follows. First, we explain how the algorithm generates all $CS$s. We introduce the novel idea, the agent contribution ratio and followed by the data structure used. We then explain the main and the working functions composing the whole algorithm which guarantees the completeness (all $CS$s are generated) and systematicity (each $CS$ is generated only once). Second, we explain how the algorithm converge to optimality. We discuss applying branch and bound technique which can accelerate the algorithm to converge to optimality quickly. We follow with giving an example of an execution of the algorithm. Lastly, we show empirical results.

## 3.2 The CH Algorithm

In contrast to other algorithms [77, 51, 22, 70, 69, 71], this work generates coalition structures based on coalition values rather than their coalition members. We consider generating coalition structures as a process of repeatedly choosing the best coalition (i.e., one that contributes the best value to the coalition structure) from available candidates such that for each generated coalition structure $i) \bigcup S_i = N$ (the *exhaustive* condition [77, 51, 22]) and $ii) S_i \cap S_j = \emptyset$ for $i \neq j$ (the *disjoint* condition [70, 69, 71]). The algorithm must be *systematic*, i.e., it must not generate/evaluate the same coalition structure more than once.

**Best Coalition:** In subadditive environments [51, 42], $V_{S \cup T} < V_S + V_T$, coalition values are inversely proportional to their cardinalities. Clearly, any optimal coalition structure would be composed of small coalitions because they individually have high values.

Any algorithm that prunes larger coalitions can rapidly approach the optimal solution. It is more complex in superadditive environments where $V_{S \cup T} \geq V_S + V_T$. In settings where $V_{S \cup T} = V_S + V_T$, and all coalition values are $k * |S|$ (for any constant $k$), where $k \in \mathbb{Z}^+$, all coalitions structures are optimal. In other superadditive systems, where $V_{S \cup T} > V_S + V_T$, a coalition structure consisting only of the grand coalition is always optimal. Both subadditive and superadditive environment are similar in that there is a monotonic relationship between coalition values and their cardinalities.

However, the problem is difficult in the general case [51], because such monotonicity does not exist–coalition values do not have any relationship with cardinalities. The highest value coalition may not be the best coalition because it might be a large coalition that leaves little room for other coalitions (whose values might be very low.) Hence, we need a metric that would permit us to pick the next coalition to add to an (incrementally constructed) coalition structure. We define a metric called *agent contribution ratio to coalition*, $\overline{a}_S$, i.e., the average value for each agent in $S$:

$$\overline{a}_S = \frac{V_S}{|S|}$$

and use this as a basis for our best-first search. We note in passing that the *agent contribution ratio to coalition structure*, $\overline{a}_{CS}$, i.e., the average value for each agent in $CS$:

$$\overline{a}_{CS} = \frac{\sum V_{S_i}}{n}$$

will always be maximal for any optimal coalition structure.

In order to be efficient in random environments, we define the best coalition, $S^*$, is the one whose $\overline{a}$ is the highest. Given two coalitions $S_1$ and $S_2$ and their respective agent contributions $\overline{a_1}$ [1] and $\overline{a_2}$, we define $S_1$ is the best coalition (with respect to the algorithm) if:

- $\overline{a_1} > \overline{a_2}$ (the agent contribution ratio of $S_1$ is higher ), or
- $\overline{a_1} = \overline{a_2}$ and $|S_1| < |S_2|$ (the agent contribution ratios are equal but $S_1$ is smaller), or
- $\overline{a_1} = \overline{a_2}$ and $|S_1| = |S_2|$ (the agent contribution ratios and sizes are equal but $S_1$ is merely precedent to $S_2$).

We shall refer to these properties as the *coalition contribution* properties. Given a number of agents, the algorithm will determine the best coalition and use it to construct a coalition structure.

[1] We use $\overline{a_1}$ instead of $\overline{a_{S_1}}$ simply to avoid it looking clumsy.

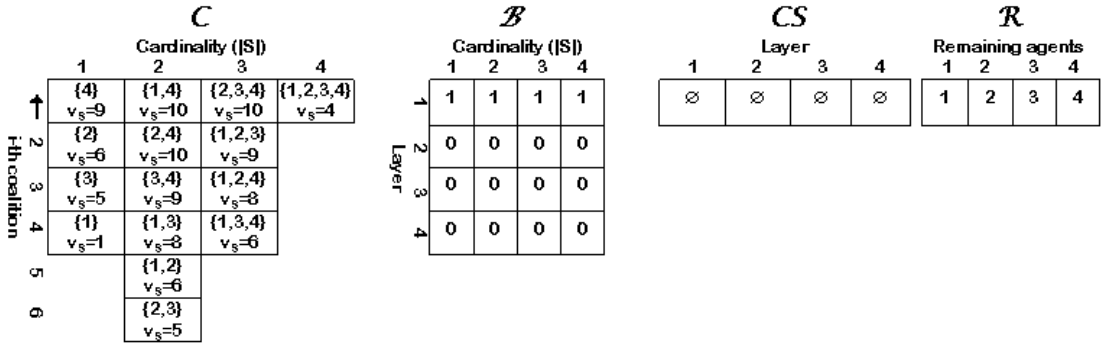


Figure 3.1: **Data Structure** Coalitions are stored in 2-dimension array $\mathcal{C}$. Available candidate coalitions for all layers are kept tracks by 2-dimension array $\mathcal{B}$. The CS being constructed is kept in 1-dimension array $\mathcal{CS}$. The remaining agents, which can be candidates for the best coalition at the present layer $l$ of $\mathcal{CS}$, are kept track by 1-dimension array $\mathcal{R}$.

**Data Structures:** Here, we define data structures to facilitate the generation below. Firstly, we need a data structure to store the coalition structure being generated. We define $\mathcal{CS}$ a 1-dimensional array of coalitions, whose size is $n = |N|$, the maximal number of coalitions the coalition structure may contain. Each element is for a coalition chosen so far. $\mathcal{CS}$ represents $CS$ and will be used interchangeably. We shall refer to the $l$-th index of $\mathcal{CS}$ as the $l$-th *layer* coalition. Elements $\mathcal{CS}[1]$ and $\mathcal{CS}[2]$ are the coalitions at the $1^{st}$ and the $2^{nd}$ layer of the coalition structure respectively. Since it is common in the literature that all coalitions must be observed before the algorithm can really proceed efficiently [77, 51, 22, 101, 100, 70, 69, 71], we store all coalitions in a 2-dimensioned array, $\mathcal{C}$. The first dimension, depicted in Figure 3.1 as columns, refers to cardinalities of coalitions. The second dimension, depicted in Figure 3.1 as rows, refers to the $p$-th coalition of a given $c$ cardinality. Coalitions in each cardinality will be sorted by their value in descending order, i.e., the $1^{st}$ position is the highest value or the best coalition, the $2^{nd}$ position is second highest value or the second best coalition, and so on. Element $\mathcal{C}[1][1]$, for example, refers to the $1^{st}$ coalition (or position $p = 1$) of cardinality 1, i.e., $S = \{1\}$ in Figure 3.2, while $\mathcal{C}[2][2]$ refers to the $2^{nd}$ coalition (or position $p = 2$) of cardinality 2, i.e., $S = \{2, 4\}$ in Figure 3.1, and so on.

As the algorithm proceeds, some agents have already been placed in $\mathcal{CS}$ and are not available anymore. In order to maintain the *disjoint* condition, we define $\mathcal{R}$ a set of *remaining* agents who are available for being chosen for the present layer $l$ in $\mathcal{CS}$. A new $CS$ is completely generated once $\mathcal{R}$ is empty, i.e., the *exhaustive* condition is satisfied. Each coalition in $\mathcal{CS}$ is the member of $CS$. Furthermore, the algorithm needs to know what coalitions are

available for $\mathcal{CS}[l]$. We define $\mathcal{B}$ a 2-dimensional array of integers for indexing coalitions in $\mathcal{C}$ that are available for $\mathcal{CS}[l]$. The first dimension, depicted in Figure 3.1 as rows, refers to the $l$-th layer of $\mathcal{CS}$. The second dimension, depicted in Figure 3.1 as columns, refers to the cardinality $c$ in each layer. The value of element $\mathcal{B}[l][c] = p$ indicates that the candidate coalition for $\mathcal{CS}[l]$ from cardinality $c$ is the element $\mathcal{C}[c][p]$. At layer $l$, all coalitions available for $\mathcal{CS}[l]$ are indexed by all elements in row $l$ of $\mathcal{B}$. These coalitions are, of course, subset of $\mathcal{R}$. We shall refer to each of these coalitions as a *candidate* coalition of its respective cardinality $c$ for $\mathcal{CS}[l]$. Consider $\mathcal{B}$ in Figure 3.1, for example. Element $\mathcal{B}[1][1] = 1$, refers to the candidate coalition from cardinality 1 for $\mathcal{CS}[1]$, which is the 1-st coalition, i.e. $\mathcal{C}[1][1] = \{4\}$. Element $\mathcal{B}[3][2] = 6$ refers to the the candidate coalition from cardinality 2 for $\mathcal{CS}[3]$, is the 6-th coalition, i.e. $\mathcal{C}[2][6] = \{3, 4\}$. The value 0 of any element in $\mathcal{B}$ implies that there is no candidate coalition for the specified layer from the respective cardinality.

**Algorithm 1 Main** Construct coalition structures by adding the best coalition (chosen from available candidate coalitions) into $\mathcal{CS}[l]$

```
1:  l ← 1                                          ▷ set the present layer l to 1
2:  S* ← chooseNextS(l)                            ▷ choose the best coalition S* for layer l
3:  while S* ≠ ∅ do                                ▷ while S* exists
4:      CS[l] ← S*                                 ▷ place S* in CS[l]
5:      R ← R \ S*                                 ▷ update R by removing S*
6:      S* ← ∅                                     ▷ reset S* to ∅
7:      if R = ∅ then                              ▷ if there is no agent left in R
8:          print "new CS generated: "+CS;         ▷ output new CS
9:      end if
10:     S* ← Extend()                              ▷ attempt to extend CS
11:     if S* != null then                         ▷ if S* is found then CS can be extended
12:         l ← l + 1;                             ▷ extend CS to the next layer
13:     else
14:         S ← Alter()                            ▷ cannot extend then attempt for altering
15:         if S* = ∅ then                         ▷ cannot alter
16:             S* ← Shrink()                      ▷ attempt to shrink
17:         end if
18:     end if
19: end while
```

### 3.2.1 Main Function

**Populating Data:** At the beginning, $\mathcal{C}$ is populated. Its elements in each cardinality are sorted by their values in descending order. Sorting coalitions in each cardinality can be done in parallel using any efficient sort algorithm. In our implementation, we use *Merge*

sort algorithm [2], which is very robust and efficient because its worst case time complexity is among the best, i.e., $m \log m$ where $m$ is the size of input. Since sorting is done in parallel, the largest value of $m$ is ${}^{n}C_{\lfloor n/2 \rfloor}$ which is slightly less than scanning all $2^n - 1$ coalitions. The space complexity is also reasonable that it is $O(m)$. (One may argue that sorting can be costly to the performance of the algorithm. The empirical results show that, taking into account the sorting time, our algorithm still converges much quicker in all data distributions.) All the coalitions in $\mathcal{C}$ are sorted by their values rather than their members as in Rahwan et. al [70, 69, 71]. The first coalition of each cardinality is the best, i.e., highest value, and so on. All elements of both $\mathcal{B}$ and $\mathcal{CS}$ are initialized to 0 and $null$ respectively. Then, $l$ is set to 1 indicating that the coalition structure is being built at layer 1 as the starting layer. At each layer $l$, the algorithm will determine what are the candidate coalitions. At the beginning, the first coalition in each cardinality is its candidate. Thus, $\mathcal{B}[1][1 \leq c \leq n]$ is set to 1, indicating that the candidate coalition for $\mathcal{CS}[1]$ of each cardinality is its first coalition. $\mathcal{R}$ is set to $N$ because none of the agents are placed in $\mathcal{CS}$.

**Main Loop:** The logic of the main loop is very simple. It keeps acquiring $S^*$ and place it in $\mathcal{CS}$ at the present layer $l$. Firstly, acquiring $S^*$ is done by calling function $ChooseNextS$, which determines the best coalition from available candidates at the present layer $l = 1$. Then the algorithm determines if it can acquire anymore $S^*$, which can be done in one of the following manners

- **Extend** the algorithm tries to extend $CS$, i.e. place the next $S^*$ in the $\mathcal{CS}[l+1]$,
- **Alter** the algorithm tries to alter $\mathcal{CS}[l]$ with its next best candidate coalition, or
- **Shrink** the algorithm repeatedly tries to remove $\mathcal{CS}[l]$ (while $l > 1$) and replace $\mathcal{CS}[l-1]$ with its next best candidate coalition.

At the beginning of the execution, since the first coalition $S^*$ has just been chosen, the algorithm enters the main loop in this first round and places $S^*$ at layer $l = 1$ of $\mathcal{CS}$. Remaining agents $\mathcal{R}$ is subtracted by $S^*$ because the agents who are the members of $S^*$ cannot be part of the next coalition. This will guarantee the disjointness of the coalition structure [70, 69, 71]. $S^*$ is then reset to null. After that the algorithm determines whether a new coalition structure has been generated by examining if agents are exhaustively used in $\mathcal{CS}$, i.e., to examine whether $\mathcal{R}$ is empty. If that is the case, the algorithm outputs the newly generated $CS$. The algorithm then tries to generated remaining $CS$s by keeping acquiring the next $S^*$ to fill in $\mathcal{CS}$. First, it tries to extend $CS$ (adding the best coalition to $\mathcal{CS}$), by calling function $Extend$.

[2] Ref: http://en.wikipedia.org/wiki/Merge_sort, accessed 12 May 2009.

If a non-empty coalition is returned, the layer $l$ is increased by 1 and the execution will go to the start of the loop. If $Extend$ returns null, the algorithm tries to acquire the best coalition by calling $Alter$ function. If a non-empty coalition is returned, the execution will go to the start of the loop. If $Alter$ returns null, the algorithm tries to acquire the best coalition by calling $Shrink$ function after which the execution will go to the start of the loop. At the start of the loop, the algorithm examines the value of $S^*$ whether it will continue in the loop. The algorithm terminates when it cannot find any more $S^*$. The 5 working functions that are used to support the main algorithm will be discussed below.

Note that variables defined in the main algorithm are accessible by supportive functions whereas variables defined in supportive functions are local.

### 3.2.2 Working Functions

**Algorithm 2** ChooseNextS Function

```
1:  function CHOOSENEXTS(l)
2:      bestS ← ∅
3:      ā* ← 0
4:      for c = 1 to |R| do                          ▷ for each valid cardinality
5:          if B[l][c] > 0 then                      ▷ if there is a candidate coalition
6:              ā ← V_{C[c][B[l][c]]}/c              ▷ compute the candidate's ā
7:              if ā > ā* then                       ▷ if the new ā is better than ā*
8:                  bestS ← C[c][B[l][c]]            ▷ set the candidate coalition as the new best
    coalition
9:                  ā* ← ā                           ▷ set ā* to the new value
10:             end if
11:         end if
12:     end for
13:     return bestS
14: end function
```

**Choosing (Next) Best Coalition (ChooseNextS):** Since we have sorted coalitions by their values in each cardinality, this guarantees that at each cycle in the main loop the best coalition, one with the highest $\overline{a}$, can be identified from candidate coalitions without ambiguity. The search for the best coalition is very simple. Firstly, the best coalition (local variable $bestS$)is set to empty as well as its agents' contribution ($\overline{a^*}$) is set to 0. The algorithm then goes through each candidate coalition in ascending order of the cardinality and compares its agents' contribution, $\overline{a}$ against $\overline{a^*}$. Only if $\overline{a} > \overline{a^*}$ then the respective candidate coalition is set to be the new $bestS$. This way, even multiple candidate coalitions have exactly the same $\overline{a}$, only the smallest coalition remains the best coalition.

**Algorithm 3** NextS Function

```
1: function NEXTS(c,p)
2:     for j = p to ^nC_c do                    ▷ starting from position p towards ^nC_c
3:         if C[c][j] ⊆ R then                  ▷ if the coalition at p is in R
4:             return j                         ▷ return its position
5:         end if
6:     end for
7:     return 0
8: end function
```

**Search for Next Candidate Coalition (NextS):** At any layer $l$, the algorithm needs to prepare the candidate coalition in each valid cardinality. A valid cardinality is one whose value is not greater than the number of remaining agents, i.e. $|S| < |\mathcal{R}|$. For each of these cardinalities, we just need only the next available coalition, i.e., the one whose members are all in $\mathcal{R}$ with the highest value, as the candidate of its cardinality for the next layer of $\mathcal{CS}$. (The value of each of these candidate coalitions can be used to determine the upper bound of the optimality in order to decide whether the algorithm should terminate. We shall cover this in later sections.) The search for the candidate will be done towards the last coalition(position) in each cardinality. As the algorithm proceeds through the main loop, there might not be candidate coalitions left in some of the valid cardinalities because at least one of the members of all of the remaining coalitions is already in $\mathcal{CS}$. In this case, the respective element of $\mathcal{B}$ is assigned the value 0. Hence, the search is needed only if there is a chance to find a candidate coalition, i.e., the value of the respective $\mathcal{B}$ is greater than 0.

**Algorithm 4** Extend Function

```
1: function EXTEND
2:     if l < n then
3:         for c = 1 to |R| do
4:             p ← B[l][c]          ▷ set the beginning position for searching for candidate
5:             if p > 0 then        ▷ only cardinalities that have coalitions left
6:                 if c = |CS[l]| then      ▷ for candidate of cardinality of CS[l]
7:                     p ← p + 1            ▷ begin the search at the next position
8:                 end if
9:                 B[c][l+1] ← NextS(c, p)  ▷ search for the next candidate of cardinality c
10:            end if
11:        end for
12:        return ChooseNextS(l + 1)        ▷ acquire the best coalition and return it
13:    end if
14:    return ∅
15: end function
```

**Extending Coalition Structures:** The algorithm examines whether there will be a new coalition which will extend (be inserted into the next layer $l+1$ of) $\mathcal{CS}$. The present $\mathcal{CS}$ can be extended if $l < n$ holds. In addition, the cardinality of the new coalition must not be larger than the number of the remaining agents, i.e., for each cardinality $c$, $c \leq |\mathcal{R}|$. For each of these cardinalities, we just need only the next available coalition, i.e., the one whose members are all in $\mathcal{R}$, as the candidate of its cardinality for the next layer of $\mathcal{CS}$. We set $p$ to $\mathcal{B}[l][c]$ the position of the candidate of the respective layer and cardinality. The search begins at *i*) $\mathcal{C}[c][p+1]$, the next position $p+1$ of the cardinality $c$, if $c = |\mathcal{CS}[l]|$ ($\mathcal{CS}[l]$ is just chosen from this cardinality), or *ii*) $\mathcal{C}[c][p]$ otherwise. The search is done through calling the function $NextS(c,p)$, where $c$ is the cardinality on which the search will be done, $p$ is the starting position of the search in $\mathcal{C}[c]$. Once a proper coalition is found, its position is returned and will be assigned to $\mathcal{B}[c][l+1]$ as the candidate coalition. Otherwise the element $\mathcal{B}[c][l+1]$ will be assigned 0 to indicate that there is no more candidate in this cardinality. The algorithm acquires the best coalition from available candidates by calling $ChooseNextS()$ and returns it.

In the case $l >= n$, the algorithm simply returns $\emptyset$

**Algorithm 5** Alter Function

```
1: function ALTER
2:     R ← R ∪ CS[l]                          ▷ return the last coalition of CS to R
3:     p ← B[|(CS[l])|][l] + 1                ▷ start to search for the alternative candidate at the next
   position
4:     B[|CS[l]|][l] ← NextS(|CS[l]|, p)      ▷ retrieve the alternative candidate
5:     CS[l] ← ∅                              ▷ reset CS[l]
6:     return ChooseNextS(l)                  ▷ acquire the best coalition and return it
7: end function
```

**Altering the Body:** This function is called when the algorithm cannot acquire $S^*$ by calling function $Extend$, i.e. cannot extend $\mathcal{CS}$ from layer $l$. Then the algorithm tries to alter $\mathcal{CS}$ by discarding its last coalition and tries to acquire the next best coalition from available candidates of this layer $l$. Before it can actually do that, it has to ensure if there is any candidate coalition beneath the discarded coalition in the same cardinality. However, it has to return the member of $\mathcal{CS}[l]$ back to $\mathcal{R}$ before it can begin searching. The starting point of the search is simply the next position of the last coalition, $\mathcal{B}[|\mathcal{CS}[l]|][l]+1$. Again, the search is done through the call of function $NextS()$. Similar to the attempt to extend $\mathcal{CS}$, the candidate will be assigned to $\mathcal{B}[|\mathcal{CS}[l]|][l]$ if there is one, otherwise 0 will be assigned to the element. The last coalition of the coalition structure is discarded, $\mathcal{CS}[l]$ is set to empty. The next best coalition will be acquired through the call of function $ChooseNextS(l)$ and

will be returned.

**Algorithm 6** Shrink Function

```
1: function SHRINK
2:     if l > 1 then
3:         l ← l − 1                                   ▷ shrink by decreasing value of l by 1
4:         R ← R ∪ CS[l]                               ▷ return members to R
5:         p ← B[|CS[l]|][l] + 1      ▷ set the starting position for the search for alternative
   candidate
6:         B[|CS[l]|][l] ← NextS(|CS[l]|, p)           ▷ search for the alternative candidate
7:         CS[l] ← ∅;                                  ▷ reset CS[l]
8:         return ChooseNextS(l);                      ▷ acquire the best coalition and return it
9:     end if
10:    return ∅
11: end function
```

**Shrinking Coalition Structures:** This function is called when the algorithm cannot find $S^*$ to extend $\mathcal{CS}$ to the next layer $l+1$ nor to alter $\mathcal{CS}[l]$. It then shrinks $CS$ and tries if coalition structure can still be generated by trying to alter the last coalition of $\mathcal{CS}$ each time $\mathcal{CS}$ is shrunk. This can be done only if the condition $l > 1$ holds. Firstly, the algorithm shrinks $CS$ by decreasing the value of $l$ by 1. The members of $\mathcal{CS}[l]$ are returned to $\mathcal{R}$. Then the algorithm tries to acquire the next best coalition of $\mathcal{CS}[l]$ ($CS$ has just been shrunk). The algorithm looks for the next candidate of $|\mathcal{CS}[l]|$. The starting point of the search is set to the next position of the $\mathcal{CS}[l]$, i.e. $\mathcal{B}[|\mathcal{CS}[l]|][l]+1$. The algorithm then searches for the next candidate coalition by calling $NextS$. The last coalition, $\mathcal{CS}[l]$ can now be discarded. The next best coalition will be acquired through the call of function $chooseNextS$ and will be returned.

### 3.2.3 Proof Completeness and Systematicity of the CH algorithm

In the following, we will prove that the CH algorithm generates all CSs (completeness) and generates each of them once and only once (systematicity). For the completeness proof, saying that any given valid CS will be generated by the algorithm is logically equivalent to saying that all CSs will be generated by the algorithm. Given $S_i$ and $S_j$. We denote by $S_i \geq_b S_j$ if $S_i$ is the best (better than $S_j$) coalition. Note that coalitions in each cardinality are generated in lexicographic order. When being sorted, the positions of each pair of coalitions being compared do not change if the values of both coalitions are equal.

Firstly, we need to prove that any given $CS'$ can be rearranged such that above properties hold.

**Lemma:** Let $CS' = \{S'_1, S'_2, \ldots, S'_m\}$, where $1 \leq m \leq n$ be a valid CS, it can be rearrange as $CS = \{S_1, S_2, \ldots, S_m\}$, where $S_1 \geq_b S_2 \geq_b \ldots \geq S_m$.

**Proof:** Since the member of $CS'$ are merely rearranged to make $CS$ such that the coalition contribution properties hold, hence $CS'$ is equal to $CS$ because both are the same sets whose members (coalitions) are the same.

**Theorem:** A given CS will be generated (completeness) once and only once (systematicity) by the CH algorithm.

**Proof:** We will show that $S_1$ will be placed in CS before the end of the execution and other remaining coalitions will be placed in a strict order.

- Let $S_1$ is located at position $1 \leq p \leq^n C_{|S_1|}$ of its respective $C[|S_1|]$. $S_1$ will be chosen as the candidate coalition of its respective cardinality at a point in time before the end of the execution because the $Extend$(line 6-9), $Alter$(line 3-4) and $Shrink$(line 5-6) functions skip the discarding coalition and call the function $NextS$(line 2-6) which searches for the next available coalition down the array $C[|S_1|]$ one by one. Once $l = 1$, $S_1$ will be the candidate of its respective cardinality. The function $ChooseNextS$ will examine the best candidate for $CS[1]$. Since the function will choose the best candidate from the remaining candidates including $S_1$, $S_1$ will always be put in $CS[1]$.

- Let's assume the algorithm has put $1 \leq i < j \leq m$ coalitions in to $CS$. Hence, the next step is to put the best candidate coalition to $CS[j]$. Let $S_j$, $1 < j \leq m$, is located at position $1 \leq q \leq^n C_{|S_j|}$ of its respective $C[|S_j|]$. Similar to the above case, $S_j$ will be chosen as the candidate coalition of its respective cardinality at a point in time before the end of the execution because the $Extend\ Alter$ and $Shrink$ functions will call the function $NextS$ which searches for the next available coalition down the array $C[|S_j|]$ one by one from the discarding position. At layer $j$, $S_j$ will be the candidate of its respective cardinality for $CS[j]$. The function $ChooseNextS$ will examine the best candidate for $CS[j]$. Since the function will choose the best candidate from the remaining candidates including $S_j$, $S_j$ will always be put in $CS[j]$. This property holds for the remaining coalitions for $CS[j], i < j \leq m$.

- Since coalitions are placed in $C$ in strict order, they will be chosen as candidate coalitions of their respective cardinalities, and they will be chosen as the best candidate by the same process in the algorithm, a CS will never be regenerated.

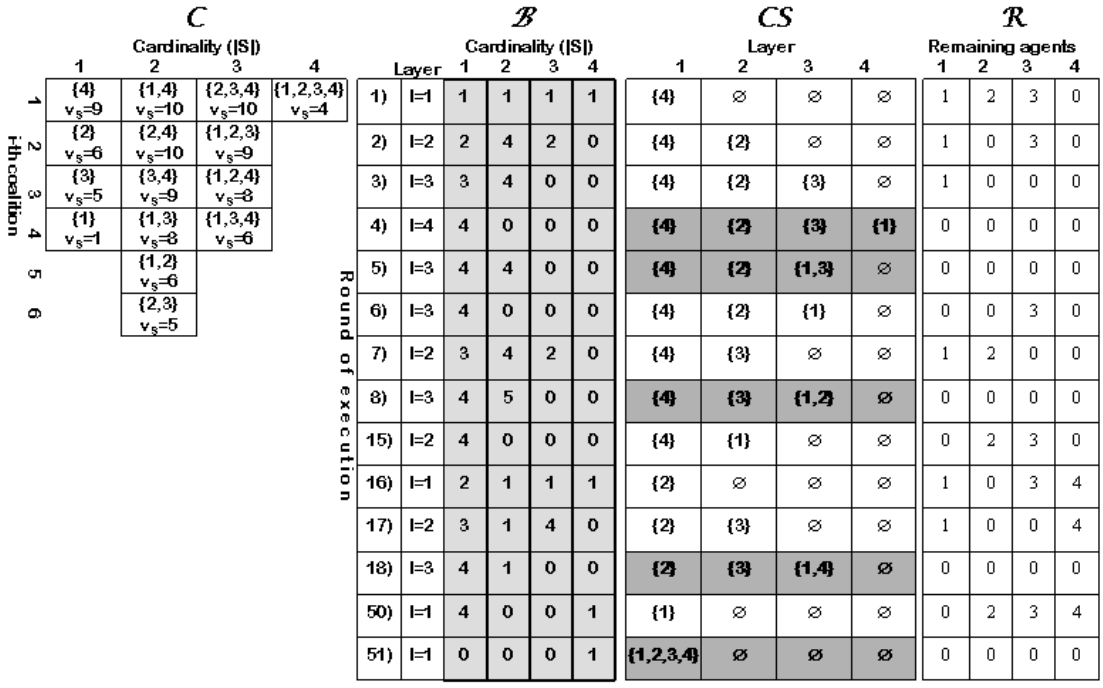


Figure 3.2: **Generating Coalition Structure** Coalitions are stored in array $\mathcal{C}$, where rows represent the position of the coalitions in each cardinality, represented by column. Candidate coalitions for each layer $l$ in $CS$ are stored in array $\mathcal{B}$, whose rows represent the layer of $\mathcal{CS}$ and columns represent the cardinality. Attached to the left of the array are two additional columns. The first one indicates the execution round, while the second one represents the respective layer of $\mathcal{CS}$. The coalition structure is stored in one dimensional array $\mathcal{CS}$. As it appeared here, multiple rows are the current state of $\mathcal{CS}$ with respect to the corresponding execution round appears in $\mathcal{B}$. Remaining agents are stored in array $\mathcal{R}$. Each row represents remaining agents after a candidate coalition has been chosen for $\mathcal{CS}$ in the same execution round in the corresponding rows of $\mathcal{B}$ and $\mathcal{CS}$.

### 3.2.4 Example of Coalition Structures Generation

This section will show how the algorithm runs in the exhaustive search. The CS whose value is the highest is kept as the optimal one until the algorithm finds a better CS. Figure 3.2 illustrates the execution of the algorithm on 4 agents. The data shown in the figure is captured from the real run. The first table on the left depicts coalitions in $\mathcal{C}$. In each cell, the coalition members are shown as a set in the upper half while the coalition value is shown in the bottom half of the cell. Coalitions in each cardinality are sorted by their values in descending order, i.e., the coalition with highest value is in layer 1, the second highest value is in layer 2, and so on. The second table depicts the status of $\mathcal{B}$ during the execution. Each row contains data from each round through the main loop. The first column on the left indicates the round of execution. The second column indicates the present layer of $\mathcal{CS}$ in the corresponding round. The remaining columns are candidate coalitions in $\mathcal{B}[l][1]$ to $\mathcal{B}[l][4]$ from which the best coalition can be chosen. The third table shows that status of $\mathcal{CS}$. Each column represents a corresponding layer in $\mathcal{CS}$. The best coalition from the candidates is placed in the corresponding element of $\mathcal{CS}$. The last table on the right shows the remaining agents in $\mathcal{R}$ after the best coalition is chosen. The algorithm will go through the three main working functions to the end of the loop. In the next round, corresponding data are presented in the next row.

In round 1), where $l = 1$, candidate coalitions are order 1 in each cardinality. Their $\overline{a}$ are $\overline{a}_{\{4\}} = 9/1 = 9$, $\overline{a}_{\{1,4\}} = 10/2 = 5$, $\overline{a}_{\{2,3,4\}} = 10/3 = 3.333$, and $\overline{a}_{\{1,2,3,4\}} = 4/4 = 1$. Hence the best coalition is $\{4\}$ and is placed in $\mathcal{CS}[l = 1]$. The remaining agents are now 1, 2 and 3. The algorithm then determines if $\mathcal{CS}$ can be extended by calling $Extend$. Here, it scans through each cardinality, whose value is not greater than 3, and call $NEXT$ in order to search for the next candidate coalition. In cardinality 1, the candidate coalition is the order 2, i.e., $\mathcal{C}[1][2] = \{2\}$. In cardinality 2, the candidate coalition is the order 4, i.e., $\mathcal{C}[2][4] = \{1, 3\}$. In cardinality 3, the candidate coalition is the order 2, i.e.,$\mathcal{C}[3][2] = \{1, 2, 3\}$. Among these candidate coalitions, $\{2\}$ has the highest $\overline{a} = 6$ and is chosen as the best coalition. This extension repeats until round 4 when $\{1\}$ is placed in $\mathcal{CS}[4]$. The algorithm outputs the newly generated $CS$. Now, it cannot extend $\mathcal{CS}$ anymore because it is at the lowest layer $l = 4$. Hence, it tries to alter the last coalition of $\mathcal{CS}$. Firstly, the algorithm returns $\mathcal{CS}[4] = \{1\}$ back to $\mathcal{R}$, reset $\mathcal{B}[4][1] = 0$ and call $NEXT$ to search for the next candidate in cardinality 1. Since $\{1\}$ is already the last coalition, the alteration cannot be done. The algorithm then tries the last option, shrinking $CS$. It decreases layer $l$ by 1, $l = 4 - 1 = 3$. Coalition $\mathcal{CS}[l = 3] = \{3\}$ is returned to $\mathcal{R}$. It then calls $NextS$ for the successor candidate coalition of $\{3\}$, which is $\{1\}$. Hence in round 4 at this layer $l = 3$, candidate coalitions

are $\{1\} and \{1,3\}$. The best coalition is $\{1,3\}$ with $\overline{a} = 4$. It reaches the end of the loop and goes to the next round. In round 5), this new coalition is placed in layer 3 of $\mathcal{CS}$ and its members are removed from $\mathcal{CS}$. There is no remaining agents. Hence, the new $CS$ is output. During round 6) to 8), the algorithm alters its last coalition, shrinks and extend $CS$ where the new $CS$ is generated.

In round 15), $CS$ is shrunk by 1 layer 1. Here, the algorithm reaches to top layer $l$=1 of $CS$. It changes the candidate coalition of this cardinality from $\mathcal{C}[1][1] = \{4\}$ to $\mathcal{C}[1][2] = \{2\}$. Among all candidates in this layer, which are shown in round 16), the best coalitions is $\mathcal{C}[1][\mathcal{B}[1][2]] = \{2\}$. It reaches the end of the loop and goes to the next round. In round 16), the head of $CS$ is now changed to $\{2\}$. The algorithm keeps extending $CS$ until it can generate the new $CS$ in round 18). The change of the head of $CS$ to a lower order coalition, similar to what happens in round 16), continues as the as the execution proceeds. This will lower the upper bound to optimality across the whole search space. The execution continues until it reaches round 51) where the layer of $CS$ is 1 and the best coalition chosen in previous round from the only candidate is the grand coalition. After outputting the new $CS$, the algorithm cannot extend nor alter. It tries the last option, shrink. This leaves no more candidate coalitions and leads the execution to the end. Note that the example discussed here is to show that the algorithm generates all $CS$s. The $CS^*$ is the second one.

### 3.2.5 Applying Branch and Bound Method

Branch and bound is a well known technique in computer science to reduce execution time by ignoring some search space which is useless, i.e. we will never find a better solution in that portion of the whole search space. Here, we use a branch and bound mechanism in order to increase the performance of the algorithm, i.e. the algorithm can converge to the optimality quickly.

Let $CS_B$, i.e. $\mathcal{CS}[l]$, is a coalition being constructed at a point in time. Its value, $V(CS_B)$, so far is the sum of the values of candidate coalitions chosen up to the point. Let $S^*$ be the best candidate whose average agent contribution is $\overline{a}^*$. The highest possible value the optimal value can be is simply

$$V(CS_B) + (n - |CS_B)|) \times \overline{a}^*.$$

Let $V(CS^*)$ be the present solution. The algorithm needs to search further from its current $CS_B$ if and only if

$$V(CS_B) + (n - |CS_B)|) \times \overline{a}^* > V(CS^*)$$

holds. We can just apply this condition to the $ChooseNextS$ function when choosing the best candidate. If the above condition is no longer satisfied, there is no need to search any further from the current $CS_B$. $ChooseNextS$ simply returns null. Hence the algorithm backtracks one step and proceeds with the available candidates.

### 3.2.6 Example of Applying Branch and Bound

There are two modifications need to make in the algorithm. First, every $CS$ that has been generated must be compared to the $CS^*$. If $V(CS) > V(CS^*)$ holds, then $V(CS^*)$ is set to $V(CS)$ as the new solution. Second, the best coalition has to be determined if there is a chance it would raise the value of $V(CS^*)$ before the function $ChooseNextS$ returns the best coalition. This is to determine if $V(CS_B) + (n - |CS_B|) \times \overline{a}^* > V(CS^*)$ holds.

With respect to the real execution in the same example above, the first $CS$ generated, $\{\{4\}, \{2\}, \{3\}, \{1\}\}$ is set as $CS^*$ with $V(CS^*) = 21$. After shrinking $CS$ up to layer 3, the new $CS^* = \{\{4\}, \{2\}, \{1, 3\}\}$ is found with the new value 23. After that the algorithm will alter at $\mathcal{CS}[3]$ where it learns that the next best coalition $\{1\}$ may raise $V(CS^*)$ (up to 25). However, a new $CS$ can never be generated because there is no more coalition left. The algorithm shrinks to layer 2 and alter $\mathcal{CS}[2] = \{2\}$ with $\{3\}$. However, it will not extend $\mathcal{CS}$ to layer 3 because $\{1, 2\}$ will never raise $V(CS^*)$ (the highest possible value is 9+5+6=20.) The similar situation will also happen at level 2 where the algorithm shrinks to level 1 and alter $\{4\}$ with $\{2\}$. However, it will never extend $\mathcal{CS}$ any further because there is no chance $\{3\}$ will raise $V(CS^*)$. These things will be repeated until there is no candidate coalition at layer 1.

## 3.3 Experimental Results

**Settings:** Since we have already discussed that the distribution of coalition values is relevant to the optimal coalition structures because it suggests how their patterns might be [70, 69, 71], this work categorizes coalition value distribution differently from previous work [51], where the coalition value distribution is based on coalition value alone regardless of coalition cardinalities. The categories of the distribution are superadditive, subadditive, normal and uniform distribution. Here, we consider the distribution in two dimensions, by taking into account coalition values and their cardinalities. In the first dimension, we consider the ranges of coalition values in all cardinalities. The low ends of the ranges are 0 while the high ends are categorized into 8 cases. $i$) STD: The maximal coalition values in all cardinalities are roughly stable. They may fluctuate slightly. $ii$)IND: The maximal coalition values increase

by the cardinalities. (Note that although this is similar to superadditive but it is not quite the same.) $iii$) DCD: The maximal coalition values decrease by the cardinalities. Again, it is similar but is not subadditive. $iv$) CCD: The maximal coalition values on cardinalities 1 and $n$ are high and decrease towards the medium cardinalities. This distribution is concave. $v$) CVD: The maximal coalition values on cardinalities 1 and $n$ are low and increase towards the medium cardinalities. This distribution is convex and is more common in real world settings. It reflects environments in which cooperation helps increase revenue and profit until cooperation costs become too costly when coalitions get too large [101, 100]., $vi$) RDD: coalition values are random. To generate a coalition value, we randomly choose cardinality, randomly choose a coalition whose value is yet to be assigned, and randomly choose a value. $vii$) NMD: This is normal distribution as in previous work [51]. $viii$) UNI: This is uniform distribution in previous work [51]. In the second dimension, we consider the distribution of coalition values in each cardinality of the STD, IND, DCD, CCD and CVD distribution. We have three varieties in each of them, i.e., normal distribution in each cardinality with the mean leaning towards the highest(F10), the middle(F5) and the lowest(F1) coalition values. We do this in order to observe the effect of distribution of coalition values within cardinalities.

Since Rahwan et.al.(RN) is apparently the state of the art, we benchmark the performance of our algorithm (CH) against it.[3] Similarly to RN, we are computationally bounded with just 2GB of RAM usable with Windows operating system on each of our computers. The available memory on each computer allows space enough for only experimenting our centralized algorithm for 26 agents. (We need $2^n \times 8$ bytes of memory to store all coalition values.) Note that the main purpose of this algorithm is to demonstrate the performance of the algorithm for a reasonable number of agents. We will demonstrate how to cope with the problem with larger number of agents in later chapters. For each of $20 \leq n \leq 26$ agents, we generate 100 samples for each of the distribution mentioned above. We allow 15, 30, 45, 60, 75, 90 and 105 minutes of execution time for $n = 20, n = 21, n = 22, n = 23, n = 24, n = 25$ and $n = 26$ respectively. These duration times are estimated values and are slightly higher than the termination times of RN's worst case (normal distribution) in order to ensure the the allowed time is enough for RN to terminate.

[3]Both algorithms are implemented in Java 1.5. Note that we were not given RN implementation, we try our best on several ways and find that using arrays allows for both algorithms to run at the fastest speed possible. The representation of the coalitions and their values are the same as in our implementation. The executions are done on 120 Pentium 4 2GHz with 2GB of ram machines running Windows XP. These 120 machines are distributed across 4 laboratories.

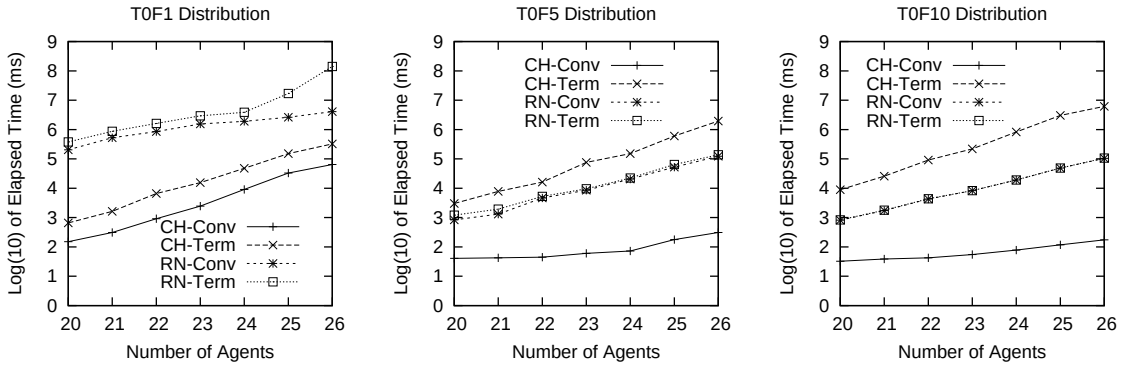


Figure 3.3: **Empirical Results on STD Distribution** The graphs show convergence and termination times of Algorithm CH against that of algorithm RN on STDF1, STDF5 and STDF10 distributions.

### 3.3.1 Empirical Results

For each sample data, we observe the elapsed time for i) convergence (the solution reaches the highest value but yet to terminate) and ii) termination (the algorithm terminates because either there is no way to improve the solution or it is timeout). For each of these elapsed time, we find the average, highest and lowest elapsed times for $20 \geq n \leq 26$ agents for both algorithms.

In STD, as shown in 3.3, our algorithm converge earlier in all variations. We trace every run of both algorithm and find that our algorithm always generates higher V(CS) at any point in time until it reaches the highest value at which RN generates at later stage. In terms of terminations, it varies. In variation F5 and F10, RN offers almost the same time the for convergence and termination. In F1, the convergence and termination of RN diverge slightly and are higher than that of our algorithm and its own performance in F5 and F10. This can be easily interpreted as RN might misled by the upper bound of configurations. In F5 and F10, our algorithm converge earlier but terminates after RN. However the termination times in both cases are still lower than the worst case of RN in F10.

In IND, as shown in 3.4, our CH algorithm converges and terminates earlier than RN in variation F1 and F5. However, our algorithm converges earlier but terminates later than RN in variation F10. The convergence and termination of RN in all variations are quite different.

In DCD, as shown in 3.5, our algorithm converges and terminates earlier than RN in variation F1. However, it terminates later than RN in variation F5 and F10. Note that IND and DCD are not really superadditive and subadditive environments in which our algorithm would perform well as discussed above.

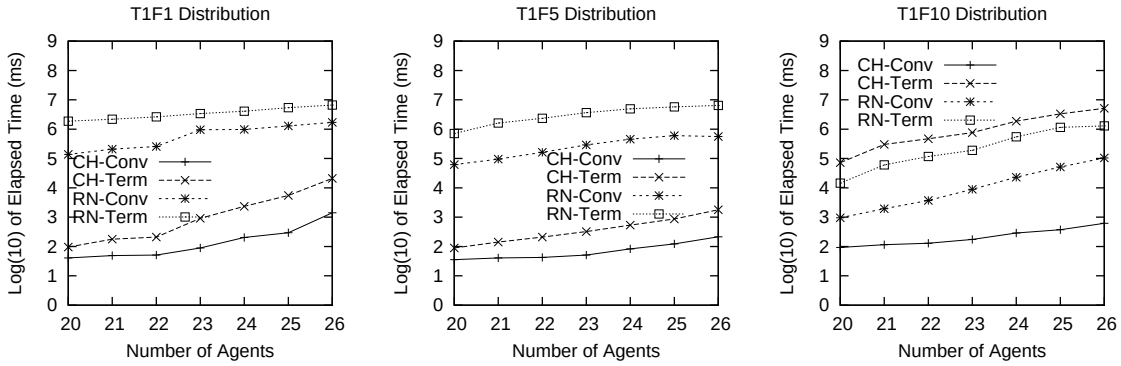


Figure 3.4: **Empirical Results on IND Distribution** The graphs show convergence and termination times of Algorithm CH against that of algorithm RN on INDF1, INDF5 and INDF10 distributions.

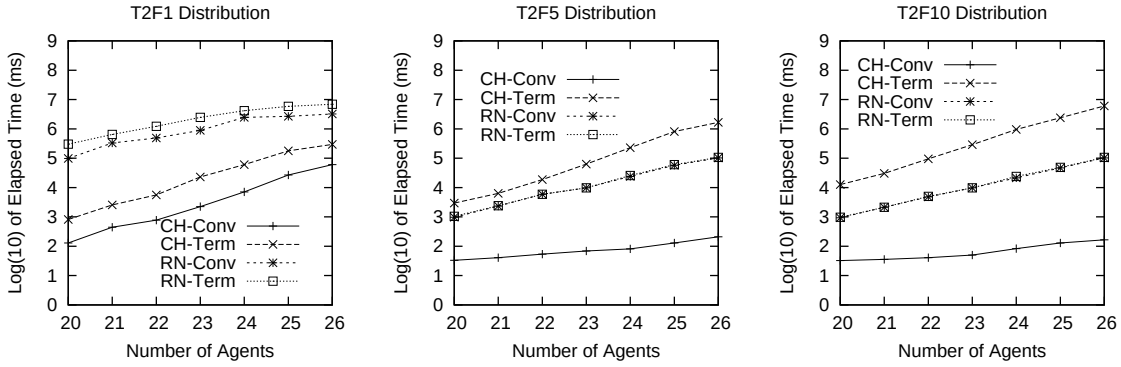


Figure 3.5: **Empirical Results on DCD Distribution** The graphs show convergence and termination times of Algorithm CH against that of algorithm RN on DCDF1, DCDF5 and DCDF10 distributions.

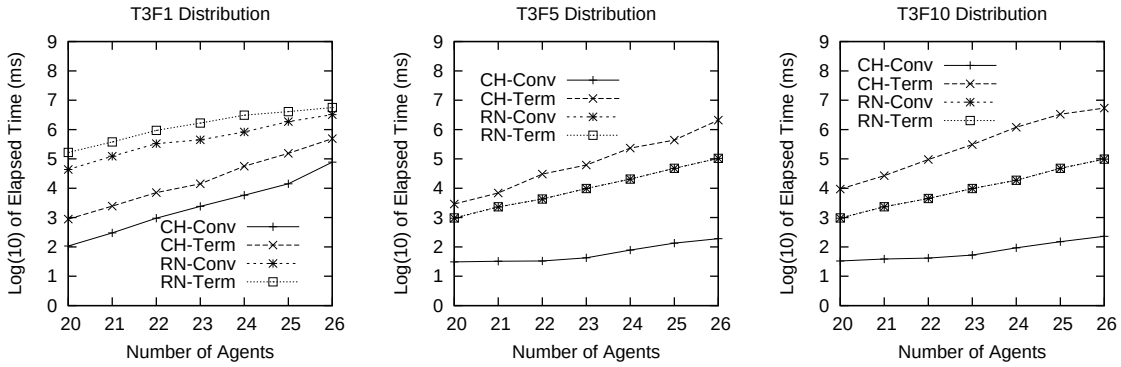


Figure 3.6: **Empirical Results on CCD Distribution** The graphs show convergence and termination times of Algorithm CH against that of algorithm RN on CCDF1, CCDF5 and CCDF10 distributions.

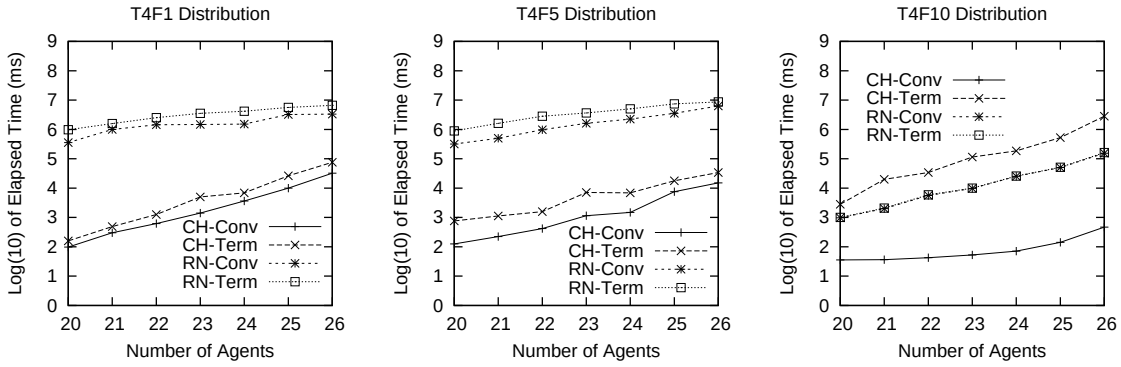


Figure 3.7: **Empirical Results on CVD Distribution** The graphs show convergence and termination times of Algorithm CH against that of algorithm RN on CVDF1, CVDF5 and CVDF10 distributions.

In CCD, as shown in 3.6, CH converges and terminates earlier than RN in variation F1 but it terminates later in variation F5 and F10. Note that the convergence of our algorithm in F1, when it terminates earliest, is higher than that in F5 and F10.

In CVD, as shown in 3.7, CH converges and terminates earlier than RN in F1 and F5. It terminates later than RN in F10. Across all variations, the worst case of average of CH is still better than that of RN.

In RDD, as shown in 3.8, CH converges and terminates earlier than RN. In NMD, in which RN performs worst across the four distributions, CH converges and terminates earlier in all variations. Both algorithms terminate shortly after convergence. In UFD, CH converges earlier RN but terminates about the same time as RN converges and terminates in all

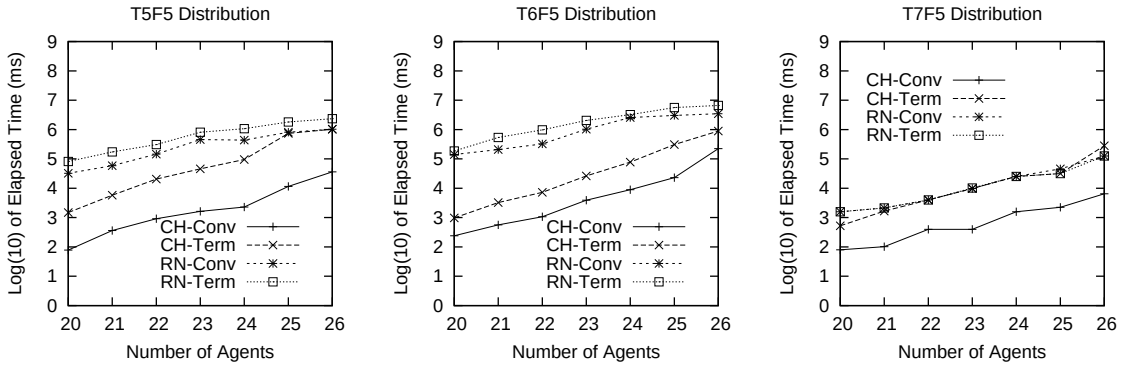


Figure 3.8: **Empirical Results on RDD, NMD and UNI Distribution** The graphs show convergence and termination times of Algorithm CH against that of algorithm RN.

variations.

As shown in the graphs above, our algorithm always converges earlier than RN. This implies that our algorithm guarantees better or, at least, as good as RN's result at anytime (although our algorithm takes longer to terminate in some cases.) The further implication of this is that generating coalition structures from best coalitions can help reach optimal coalition faster. This complies with Aumann's alpha core which states that optimal coalition structures involve a small set of coalitions. This is based on the simple fact that merely generating coalitions alone can be intractable for a centralized system because all the existing algorithms (including ours) requires that all coalitions and their values must be observed before the actual generation–let alone the coalition structure generation. For example, it is impossible to execute any of the existing centralized algorithms for 60 agents because none of the existing single computer systems can offer enough memory. In terms of anytime algorithms which are more appropriate for multi-agent systems, our algorithm empirically shows that it always generates better or, at least, as a good solution as RN. Note that the coalition value distribution within cardinalities affects the performance of both algorithms differently.

Across all 18 distribution variations, there are 8 cases where our CH algorithm terminates later than RN. These cases are STDF5, STDF10, INDF10, DCDF5, DCDF10, CCDF5, CCDF10 and CVDF10. With STDF10, INDF10, DCDF10, CCDF10 and CVDF10, the average of coalition values in each cardinality is very close to the upper bound of the coalition values in each cardinality. This obviously results in that the upper bound of each configuration is very close to the exact optimal coalition structure value of that configuration. Therefore, RN can find the optimal coalition structure value of the configuration quickly and, as

a consequence, it can also determine if it needs to search in anymore configuration quickly. With STDF5, DCDF5 and CCDF5, the average of the coalition values of small coalitions are larger than that of the large ones (this is common for all STDs, DCDs and CCDs). This results in that configurations containing small coalitions tend to have both higher upper bounds and exact optimal coalition structure values. (Example of these configurations for 4 agents are 1+1+1+1, 1+1+2, etc.) Since, there are not many of them and the size (number of coalition structures) of each of them is relatively small, the search for real optimal coalition structure value in each configuration can be done quickly. As a consequence of this, RN can prune and terminate quickly. Note that these distributions are invented by us to show if an algorithm's performance is robust enough, i.e. reaching optimality quickly regardless of the distribution of coalition values.

## 3.4 Conclusion

In this chapter, we have sought to develop a best-first search anytime algorithm that provides a guarantee of generating the optimal solution, given it has enough time. The algorithm advances the state of the art, in terms of anytime algorithms, by always generating a solution better or, at least, as good as RN at any point in time. We present empirical results that support our claims.

Thus, the work presented in this chapter has achieved the first objective of this research, namely:

1. to develop a best-first, anytime algorithm that is an efficient solution for OCS problems in environments where coalition values are known a priori

Having successfully met this objective, our goal is now to deploy the best-first anytime algorithm to a number of more complex OCS domains, particularly those in which the coalition values and coalition structure values are not known a priori. In the next chapter we will adapt this algorithm to a linear production domain.

# Chapter 4

# Computing OCS in Linear Production Domain

## 4.1 Introduction

The previous chapter has achieved the first objective of this research, namely:

1. to develop a best-first, anytime algorithm that is an efficient solution for OCS problems in environments where coalition values are known a priori

The next four chapters will adapt the algorithm presented in Chapter 3 to solve other more difficult or more interesting problems in a variety of domains. This chapter will attempt to solve OCS problems in a linear environment where coalition values and coalition structure values are not known a priori but must be calculated thereby achieving the second objective of this research.

Although the best-first search anytime algorithm presented in the previous chapter shows that generating $CS$ from best coalitions can empirically reach optimality relatively quickly, it is not pragmatic in real world setting for several reasons. Firstly, it is a centralized approach and is computationally bounded. Since coalition values are known a priori, all the values must be kept in memory in practice for prompt access. The most efficient algorithm can work for only 27 agents on typical computers which have 4GBs of RAM. To handle 50 agents, it requires at least one Peta bytes ($10^{15}$) of RAM, which no single computer in the world can offer. Secondly, given the huge number of coalitions, computing all coalition values alone can take too much time because computing a coalition value can be a complex optimization problem. Hence, it may not be acceptable in practice.

On the other hand, this chapter approaches coalition formation in more realistic settings. This chapter considers coalition formation where coalition values are not known a priori, which is common in real world environments as we have already discussed in Chapter 2. These real world scenarios make coalition formation highly complex because agents have to $i$) compute coalition values, and $ii$) compute the optimal coalition structures. Given $n$ agents in a coalition formation process, the number of possible coalitions is $2^n$, which is

also the number of coalition values to be computed. The process of computing coalition values is complex, as is the process of deliberation. The problem becomes intractable even for relatively small values of $n$.

Our goal in this research is to deal with this complexity. We modify Owen's linear production game where agents have to agree to pool their resources together in order to produce goods. The original work assumes a superadditive environment, where agents can simply form the grand coalition. Such an assumption is impractical in the real world since the cost of cooperation has to be taken into account while negotiating to form coalitions.

What we have learned is that the best-first anytime search algorithm presented in the previous chapter shows that generating $CS$ from best coalitions can empirically reach optimality relatively quickly. However, the limitation of the algorithm is that it requires sorted coalitions (which can be done in polynomial time) and requires dedicated memory allocation during the execution. This prevents the algorithm from being practical. However, we can use this best-first approach to develop an algorithm that can generate $CS$s quickly without the need to scan all of the coalitions. Furthermore, the $CS$s generated must be very close to optimality.

## 4.2 Coalition in a Linear Production Domain

Linear production games [63] are those in which agents are given resources and try to pool resources to produce goods in order to maximize the system's profit. Owen [63] studied linear production games in superadditive environment. Here, we consider linear production games in non-superadditive environments. We are given a set of agents, $A = \{a_1, a_2, \ldots, a_m\}$, whose goals are to maximize the system's profit. We are also given a set of resources $R = \{r_1, r_2 \ldots, r_n\}$ and a set of goods $G = \{g_1, g_2, \ldots, g_o\}$. Resources themselves are not valuable but they can be used to produce goods, which are valuable to agents. Let $L = [\alpha_{ij}]_{n \times o}$, where $\alpha_{ij} \in \mathbb{Z}^+$, be the matrix that specifies the units of each resource $r_i \in R$ required to produce a unit of the good $g_j \in G$. Such a matrix is called a *linear technology matrix* [63]. The price of each unit of goods produced is specified by the vector $P = [p_j]_{1 \times o}$. Each agent $a_k \in A$ is given a resource bundle $b^k = [b_i^k]_{n \times 1}$. In this setting, some agents would have the incentive to cooperate, e.g., if they cannot produce a certain good using only the resources at their disposal. Hence agents have to cooperate, i.e. form coalitions, in order to create value from their resources. Let $S \subseteq A$ be a coalition. It will have a total of

$$b_i^S = \sum_{k \in S} b_i^k$$

of the $i^{th}$ resource. The members of coalition $S$ can use all these resources to produce any vector $x = \langle x_1, x_2, \ldots, x_o \rangle$ of goods that satisfies the following constraints:

$$\begin{array}{rcl} \alpha_{11}x_1 + \alpha_{12}x_2 + \ldots + \alpha_{1o}x_o & \leq & b_1^S, \\ \alpha_{21}x_1 + \alpha_{22}x_2 + \ldots + \alpha_{2o}x_o & \leq & b_2^S, \\ \vdots & \vdots & \vdots, \\ \alpha_{n1}x_1 + \alpha_{n2}x_2 + \ldots + \alpha_{no}x_o & \leq & b_n^S \end{array}$$

and

$$x_1, x_2, \ldots, x_o \geq 0.$$

We assume that agents have to pool their resources together at a coalition member's location to produce these goods. Thus agents' cooperation incurs some costs, e.g., transportation cost, etc. The cooperation cost among agents is specified by the matrix $C = [c_{kl}]_{m \times m}$, which assigns a cooperation cost between each pair $(a_k, a_l)$ of agents such that

$$c_{kl} \in \begin{cases} \mathbb{Z}^+ & \text{if } k \neq l \\ \{0\} & \text{if } k = l \end{cases}$$

We assume that all of the resources of agents are pooled at one location, which can be the location of any agent in the coalition. A singleton coalition yields cooperation cost of 0. For a coalition of size two, $S = \{a_1, a_2\}$, pooling coalition resources at any of the two sites yield the same cost for the coalition (i.e. the cooperation cost matrix is symmetric). The total cost for cooperation incurred by a coalition will be taken to be the sum of the pairwise cooperation costs between the agent at whose location coalition resources are pooled, and the other members of coalition. For a coalition of size three or larger, there is at least one agent, $a_k$, such that

$$\sum_{k'=1}^{m} c_{kk'} \leq \sum_{l'=1}^{m} c_{ll'}$$

for all $a_l \in S$. We shall call a coalition member $a_k$ who yields the minimal cooperation cost for the coalition a *coalition center*.

Agents in the coalition $S$ have to find a vector $x$ to maximize the revenue accruing to a coalition. Let

$$P_S = \sum_{l=1}^{o} p_l x_l.$$

be the maximal revenue the coalition can generate. Let

$$C_S = \sum_{l \in S} c_{kl}.$$

be the minimal cooperation cost for the coalition (obtained by selecting the optimal coalition center). Obviously, the ultimate objective of agents in the coalition is to maximize profit, i.e., the coalition value $v_S$, where

$$v_S = P_S - C_S.$$

The linear inequalities referred to above, together with this objective function constitutes a linear programming problem. We shall call the solution, the vector $\langle x_1, x_2, \ldots, x_o \rangle$ that represents the optimal quantities of goods $g_1, g_2, \ldots, g_o$ *optimal product mix*.

## 4.3 Distributed Algorithm for Coalition Formation

Here, we consider a distributed algorithm that allows agents to compute coalition values and approach the optimal coalition structures as they proceed. Each agent has to do to two tasks: $i$) Delibarating: deliberate over what coalitions it might form by incrementally improving the initial set of coalitions, and $ii$) Forming coalitions: exchange information to form coalitions such that those coalitions yield maximal profit to the system. The sets of such coalitions are the optimal coalition structures. The main goal of the algorithm is to reduce the search space for finding the optimal coalition structures. This can be achieved by reducing the number of coalitions to be considered. In our setting, the optimal coalition structures must yield a profit, a non-negative utility, to the system. In the worst case, the system's profit is 0—each agent is a singleton coalition and cannot produce anything at all.

### 4.3.1 Deliberating Process

In the following, we will identify a coalition by the identifier of its coalition center agent. Thus the coalition $S^k$ will have agent $a_k$ as its center. Hence $b^S$ represents the resource vector of $S^k$. The reasoning described below is conducted by the coalition centre agent for each coalition. Given a coalition $S^k$, let $G^k$ refer to the set of goods whose resource requirements are fully or partially satisfied by $b^S$, the resources available in $S^k$ (excluding goods whose resource requirement might be trivially satisfied because these are $0$). For each good $g_j \in G^k$, the coalition centre agent $a_k$ ranks agents not currently in its coalition on a per good basis. For each resource $r_i$ of good $g_j$, agent $a_k$ ranks non-member agents by computing for each $a_l \notin S^k$, whose $b_i^l > 0$, the value $\pi_i^j$—its proportional contribution to the profit of the good (using its fraction of the resource requirements for that good provided by

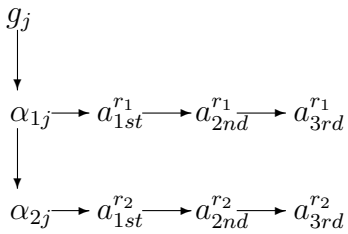


Figure 4.1: **Ranking Agents**Agents are ranked by their potential profit per each resource of a good.

the $a_l$) minus the (pair-wise) collaboration cost between $a_l$ and $a_k$, i.e.,

$$\pi_i^j = \frac{b_i^l}{\alpha_{ij}} p_j - c_{kl}.$$

The agent $a_k$ uses this proportional contribution $\pi_i^j$ to construct a binary tree for each $g_j$. The only child of the root $g_j$ is the first resource $\alpha_{1j}$, whose left child is the second resource $\alpha_{2j}$, and so on. For each $\alpha_{ij}$, its right child is either $i$) null if $\alpha_j^i = 0$, or $ii$) the agent $a_{1st}^{r_i}$, whose $pi_i^j$ value is the greatest. The right child of $a_{1st}^{r_i}$ is the agent $a_{2nd}^{r_i}$, whose $\pi_i^j$ value is the second greatest, and so on. Every time $a_k$ wants to produce additional units of $g_j$, it traverses the tree down to the appropriate resource $r_i$ and add more agents into its coalition based on $b^S$. Figure 4.1 depicts this structure.

The agent $a_k$ uses $b^S$ to determine additional resources needed to produce additional units of a good $g_j$. For each $g_j \in G^k$ and resource $r_i$,

$$\beta_i^j = I(\alpha_{ij}) - b_i^S,$$

where $I \in \mathbb{Z}^+$ is the smallest integer such that $\beta_i^j > 0$, represents the amount of $r_i$ that coalition $S^k$ lacks to produce good $g_j$, provided the amount is non-negative ($\beta = 0$ otherwise). The *indicative vector*, $\beta^j = [\beta_i^j]_{1\times n}$, represents un-met requirements for each resource $r_i$ of good $g_j$.

In this process, an agent may choose to do one of the following in order to generate coalitions: $i$) *growing coalitions*: adding profitable agents to existing coalitions, or $ii$) *shrinking coalitions*: removing costly agents from existing coalitions. The following subsection describes both processes in details.

### Growing Coalitions

The agent $a_k$ uses the indicative vector $\beta^j$ to help collect additional coalition members into its coalition. If the agent $a_k$ wants to produce an additional unit of $g_j$, it identifies the resource

that is needed the most, $\beta^j_{i^*} = max^n_{i=1}(b^j_i)$, from the indicative vector. It locates the node $\beta^j_{i^*}$ in $T^{g_j}$ and collects the next available agent $a^{i^*}_l$ into the coalition. The total resources of the coalition $b^S$ is updated. Each $\beta^j_i$ of indicative vector will be subtracted by it corresponding $b^l_i$. The agent $a_k$ keeps adding more agents into its coalition until there are enough resources to produce an additional unit of $g_j$, i.e., $\beta^j_i > 0 \; \forall i$. The algorithm to collect additional agents into the coalition is shown in algorithm 7.

**Algorithm 7** Select additional agents

**Require:** the present coalition $S$
**Require:** the focused good $g$
  initialize additional agents $S' = \emptyset$
  get the coalition's resource $b^S$
  get the indicative vector $\beta^{g'}$
  identify the most needed resource $r_{i^*}$
  **while** $r_{i^*} > 0$ **do**
    locate next available agent $a_l^{r_j}$
    **if** $a_l^{r_j}$ ¡0 **then**
      break
    **end if**
    set $S' = S' \cup a_l^{r_j}$
    **for all** $\beta^j_i$ **do**
      set $\beta^j_i = \beta^j_i - b^l_i$
    **end for**
    identify the most needed resource $r_{i^*}$
  **end while**
  return $S'$

In the extended part, each agent ranks profitable coalitions in its ranking tree. The root of the tree is the singleton coalition of the agent, $S^k$. So far, the agent $a_k$ knows that if it wants to produce at least an additional unit of $g_j$, it needs to acquire additional agents, $S'$, into its $S^k$. The agent $a_k$ create a trial coalition by merging $S'$ into $S$. Since each new agent may posses other resources not required for producing $g_j$, the trial coalitions may find a better solution for producing goods. Hence the profits $v$ of trial coalitions vary. Each $S'$ will be added to the tree as the children of $S$. The sub algorithm for selecting profitable members is shown in algorithm 8.

In the main algorithm, the agent $a_k$ considers itself a singleton coalition at the beginning of deliberating. It create the ranking tree $T^G$ of all agent for each good. At this point it is only the root of the profitable-coalition tree, $L^+$, and is the base of the growing coalition. It acquires the additional agents $S^+$ into the coalition. Each $S'_j \in S^+$ will be added as the children of the base coalition. Among all $S'_j$s, the most profitable agents $S^*$ are those

**Algorithm 8** Select the most profitable members

**Require:** A coalition $S$
**Require:** ranking trees $T^G$
  set highest profit $\upsilon^* = 0$
  set profitable members $S^+ = null$
  **for** j:=1 **do**—S—
    **if** $S$ is not capable of producing $g_j$ **then**
      continue
    **end if**
    get additional agents $S'$
    set trial coalition $S'_j = S \cup S'_j$
    compute trial coalition's profit $\upsilon_{S'_j}$
    set $S^+ = S^+ \cup S'_j$
  **end for**
  return $S^+$

that provide the highest additional profit $\upsilon^*$ and are kept as the base for the further growing coalition. The coalition keeps growing in this fashion until there are no profitable members left in $T^G$. Then the next most profitable sibling of the base $S'_j$ will be the new base. This repetition goes on until it cannot find a new base. This will keep the coalition's marginal profit growing while the size of the coalition is growing. The number of coalitions each agent $a_k$ has to maintain is also much smaller compared to that of an exhaustive search. The main algorithm is shown in algorithm 9.

**Algorithm 9** Main Grow

  set $L^+ = \emptyset$
  create a singleton coalition $S = \{a_k\}$
  set $A'$ = $A - \{a_k\}$
  create ranking trees $T^G$ for all goods
  collect profitable members $S^+$
  **while** $S^+ \neq \emptyset$ **do**
    locate $S^* \in S^+$
    set $A' = A' - S^*$
    set $S = S \cup S^*$
    set $L^+ = L^+ \cup S$
    collect profitable members $S^+$
    **if** $S^+ = null$ **then**
      set $S^*$ = the next profitable sibling of$S^*$
    **end if**
  **end while**

### Shrinking Coalitions

Alternatively, coalitions can be generated by shrinking the grand coalition. The agent $a_k$ creates the grand coalition and tries to shrink it by pruning the least profitable members. The agent utilizes indicative vectors $\beta^j$s and the the tree $T^j$ in order to locate the agent who is the least useful to its present coalition. For each good, the positive value of $\beta_i^j$ in the indicative vector indicates surplus resource that the agent who possesses the equivalent resource should be eliminated from the present coalition. The agent $a_k$ create a trial coalition $S'$ for each good. The surplus agents will be eliminated from $S$ for the next smaller quantity of the good possible. Each trial coalition will be inserted into the pruning members $S^-$. The sub-algorithm for selecting profitable members is shown in algorithm 10.

**Algorithm 10** Select the least profitable members

**Require:** A coalition $S$
**Require:** ranking trees $T^G$
  set highest profit $\upsilon^* = 0$
  set pruning members $S^- = S$
  **for all** $g_j \in G$ **do**
    **if** $S$ is not capable of producing $g_j$ **then**
      continue
    **end if**
    get surplus agents $S'$
    set trial coalition $S'_j = S \cup S'_j$
    compute trial coalition's profit $\upsilon_{S'_j}$
    set $S^- = S^- \leftarrow S'_j$
  **end for**
  return $S^-$

In the main algorithm, the agent $a_k$ considers itself a virtual coalition center of the grand coalition. at the beginning of deliberation. It create the ranking tree $T^G$ of all agent for each good. At this point, it is root and the only member of the profitable-coalition tree,$L^-$. It prunes the pruning agents $S^-$ from the coalition. Each $S'_j \in S^-$ will be added as the children of the base coalition. Among all $S'_j$s, the most profitable agents $S^*$ are those that provide the highest additional profit $\upsilon^*$ and are kept as the base for the further shrinking coalitions. The coalition keeps shrinking in this fashion until there are no prunable members left in $T^G$. Then the next most profitable sibling of the base $S'_j$ will be the new base. This repetition goes on until it cannot find the new base. The number of coalitions each agent $a_k$ has to maintain is also much smaller compared to that of an exhaustive search. The main algorithm is shown in algorithm 11.

**Algorithm 11** Main Shrink

set $L^- = N$
create ranking trees $T^G$ for all goods
collect pruning members $S^-$
**while** $S^- \neq \emptyset$ **do**
  locate $S^* \in S^-$
  set $A' = A' - S^*$
  set $S = S \cup S^*$
  set $L^- = L^- \cup S$
  collect pruning members $S^+$
  **if** $S^+ = null$ **then**
    set $S^* =$ the next profitable sibling of $S^*$
  **end if**
**end while**

### 4.3.2 Coalition Formation Algorithm

At this point, each agent has a number of coalitions it has generated. It also needs to know other coalitions generated by other agents before generating coalition structures. It is also important that the number of coalitions to be input into the process of generating coalition structures must be relatively small in order to avoid intractability.

The first thing is to prune the unuseful coalitions–those that are non-profitable. Once each agent finishes its deliberation, it ranks all of its coalitions by profit. Let $S^-$ be a non-profitable coalition, whose value $v_{S^-} \leq 0$, and $S^+$ be a profitable coalition, whose value $v_{S^+} > 0$.

**Lemma 1** *Any $S^-$ coalition can be replaced by a set of its members' singleton coalitions, whose $v_{a_{k \in S}} \geq 0$, such that the coalition structure's value will not be decreased.*

Therefore, all non-profitable coalitions can be ignored. Each agent will remove all of the non-profitable coalitions, if there are any, one by one. The remaining coalitions are profitable. In fact, our algorithm in the deliberation process can simply prevent this happening using its tree $T^G$. It always generates profitable coalitions. Obviously, each singleton coalition is non-negative. Hence, non-profitable coalitions must not exist in the coalition structures.

Given that the deliberation algorithm generates all profitable coalitions among agents inclusively, agents can $i$) exchange information about coalitions generated and their singleton coalitions, and $ii$) decide to form coalitions that yield the optimal coalition structure value.

**Proposition 1** *The optimal coalition structure can be constructed by profitable coalitions generated by agents and their singleton coalitions.*

As for exchanging information about the profitable coalitions among agents, we use a high-level algorithm since it is a common practice in the literature for broadcasting information among agents [77, 48, 32, 87, 47, 84, 46] during coalition formation. Once the agent is ready, i.e. it has finished the pruning, it broadcasts the list of profitable coalitions to other agents one by one. After that, the agent just collects each of the lists sent over from all other agents.

Having received all the coalitions, further improvement can still be achieved–keeping only optimal coalitions. Note that at this point there will be $|S|$ variations of $S$, each of which is $S$ with different center (which generates $S$) and value. Given a coalition of agents, $S = \{a_k, a_l\}$, for example, there is one variant that was generated by $a_k$, as the center, which we denote by $S_k$, and the other variant that was generated by $a_l$, as the center, which we denote by $S_l$. Both $a_k$ and $a_l$ have these coalitions in their list after exchanging. However, among these variant coalitions, there will be just one optimal coalition,

$$S^* = argmax_{S_k \in S} V_{S_k},$$

where $S_k$ is a variant of $S$.

In the process of finding $S^*$, agent $a_k$ can simply set $S_k$ as the $S^*$ and compares it against other variants. If any of those variants has higher coalition value, it will be set as the new $S^*$. Each agent will keep only $S^*$, i.e. other $S_* \neq S^*$ will be deleted from the list. After this, each agent will have just one coalition with optimal value, i.e. ($S^*$) [42]. Actually, this step reduces a large number of possible redundancies during the calculation for the optimal value of a coalition. Otherwise, for each coalition, each member of the coalition has to compute for the optimal coalition value by choosing the all possible centers, which is inefficient. The main distributed algorithm for generating profitable coalitions among agents is shown in algorithm 12. For the sake of illustration, we assume this algorithm be run at agent $a_k$.

One may raise a question regarding the communication cost among agents as it seems agents have to communicate extensively. Actually, the agents merely send messages to each other just once, i.e. after the deliberation. The number of messages sent across communication network is relatively small, i.e. $n^2$. The content of each message (containing coalitions and their values) can be represented in plain text and can be zipped into a small piece of data. Given the present communication infrastructure, where multi-million messages are being sent across the communication network and the cost is well covered, it is unlikely that the algorithm would incur any significant burden, in terms of communication cost and performance, to the system.

The other concern can be the synchronization during the exchange of messages. While

**Algorithm 12** Main Distributed Algorithm

**Require:** $L_k$ set of self-generated coalitions.
  set $L^* = L_k$
  **for** each $S_k$ in $L_k$ **do** ▷ prune non-profit coalitions
    **if** $\upsilon_{S_k} \leq 0$ **then**
      set $S_k = \emptyset$
    **end if**
  **end for**
  **for** each $a_l \in N$ and $a_l \neq a_k$ **do** ▷ broadcast generated coalitions
    $a_k$ send $L_k$ to $a$
  **end for**
  **for** each $a_l \in N$ and $a_l \neq a_k$ **do** ▷ collect generated coalitions
    agent $a_k$ collects all coalitions, $L^* \cup L_l$
  **end for**
  **for** each $S \in L^*$ **do** ▷ find optimal coalition
    set $S^* \leftarrow S_k$
    **for** each $S_l$ of $S$ and $S_l neq S_k$ **do**
      **if** $\upsilon^*_S < \upsilon_{S_l}$ **then**
        set $S^* \leftarrow S_l$
      **else**
        $S_l \leftarrow \emptyset$
      **end if**
    **end for**
  **end for**

our high-level algorithm sends and collect lists of coalitions consequently, one can deploy any low-level technique, such as having a daemon program for taking care of sending and receiving messages, in real implementation. This may also allow the sending and receiving on each agent be done independently. However, we leave this to the further development as it is beyond the focus of this work.

At this point, these remaining coalitions kept at each agent will be profitable and optimal. Furthermore, their number is relatively small. Each agent can use these highly-valued coalitions to generate coalition structures. The detailed algorithm will be discussed in the next section.

### 4.3.3 Best Coalition and Coalition Structure Pattern

In previous studies [77, 22], coalition structures are generated based on the size of coalition structures and the cardinality of the coalitions. It appears that the search space is very large. Here, we try to reduce the search space. For each cardinality, each agent tries to do local search for a small number of coalitions. Firstly, we define the agent $a_k$'s *best coalition* for the cardinality $\kappa$ the coalition $S_k^\kappa$, whose members include $a_k$, that is found from a search within a given time and yields the maximal $\upsilon_S$. Within the same cardinality, the next coalition that yields the second highest coalition value is *second* best coalition, and so on.

We introduce the *pattern* of generating coalition structures. A pattern of a coalition structure describes the number of coalitions and their cardinalities in the coalition structure. It is written in the form

$$B_1 + B_2 + \ldots + B_\kappa, \text{ where } B_\iota \in \mathbb{Z}^+ \text{ and } \sum_{\iota=1}^{\kappa} B_\iota = m$$

Our work proposes coalition structure pattern in breaking manner as the following. Given a set of 6 agents, for example, the first pattern is 6 in layer $L_1$. There can be just one coalition, which is the grand coalition, whose cardinality is 6. In the next layer, $L_2$, the grand coalition will be broken into 2 coalitions by splitting a member from the grand coalition into the new coalition. Hence the pattern is 5 + 1. The next pattern is 4+2 and 3+3. The pattern in each layer cannot grow once the difference between each pair of coalitions' cardinalities is $\leq 1$. Then the pattern breaks into the next layer, i.e., 4 + 1 + 1, 3 + 2 + 1, 2 + 2 + 2. The last pattern is obviously 1 + 1 + 1 + 1 + 1 + 1. The pattern breaking process for 6 agents is shown below:

| No. of coalitions | 1 | 2 | 3 | 4 | 5 | 6 |
|---|---|---|---|---|---|---|
| Patterns | 6 | 5 + 1 | 4 + 1 + 1 | 3+1+1+1 | 2+1+1+1+1 | 1+1+1+1+1+1 |
| | | 4 + 2 | 3 + 2 + 1 | 2+2+1+1 | | |
| | | 3 + 3 | 2 + 2 + 2 | | | |

Agents can use best coalitions to generate coalition structures by following these patterns. By using the best coalitions alone, agents will achieve some coalition structures whose best one will be close to the optimal one. Using more coalitions, i.e., the second best, third best and so on, coalition structure values can be improved.

### 4.3.4 Generating Coalition Structures

Once each agent finishes its deliberation in the first stage, it exchanges all the coalitions generated with all other agents. It then uses the pattern to generate coalition structures. Starting with the best coalitions, it follows the patterns layer by layer from left to right and from top to bottom in each layer. For each pattern, the agent will choose a combination of its own best coalitions and those it received from other agents to generate coalition structures. For example, with a pattern of 4 + 3 + 2, the agent will place it's best coalition of cardinality 4 as the first coalition of that coalition structure. One of the best coalitions of cardinality 3, whose members are not in the first coalition, will be placed as the second coalition. One of the best coalitions of cardinality 2, whose members are not in the first two coalitions will be placed as the coalition structure as the last coalition. In the case the agent can not find appropriate coalitions to fit in, it places an empty set instead. The coalition structure value is the sum of those coalition values. In each round of proceeding through all patterns, an agent can extend the scope of best coalitions involved one by one. It, for example, generates the coalition structure using only the best coalitions in the first round. It then uses the best plus the second best coalition for the second round, and so on. The algorithm for generating coalition structures is shown in algorithm 13:

### 4.3.5 An Example of Generating Coalition Structure

This section gives an example of how this algorithm works. Let the system be composed of a set of four agents: $A = \{a_1, a_2, a_3, a_4\}$. After the first deliberation process, all the coalition values are computed and sent across. Their values are the following:

**Algorithm 13** Generating Coalition Structures

```
exchange best coalitions with all other agents
sort coalitions for each cardinality by their coalition values in descending order
generate patterns for each layer
set bestcoal to 1
while time is available do
    insert the bestcoal coalitions for each CScardinality
    for all layers do
        for all patterns do
            generate combinations of best coalitions in CScardinality
        end for
    end for
    increase bestcoal by 1
end while
```

$v_1 = 8$ $v_{12} = 13$ $v_{123} = 21$ $v_{1234} = 22$
$v_2 = 12$ $v_{13} = 16$ $v_{124} = 23$
$v_3 = 13$ $v_{14} = 10$ $v_{134} = 16$
$v_4 = 6$ $v_{23} = 18$ $v_{234} = 19$
$v_{24} = 20$
$v_{34} = 15$

After exchanging the coalitions generated among each other, each agent can select for each cardinality its best coalition. Let's assume that agents only operate on the best coalitions. Agents' best coalitions are the following:

| Cardinality | $a_1$ | $a_2$ | $a_3$ | $a_4$ |
|---|---|---|---|---|
| 1 | $v_1$ | $v_2$ | $v_3$ | $v_4$ |
| | 8 | 12 | 13 | 6 |
| 2 | $v_{13}$ | $v_{24}$ | $v_{23}$ | $v_{24}$ |
| | 16 | 20 | 18 | 20 |
| 3 | $v_{124}$ | $v_{124}$ | $v_{123}$ | $v_{124}$ |
| | 23 | 23 | 21 | 23 |
| 4 | $v_{1234}$ | $v_{1234}$ | $v_{1234}$ | $v_{1234}$ |
| | 22 | 22 | 22 | 22 |

For the system of 4 agents, the breaking patterns of coalitions are the following:

| No. of coalitions | 1 | 2 | 3 | 4 |
|---|---|---|---|---|
| Patterns | 4 | 3 + 1 | 2 + 1 + 1 | 1+1+1+1 |
| | | 2 + 2 | | |

Using the algorithm in the second deliberation process, each agent's coalition structures computed are shown below. Each agent will achieve the same optimal coalition structure whose value is 41.

| $a_1$ | $a_2$ |
|---|---|
| $CS_{1234} = 22$ | $CS_{1234} = 22$ |
| $CS_{124,3} = 23 + 13 = 36$ | $CS_{124,3} = 23 + 13 = 36$ |
| $CS_{1,234} = 8 + 0 = 8$ | $CS_{2,134} = 12 + 0 = 12$ |
| $CS_{13,24} = 16 + 20 = 36$ | $CS_{24,13} = 20 + 16 = 36$ |
| $CS_{13,2,4} = 16 + 12 + 6 = 34$ | $CS^*_{24,1,3} = 20 + 8 + 13 = 41$ |
| $CS_{1,23,4} = 8 + 18 + 6 = 32$ | $CS_{2,13,4} = 12 + 16 + 6 = 34$ |
| $CS_{1,2,34} = 8 + 12 + 0 = 20$ | $CS_{2,3,14} = 12 + 13 + 0 = 23$ |
| $CS^*_{1,3,24} = 8 + 13 + 20 = 41$ | $CS_{2,1,34} = 12 + 8 + 0 = 20$ |
| $CS_{1,2,3,4} = 8 + 12 + 13 + 6 = 39$ | $CS_{1,2,3,4} = 8 + 12 + 13 + 6 = 39$ |

| $a_3$ | $a_4$ |
|---|---|
| $CS_{1234} = 22$ | $CS_{1234} = 22$ |
| $CS_{123,4} = 21 + 6 = 27$ | $CS_{124,3} = 23 + 13 = 39$ |
| $CS_{3,124} = 13 + 23 = 26$ | $CS_{4,123} = 6 + 21 = 27$ |
| $CS_{23,14} = 18 + 0 = 18$ | $CS_{24,13} = 20 + 16 = 36$ |
| $CS_{23,1,4} = 18 + 8 + 6 = 32$ | $CS^*_{24,1,3} = 20 + 8 + 13 = 41$ |
| $CS^*_{3,1,24} = 13 + 8 + 20 = 41$ | $CS_{4,13,2} = 6 + 16 + 12 = 32$ |
| $CS_{3,2,14} = 13 + 12 + 0 = 25$ | $CS_{4,23,1} = 6 + 18 + 8 = 32$ |
| $CS_{1,2,3,4} = 8 + 12 + 13 + 6 = 39$ | $CS_{1,2,3,4} = 8 + 12 + 13 + 6 = 39$ |

## 4.4 Experiments

### 4.4.1 Generating Coalitions

We tested of our algorithm within a range of $10 - 100$ agents. In each round, the agents number increased by $5$. The number of goods and resources are equal and increase by $1$ in every $2$ rounds. In each round, the technology matrix, agents' resources and cooperation costs among agents are randomly generated with uniform distribution. The number of each resource $\alpha_{ij}$ in the technology matrix is in the range $0-10$. The prices of the goods are in the range of $10 - 20$ while the cooperation costs are in the range of $0$ and the number of agents in that round, e.g., $10, 15, \ldots$. As our algorithm deals with non-superadditive environments, this setting tends to increase the cooperation cost of a coalition as its size grows. Hence it

| No. of Agents | No. of Goods Resources | Exhaustive Search | Our Search |
|---|---|---|---|
| 10 | 4 | 781 | 121 |
| 15 | 4 | 42269 | 123 |
| 20 | 5 | 1272703 | 197 |
| 25 | 5 | 5092317 | 234 |
| 30 | 6 | 19384629 | 607 |
| 35 | 6 | 80429663 | 1608 |
| 40 | 7 | NA | 1696 |
| 50 | 8 | NA | 4730 |
| 60 | 9 | NA | 13346 |
| 70 | 10 | NA | 24298 |
| 80 | 11 | NA | 23276 |
| 90 | 12 | NA | 26933 |
| 100 | 12 | NA | 81845 |

Table 4.1: This table compares the average deliberation time of each agent using our algorithm against exhaustive search. Our algorithm outperforms exhaustive search after the number of agents exceeds 35 (exhaustive time not available—NA).

forces agents to work harder to form profitable coalitions and to achieve optimal coalition structure. Both algorithms use the Simplex algorithm to find the optimal solution for each coalition. The revenue generated is subtracted to achieve the coalition's profit.

The Table 4.1 compares the average deliberation time agents spent using exhaustive search to that using our algorithm. The time is measured in milliseconds. We observed that exhaustive search hardly makes any progress after the number of coalitions generated exceeded 2.5 millions. As shown in the table, the time spent on deliberation using exhaustive search was approximately doubled as the number of agents increased by 1. With 20 agents, the time spent on deliberation using exhaustive search is far larger than that using our algorithm. Our computer system could not carry on experiments any further after we reached 35 agents using exhaustive search. We continued experiment using our algorithm until the number of agents reached 100. (Although we carried on the experiment up to 300 agents, the results are not shown here.) Since the number of coalitions generated are small, the optimal coalition structure can be found more rapidly.

Having pruned a large number coalitions, the number of remaining coalitions are small. Hence the number of coalition structures are small. Applying our algorithm can intuitively achieve optimal coalition structure in timely fashion.

### 4.4.2 Generating Optimal Coalition Structures

We conducted experiments where agents executing our algorithm against exhaustive search within the range of $10 - 50$ agents. We compared the performance of both algorithms in terms of number of partitions generated and elapsed time of generating optimal coalition structures. Since existing exhaustive search algorithms, e.g., [77], does not specify how exactly the partitions are generated, we generate partitions for exhaustive search by $i$) increasing the number of blocks, e.g., $1, 2, \ldots, n$, and $ii$) for each block, the coalition size will be propagated from left to right. For example, if there are 3 agents, the partitions generated will be {1,2,3}, {1,2}{3}, {1,3}{2}, {2,3}{1} and {1}{2}{3}. In each round, the number of agents increases by $5$. The number of goods and resources are equal and increase by $1$ in every $2$ rounds. The technology matrix, agents' resources and cooperation costs among agents are randomly generated with a uniform distribution. The number of each resource $\alpha_{ij}$ in the technology matrix is in the range $0 - 10$. The prices of the goods are in the range $10 - 20$, while the cooperation costs are in the range of $0$ and the number of agents in that round, e.g., $10, 15, \ldots$. As our algorithm deals with non-superadditive environments, this setting tends to increase the cooperation cost of a coalition as its size grows. Hence it forces agents to work harder to form profitable coalitions and to achieve optimal coalition structures. Both algorithms use the Simplex algorithm to find the optimal solution for each coalitions. The revenue generated is subtracted to achieve the coalition's profit.

Figure 4.2 compares the performance of our algorithm against that of exhaustive search. The left y-axis is the number of coalition structures generated while right y-axis is the elapsed time spent for generating optimal coalition structures in milliseconds. The empirical results show that our algorithm performs significantly better than exhaustive search. We observed that the exhaustive algorithm hardly makes progress after the number of agents exceeds 40. As shown in the figure, the number of coalition structures generated by the exhaustive algorithm is much larger than that of our algorithm. Furthermore, the elapsed time for generating optimal coalition structures by the exhaustive search is also much larger than that of our algorithm. Since our computer system could not carry on experiments using the exhaustive search for a large number of agents, we limit the comparison only for 50 agents. However, we continued experiment using our algorithm until the number of agents reached 100 but the results are not shown here.)

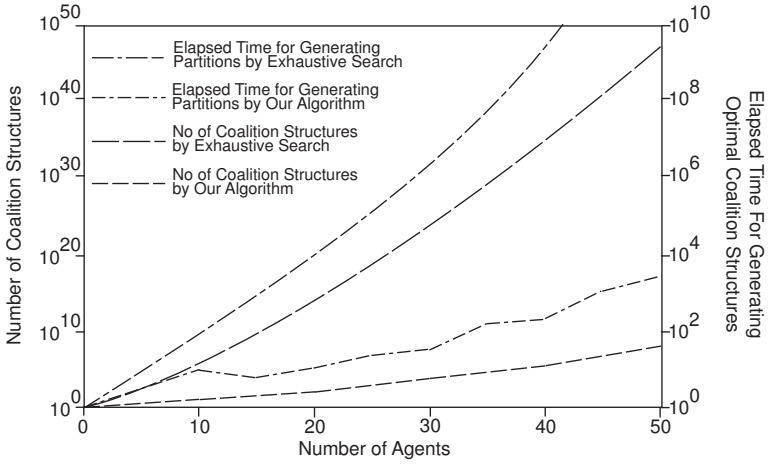


Figure 4.2: **Empirical Results** This graph shows the number of coalition structures generated and elapsed time for generating the optimal coalition structures of our algorithm against those of exhaustive search.

## 4.5 Conclusion

Coalition formation is an important area of research in multi-agent systems. The problem of generating optimal coalition structures, the partitioning of a set of agents such that the sum of all coalitions' values within the partitioning is maximal, is an important issue in the area. The small number of existing studies assume each coalition value is known a priori. Such assumption is impractical in real world settings. Furthermore, finding all coalition values becomes intractable for a relatively small number of agents.

We proposed a distributed branch-and-bound anytime algorithm for computing optimal coalition structure for linear production domains among fully cooperative agents. Instead of assuming that each coalition value is known a priori, our algorithm tries to reduce the number of coalitions. We extend our previous algorithm in the deliberation process in order to improve the performance. Non-profitable coalitions are not generated by the deliberation algorithm. Then the information about remaining coalitions will be exchanged among agents. Lastly, each agent uses an existing algorithm [77] to compute optimal coalition structures.

The empirical results show that our algorithm help generate the optimal coalition structures much faster than exhaustive search. Our algorithm dramatically reduces the number of coalitions generated hence reducing the number of coalition structures. As a result, the elapsed time of generating the coalition structures is relatively small.

So, the work presented in this chapter has achieved the second objective of this research, namely, to adapt the algorithm presented in Chapter 3 so as:

2. to solve OCS problems in a linear environment where coalition values and coalition

structure values are not known a priori but must be calculated.

Having successfully met this second objective, our goal is to deploy the best-first anytime algorithm to a number another complex OCS domains in which the coalition values are not known a priori. In the next chapter we will adapt the algorithm for use in an NP hard, non-linear environment where coalition costs and coalition structure values are not known a priori.

# Chapter 5

# Non-Linear Optimal Coalition Structure

## 5.1 Introduction

The previous 2 chapters have achieved the first and second objectives of this research by developing a best-first, anytime algorithm that is an efficient solution for OCS problems in environments where coalition values are known a priori, and adapting that algorithm for use in a linear environment, which is much more computationally complex. This chapter will continue our development of the original algorithm for use in an NP hard, non-linear environment where coalition costs and coalition structure values are not known a priori, thereby achieving the third objective of this research.

So far, we have proposed a heuristics algorithm for generating optimal coalition structures in generic domains in chapter 3 and a distributed algorithm to compute (near) optimal coalition structures in linear production domains. In this chapter, we study optimal coalition structures in non-linear domains. Here, we use an example of logistics providers distributing goods from manufacturing sites to end customers. This is because optimizing routes for logistics providers is a well known hard problem in real world setting.

Typical logistics providers operations are to distribute goods from depots and try to minimize costs by deploying as smaller number of trucks as possible, as well as try to meet with the customers' dateline requirements. Here, we consider the operation of small independent logistics providers, whose individual resource is merely a truck, in a small but complex economy system. Note that these trucks are independent economy units that they have to make their own living. An example of this kind of setting is the work of Shehory et.al. [89]. Hence it is important that agents have fair opportunity and fair share on their contribution.

We consider distributing goods from a manufacturing site is a task. Here, we allow agents to form a non-overlapping coalition for a given task. We want to examine how economically agents can address this problem in such a setting. We propose a set of distributed algorithms to tackle the task. As commonly seen in the literature [48, 49, 84, 89, 91], agents begin

by choosing the tasks it can perform most efficiently, i.e. the total distance of the routes performed individually in the the coalition is minimal. Given a task, the size of the coalitions who can perform the task varies. Hence, agents need to find the most appropriate coalitions for all tasks. By assigning a coalition to each task, while all agents are part of the coalitions and all tasks are assigned with coalitions, we can consider this as a task-agent coalition. Each task-agent coalition will incur a distribution cost. We are interested in minimizing the total cost of the system, which is another level of optimal coalition structure.

Note that this problem shares some degree of similarity with other problems in the literature. It is similar to the Traveling Salesman problem [66, 4] in that a vehicle has to take a circular route to deliver the goods. However, the vehicle, here, has to collect the good from the manufacturing site, first, before it can distribute the goods. With regards to the Vehicle Routing problem{Golden-2008}, each vehicle has a set of certain capability constraints. However, each agent, here, requires cooperation from other agents. With regards to Sandholm et.al.'s work [75], each agent in each possible coalition knows the set of the tasks they are required to do (agents just need to optimize the coalition values and the optimal coalition structures can be computed based on them.) However, each agent, here, seeks for a set of coalitions that are efficient for a given task. The optimal coalition structures will be searched from this larger set of coalitions. With respect to Shehory et.al. [89], each agent is assigned a small set of tasks and the number of agents in the system is quite small. On the other hand, our agents need to find a set of the most appropriate tasks (forming task-oriented coalitions before searching for optimal coalition structures) and the number of agents involved is larger.

Rahwan et.al. [69] propose an algorithm for computing coalition values among cooperative agents. However, it is not applicable here because the algorithm works in practice only for a small number of agents, i.e., it has to scan all the coalition values as input. Here, the number of coalitions involved is much larger, scanning all coalition values are impractical. For example, choosing a coalition of 10 agents from 100 agents requires at least ${}^{100}C_{10} = 1.731E + 13 \approx 17.3$ trillions bytes, while choosing a coalition of 11 agents from 100 agents requires at least ${}^{100}C_{11} = 1.42E + 14 \approx 142$ trillions bytes. We decide to follow the same principle proposed in previous chapters, that is to compute only a relatively small number of coalitions whose values are more relevant to the process of computing optimal coalition structure values. Agents may choose any appropriate heuristics to generate coalitions.

We are interested in exploring the difference between

1. various time allocation strategies.

2. various data distributions.

It is common in the literatures on multi-agent systems that agents optimize their tasks locally and deliberately exchange their tasks to increase the performance of the systems [48, 49, 84, 89, 91]. Postman protocol, for example, allows agents to exchange their task one by one. The tasks being exchanged are usually costly for the agents who want to exchange. We explore how the system performs if the numbers of tasks being exchanged are larger.

## 5.2 Distributed Algorithm for Distributing Goods

Here, we propose a distributed anytime algorithm for agents to cooperatively execute the tasks. Each agent seeks the most efficient tasks it can perform. This can be achieved quickly by ranking all the tasks by the distance between the location of the agent and the locations of the sources of the tasks in ascending order, i.e., the agent prefers the nearest source. For each of the sorted tasks, the agent tries to match it with set of coalitions, each of which includes itself, from the smallest one possible to the largest one necessary to complete the task.

The cost to jointly execute a task can be different for each agent. This is due to the fact that agents' capabilities and circumstances are different. This simply means there are more constraints governing the assignment of agents to the tasks, resulting in there are fewer agents available to particular tasks. In this case, it is relatively easy to compute optimal coalition structures. On the other hand, if agents have equal capability, there more agents capabled of executing particular tasks. Hence it is harder to compute optimal coalition structure. In this chapter, we refer to assigning agents to execute tasks such that the total cost of the system is minimal as the *optimal task-agent coalition structure* (OTCS).

As we have done in the previous chapter, we compute the coalition values based on heuristics that can lead to coalitions of high value. (Instead of following the lexicographic order [50], we are interested in computing high-valued coalitions under time constraints.) In computing the optimal task-agent structure, there are two levels of intervening deliberation agents have to do.

1. The first one is to optimize possible solutions for each task.

2. The second one is to compute OTCS.

Since optimization of possible solutions is a hard problem itself, agents have to ensure they do not spend too much time on this deliberation and do not have enough time to deliberate for OTCS.

We assume agents are under time constraints in order to finish their tasks. Hence, agents may consider a swapping strategy which it considers to be the most appropriate. It may

use the maximum change strategy to quickly achieve a solution or it may use the thorough strategy to gradually swap the task.

The calculation of coalitions associated with the tasks can be done distributedly. Since the computation for optimal distribution of the goods can be done by any computing unit, agent can evenly share the computation for the optimal routes for each task.

### 5.2.1 Setting

There are two levels of coalitions in our setting: $i$) task-plan in order to find the best solutions for each task at each point in time, and $ii$) task-agent in order to assign to each best task-plan a coalition of agents such that the total cost is minimal.

#### Task-Plan Solution

We define a graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ where $\mathcal{V}$ is a set of vertices and $\mathcal{E}$ is a set of edges connecting each pair of vertices with a certain distance. The graph represents an economic map of a road network, i.e., a vertex represents a location in the road network, a vertex represents the cost in term of the shortest distance achieved optimally between each pair of locations. We define a task as a tuple $T = \langle \mathcal{S}, \{\mathcal{D}\} \rangle$, where $\mathcal{S} \in \mathcal{V}$ is the source and $\{\mathcal{D}\} \subseteq \mathcal{V}$ is a set of destinations associated with $\mathcal{S}$. We refer to the number of $\mathcal{D}$s in $T$ as the size of $T$ and denote it by $|T|$. We assume the smallest $|T| = 1$. The set of all tasks is denoted by $\mathcal{T}$.

We define $\mathfrak{L}$ a set of $n$ logistics provider (LP) agents. Each of these agents, $\mathcal{L} \in \mathfrak{L}$, is a truck with the same capacity load and travel distance. Agents are to cooperatively deliver the goods as per requests. We assume the number of agents is at least equal to the number of tasks and the maximal number of agents is not greater than $\sum_{T \in \mathcal{T}} |T|$. A task $T$ can be partitioned into $1 \leq p \leq |T|$ parts, each of which to be executed by an agent. The agents assigned to a task is then a coalition. We shall refer to a set of partitions of the same $p$ parts as *partition* $P_p$. We shall refer to each instance of $P_p$ as *partition instance* and denote it by $P_{p.i}$, where $i$ is the lexicographical order index of the partitions of the same $p$ size. We denote by $|P_p|$ the number of all instances $P_{p.i}$ in $P_p$[1]. We shall denote the $j$-th part of $P_{p.i}$ by $P_{p.i.j}$, starting from left to right. Hence $P_{p.i.j}$ is the index indicating the number of destinations in the $j$-th part of $P_{p.i}$ and we denote such a number by $|P_{p.i.j}|$.

For example, a task of 5 destinations can have 5 partitions, which can be broken down to 7 partition instances altogether as shown below:

[1]The exact number of $P_p$ and integer partition are well explained in [50]

| Partition | Partition Instance | Integer Partition | Description |
|---|---|---|---|
| $P_1$ | $P_{1.1}$ | 5 | One part of 5 destinations |
| $P_2$ | $P_{2.1}$ | 4+1 | One part of 4 and one part of 1 destinations |
| | $P_{2.2}$ | 3+2 | One part of 3 and one part of 2 destinations |
| $P_3$ | $P_{3.1}$ | 3+1+1 | One part of 3 and two parts of 1 destinations |
| | $P_{3.2}$ | 2+2+1 | Two parts of 2 and one part of 1 destinations |
| $P_4$ | $P_{4.1}$ | 2+1+1+1 | One part of 2 and three parts of 1 destinations |
| $P_5$ | $P_{5.1}$ | 1+1+1+1+1 | Five parts of 1 destinations |

Here, partition $P_1, P_4$ and $P_5$ has just one instance, while $P_2$ and $P_3$ which have two instances each, i.e. $P_{2.1}, P_{2.2}, P_{3.1}$ and $P_{3.2}$ respectively. Given $P_{2.1}$, for example, we partition the whole task into 2 parts: one part of 4 destinations and one part of 1 destination. In other words, we may assign a coalition of 2 agents for this task: one agent to execute $P_{2.1.1}$ which has 4 destinations, and the other one to execute $P_{2.1.2}$ which has 1 destination. Alternatively, we may assign 3 agents to execute the task: the first agent to execute $P_{3.1.1}$ which has 3 destinations, the second agent to execute $P_{3.1.2}$ which has 1 destination, and the third agent to execute $P_{3.1.3}$ which has 1 destination. We shall discuss how can we partition, i.e. group destinations, below.

Note that partitioning a task itself is of the same complexity class as partitioning coalition in order to compute OCS as we have discussed in chapter 3. Whereas traditional OCS works assume coalition values are known a priori, this chapter takes into account computational time consumed to calculate coalition values.

In each $P_{p.i.j}$, a collection of $|P_{p.i.j}|$ destinations can be chosen to build a route on which any agent can travel and distribute the goods. Let $D_{p.i.j}$ be a set of destinations yet to be chosen into $P_{p.i.j}$. Hence there are ${}^{|D_{p.i.j}|}C_{|P_{p.i.j}|}$ alternative combinations. For each of these, we can have $|P_{p.i.j}|!$ alternative routes, which we denote as $R_{p.i.j}$. Given $D_{p.i.j}$, we can compute the number of alternative routes, $|R_{p.i.j}|$, as follows:

$$\frac{|D_{p.i.j}|!}{(|D_{p.i.j}| - |P_{p.i.j}|)! \cdot |P_{p.i.j}|!} \cdot |P_{p.i.j}|!$$

Hence, for each $P_p$ there are

$$\prod_{i=1}^{|P_p|} \prod_{j=1}^{p} ({}^{|D_{p.i.j}|}C_{|P_{p.i.j}|})$$

alternatives. We shall refer to each of these alternative as a *plan*, $\mathcal{P}_p$. Given an optimization technology and a computation time $t$, we denote $R^t_{p.i.j}$ as the optimized route with

minimal cost (distance). We denote this optimal cost by $C^t_{p.i.j}$. Hence, the execution cost of a coalition of agents for a $\mathcal{P}_p$ is

$$CE^t_{\mathcal{P}_p} = \sum_{j=1}^{p} C^t_{p.i.j}$$

For each partition of task $T$, we are interested in achieving the best plan $\mathcal{P}^*_{T,p}$, whose cost

$$CE^*_{\mathcal{P}_p} = arg_{min} C^t_{\mathcal{P}_p}$$

is minimal.

Agents are to cooperatively compute each $T.\mathcal{P}_p$. Rahwan et.al. [67] propose an algorithm for computing coalition values among cooperative agents. However, that algorithm seems inapplicable here because it computes the values of all coalitions, whose number is too large in our setting.

**Task-Agent Coalition**

The next step is to assign appropriate coalitions of agents to tasks to achieve the overall minimal cost for the system. We have defined a set of logistic providers $\mathfrak{L}$. There is a location function $Loc : \mathfrak{L} \rightarrow \mathcal{V}$ which associates an agent $\mathcal{L} \in \mathfrak{L}$ to a vertex $\mathfrak{n} \in \mathcal{V}$, where the agent is located. Let $RT \subseteq \mathfrak{T}$ be the set of tasks yet to be assigned with agents. Let $RL \subseteq \mathfrak{L}$ be the set of agents yet to be assigned tasks. In order to execute a task $T$, a coalition $S \subseteq \mathfrak{L}$ of $1 \leq |S| \leq (|RL| - (|RT| - 1))$ agents have to travel the source of the task and collect goods before distributing them. For each agent $\mathcal{L}$, this will incur the *access cost*, $CA_{T,\mathcal{L}}$, to the agent. Hence the access cost for $S$ to $T$ is

$$CA_{T,S} = \sum_{\mathcal{L} \in \mathfrak{L}} CA_{T,\mathcal{L}}.$$

This cost is consistent and is independent of time. Let $\mathfrak{L}_{\mathcal{P}^*_{T,p}} \subseteq \mathfrak{L}$ be the set of available agents for assigning to $\mathcal{P}^*_{T,p}$, there are $^{|}\mathfrak{L}_{\mathcal{P}^*_{T,p}}|C_{\mathcal{P}^*_{T,p}}$ ways of assigning a coalition of agents to $\mathcal{P}^*_{T,p}$. We shall refer to each pair of assigning $S \subseteq \mathfrak{L}_{\mathcal{P}^*_{T,p}}$ to $\mathcal{P}^*_{T,p}$ as a *task-agent* coalition and denote it by $S_{\mathcal{P}^*_{T,p}}$. Hence the total cost of executing plan $\mathcal{P}^*_{T,p}$ by $S$ is

$$C_{\mathcal{P}^*_{T,p},S} = CA_{T,S} + C^*_{\mathcal{P}_p}$$

### Task Coalition Structure

Here, we are not only interested in finding the cheapest assignment to each $T$, but we are also interested in finding the cheapest assignment to $\mathfrak{T}$. We define a *task coalition structure*,

$$TCS = \{S_{\mathcal{P}^*_{T,p}} | \text{ where } \bigcup_T = \mathfrak{T}, \text{ and } T_i \cap T_j = \emptyset, \text{ and } \bigcup_S = \mathfrak{L}, \text{ and } S_i \cap S_j = \emptyset\},$$

is a set of task-agent assignments where all tasks are assigned with a a unique coalition of agents and all agents are assigned to tasks. We define the total assignment cost,

$$V(TCS) = \sum C_{\mathcal{P}^*_{T,p},S}$$

the sum of all assignments in a task coalition structure. We are interested in finding a task coalition structure such that the sum of total cost is minimal, i.e.,

$$TCS^* = arg_{min} V(TCS)$$

### Best Task-Agent Assignment

In chapter 3, we have defined the *best* candidate coalition, based on the ratio between the coalition value and its cardinality, in order to place the next coalition into CS. Here we apply the same principle. In order to find the most appropriate $S^*_{\mathcal{P}^*_{T,p}}$ as the next assignment to one of the remaining tasks, we define the *reduction contribution* as

$$\bar{A} = \frac{C_{\mathcal{P}^*_{T,p},S}}{|S|}.$$

Hence the *best task-agent* assignment is

$$S^*_{\mathcal{P}^*_{T,p}} = arg_{min} \bar{A},$$

whose reduction contribution is the lowest among all the possible assignments.

### Time Allocation Strategy for Overall Deliberation

We define a time allocation strategy as a tuple $\mathcal{T}_{ST} = \langle \mathcal{T}_{PA}, \mathcal{T}_{TA} \rangle$, where $\mathcal{T}_{PA}$ is the percentage of remaining time to be allocated to the deliberation to find $\mathcal{P}^*_{T,p}$ for each $T$, $\mathcal{T}_{TA}$ is the percentage of remaining time to be allocated to the deliberation to find $S^*_{\mathcal{P}^*_{T,p}}$, and $\mathcal{T}_{PA} + \mathcal{T}_{TA} \leq 100$. Agents may choose to split the remaining time equally, i.e., $\mathcal{T}_{ST} = \langle 50, 50 \rangle$, which will give agent just one round of each deliberation. However, this strategy

could also lead to a dangerous situation because the exact computing of the deliberation may take more time than it is allocated and lead to the overtimed solution. Alternatively, agents may choose to spare some time for the exact computing, and repeat the process of both deliberations for the remaining time. Such strategies may include $\langle 50, 40\rangle$, $\langle 40, 40\rangle$, $\langle 40, 30\rangle$, $\langle 30, 30\rangle$, $\langle 30, 25\rangle$, $\langle 25, 25\rangle$, which would recursively leave spare time for deliberations in later rounds as $10\%$, $20\%$, $30\%$, $40\%$, $50\%$, respectively.

**Time Allocation for Task-Plan Deliberation**

There are two issues for time allocation in the task-plan deliberation. $i$) Since planning for optimal routes for each task is a variant of the Vehicle Routing problem, which is a hard problem, time allocation for each task has to be distributed to each task efficiently and has to take into account the trade off between the time spent on the computation and the quality of the result. $ii$) The distribution of each task has to meet the time constraints required by customers and all agents have to play their parts in distributing goods. We invent a heuristic strategy which will allocate time to each task based on the potential computation workload on each task, i.e. the sum of the size of the search space in each partition of the task.

In general, the algorithm needs to work out how to slice the available time for the optimization of each task. This is depending on the size of each task, which specifies the size of search space in each of its partition. Intuitively, the time allocation should be based on the size of search space but we simply cannot do that directly because the numbers involved will be too large. For example, choosing 50 from 100 cities or agents would involve

$$\begin{aligned} {}^{100}C_{50} &= 1.00891345 \times 10^{29} \\ &\approx 1.0 \times 10^2 \cdot 10^3 \cdot 10^6 \cdot 10^6 \cdot 10^6 \cdot 10^6 \\ &\approx 1.0 \times 10^2 2^{10} \cdot 2^{20} \cdot 2^{20} \cdot 2^{20} \cdot 2^{20} \\ &\approx 1.0 \times 10^2 2^{90}. \end{aligned}$$

This number is too large to be computed efficiently in a typical computer.[2]

Instead, we consider just the partition numbers of each task. We compute for the weight of each partition and allocate the time to optimize the partition based on its proportional weight to the total weight of the task. Firstly, we compute the product of each part $j$ in each partition instance $j$, $\prod |P_{p.i.j}|$. The weight for each partition is then defined by

$$W_p = \frac{\sum \prod |P_{p.i.j}|}{|i|},$$

[2]Java offers the class BigInteger which can handle large number but the performance is relatively slow.

where $|i|$ is the number of instances in $P_p$. The actual time allocation for the each $P_p$ in task $T$ is

$$A(W_p) = \frac{W_p}{\sum_{k=1}^{|T|} W_k}$$

Let $W_T = \sum_{k=1}^{|T|} W_k$ be the total weights of all $P_p$ $T$. Let $W_{\mathfrak{T}} = \sum_{T \in \mathfrak{T}} W_T$ be the total weights of all $T \in \mathfrak{T}$. The proportional time allocation out of the available time for task-plan deliberation is merely

$$A(W_T) = \frac{W_T}{W_{\mathfrak{T}}}.$$

Below is the example of how we can split the allocation time for each partition of a task $T, |T| = 10$. Note that the time will be allocated more to partitions, whose search space are larger.

| $P_{1.i}$ | | $P_{2.i}$ | | $P_{3.i}$ | |
|---|---|---|---|---|---|
| $p=1$ | $\prod \lvert P_{1.i.j}\rvert$ | $p=2$ | $\prod \lvert P_{2.i.j}\rvert$ | $p=3$ | $\prod \lvert P_{3.i.j}\rvert$ |
| 10 | 10 | 9+1 | 9 | 8+1+1 | 8 |
| | | 8+2 | 16 | 7+2+1 | 14 |
| | | 7+3 | 21 | 6+3+1 | 18 |
| | | 6+4 | 24 | 6+2+2 | 24 |
| | | 5+5 | 25 | 5+4+1 | 20 |
| | | | | 5+3+2 | 30 |
| | | | | 4+4+2 | 32 |
| | | | | 4+3+3 | 36 |
| $W_1 = \frac{\sum \prod \lvert P_{1.i.j}\rvert}{1}$ | 10 | $W_2 = \frac{\sum \prod \lvert P_{2.i.j}\rvert}{5}$ | 19 | $W_3 = \frac{\sum \prod \lvert P_{3.i.j}\rvert}{8}$ | 22.75 |
| $A(W_1) = \frac{W_1}{\sum W_p}$ | 9.154814691 | $A(W_2) = \frac{W_2}{\sum W_p}$ | 17.39414791 | $A(W_3) = \frac{W_3}{\sum W_p}$ | 20.82720342 |
| $P_{4.i}$ | | $P_{5.i}$ | | $P_{6.i}$ | |
| $p=4$ | $\prod \lvert P_{4.i.j}\rvert$ | $p=5$ | $\prod \lvert P_{p.i.j}\rvert$ | $p=6$ | $\prod \lvert P_{p.i.j}\rvert$ |
| 7+1+1+1 | 7 | 6+1+1+1+1 | 6 | 5+1+1+1+1+1 | 5 |
| 6+2+1+1 | 12 | 5+2+1+1+1 | 10 | 4+2+1+1+1+1 | 8 |
| 5+3+1+1 | 15 | 4+3+1+1+1 | 12 | 3+3+1+1+1+1 | 9 |
| 5+2+2+1 | 20 | 4+2+2+1+1 | 16 | 2+2+2+2+1+1 | 12 |
| 4+4+1+1 | 16 | 3+3+2+1+1 | 18 | | |
| 4+3+2+1 | 24 | 3+2+2+2+1 | 24 | | |
| 3+3+3+1 | 27 | 2+2+2+2+2 | 32 | | |
| 3+3+2+2 | 36 | | | | |
| $W_4 = \frac{\sum \prod \lvert P_{4.i.j}\rvert}{8}$ | 19.625 | $W_5 = \frac{\sum \prod \lvert P_{5.i.j}\rvert}{7}$ | 16.85714 | $W_6 = \frac{\sum \prod \lvert P_{6.i.j}\rvert}{4}$ | 8.5 |
| $A(W_4) = \frac{W_4}{\sum W_p}$ | 17.96632383 | $A(W_5) = \frac{W_5}{\sum W_p}$ | 15.43239929 | $A(W_6) = \frac{W_6}{\sum W_p}$ | 7.781592487 |
| $P_{7.i}$ | | $P_{8.i}$ | | $P_{9.i}$ | |
| $p=7$ | $\prod \lvert P_{p.i.j}\rvert$ | $p=8$ | $\prod \lvert P_{p.i.j}\rvert$ | $p=9$ | $\prod \lvert P_{p.i.j}\rvert$ |
| 4+1+1+1+1+1+1 | 4 | 3+1+1+1+1+1+1+1 | 3 | 2+1+1+1+1+1+1+1+1 | 2 |
| 3+2+1+1+1+1+1 | 6 | 2+2+1+1+1+1+1+1 | 4 | | |
| 2+2+2+1+1+1+1 | 8 | | | | |
| $\frac{\sum \prod \lvert P_{7.i.j}\rvert}{3}$ | 6 | $\frac{\sum \prod \lvert P_{8.i.j}\rvert}{2}$ | 3.5 | $\frac{\sum \prod \lvert P_{9.i.j}\rvert}{1}$ | 2 |
| $A(W_7) = \frac{W_7}{\sum W_p}$ | 5.492888815 | $A(W_8) = \frac{W_8}{\sum W_p}$ | 3.204185142 | $A(W_9) = \frac{W_9}{\sum W_p}$ | 15.43239929 |
| | | | | $P_{10.i}$ | |
| | | | | $p=10$ | $\prod \lvert P_{p.i.j}\rvert$ |
| | | | | 1+1+1+1+1+1+1+1+1+1 | 1 |
| | | | | $\frac{\sum \prod \lvert P_{10.i.j}\rvert}{1}$ | 1 |
| | | | | $A(W_10) = \frac{W_10}{\sum W_p}$ | 7.781592487 |

**Heuristic for Choosing Shrinking and Altering Point**

Although the algorithm to find $TCS^*$ is similar to algorithm 1 in principle, it cannot repeat shrinking and altering too often without significant improvement on $V(TCS)$ because the number of agent coalitions can be much larger in this case, ${}^{100}C_{50}$, for example, is too large for any typical computer as we have discussed. We argue that the shrink and alter points have to be different from algorithm 1. Instead of doing thorough search by altering and shrinking once the algorithm cannot extend anymore, we introduce a heuristic to find the alter and shrink point which is more appropriate to our setting. This heuristic repeatedly bi-partitions $TSC$ into sections. We define a bi-partitioned of a number $I \in \mathbb{I}^+, I > 1$ a tuple $\langle HH, LH \rangle$, where $HH = LH = \frac{I}{2}$ if $I$ is and even number, or $HH = \lceil \frac{I}{2} \rceil$, $LH = \lfloor \frac{I}{2} \rfloor$ otherwise. At

each time of the bi-partitioning, we refer to the partition as the level, $l$. We refer to each part of the partition as the section, $s$. The shrinking and altering of $TCS$ will take place at the position indicated by this bi-partition. We compute the weight for each bi-partition as the following

$$W_{I_l} = \frac{\prod I_{l,j}}{I_l}$$

The most appropriate bi-partition is

$$I_l^* = arg_{max} W_{I_l}$$

The present $TCS$ will be shrunk from task $|j|, |j-1|, \ldots, 1$. Each time $TCS$ is shrunk, the alter takes place at $T_{|j|}$ in order to assign the next best $S$ to the task, replacing the previous one.

Given, $|\mathfrak{T}| = 20$, we can bi-partition it into 5 level as shown below. By computing weight for each level, $I_2$ is the most appropriate level. The shrinking TCS and altering for a new $S$ will take place at $T_{16}, T_{11}, T_6, T_1$.

| I | Bi-Partition | $I_l$ | $\frac{\prod I_{l,j}}{I_l}$ |
|---|---|---|---|
| 20 | = (10+10) | $I_1$ | $\frac{100}{2^2} = 25$ |
| | = (5+5)+(5+5) | $I_2$ | $\frac{625}{2^4} = 39.0625$ |
| | = ((3+2)+(3+2))+((3+2)+(3+2)) | $I_3$ | $\frac{1296}{2^8} = 5.0625$ |
| | = (((2+1)+(1+1))+((2+1)+(1+1)))+(((2+1)+(1+1))+((2+1)+(1+1))) | $I_4$ | $\frac{16}{2^{16}} = 2.44 \times 10^{-}4$ |
| | = ((((1+1)+1)+(1+1))+(((1+1)+1)+(1+1)))+((((1+1)+1)+(1+1))+(((1+1)+1)+(1+1))) | $I_5$ | $\frac{1}{2^{20}} \approx 9.54 \times 10^{-07}$ |

### 5.2.2 Main Algorithm

In the beginning, the algorithm requires the time allocation strategy $\langle \mathcal{T}_{ST} \rangle$ and the time available for deliberations, which we shall refer to as the remaining time, $\mathcal{T}_R$. The function $AllocPlanTime(\mathcal{T}_{ST}, \mathcal{T}_R)$ will return the time allocated to the task plan deliberation process by $planDelibTime = \mathcal{T}_R/100 * \mathcal{T}_{PA}$. Agents spend at least $planDelibTime$ to solve the optimization problem in all tasks by calling the function . Once the solution deliberation process in this round is done, agents spend $assignDelibTime = \mathcal{T}_R/100 * \mathcal{T}_{TA}$ to compute the most efficient assignment task-agent for the time being. The algorithm then computes for $\mathcal{T}_R$ by deducting the time spent on both deliberation in that round from $\mathcal{T}_R$. If there is remaining time, the algorithm goes into the loop and repeat the deliberation processes again until $\mathcal{T}_R \leq 0$. The details processes are shown in algorithm 14.

**Algorithm 14** Main Algorithm: Construct task-agent structures by repeatedly deliberate for optimal task-cardinality and task-agent assignment

**Require:** $\mathcal{T}_{ST}$
**Require:** $\mathcal{T}_R$
1: $elapsedTime \leftarrow 0$
2: $startTime \leftarrow presentTime$
3: $planDelibTime \leftarrow AllocPlanTime(\mathcal{T}_{ST}, \mathcal{T}_R)$
4: $DelibTaskPlan(\mathfrak{T}, planDelibTime)$
5: $assignDelibTime \leftarrow AllocAssignTime(\mathcal{T}_{ST}, \mathcal{T}_R)$
6: $DelibTaskAssign(Tasks, assignDelibTime$
7: $elapsedTime \leftarrow presetTime - startTime$
8: $\mathcal{T}_R \leftarrow \mathcal{T}_R - elapsedTime$
9: **while** ( **do**$\mathcal{T}_R > 0$)
10: $planDelibTime \leftarrow AllocateTime(\mathcal{T}_{ST}, \mathcal{T}_R)$
11: $DelibTaskPlan(\mathfrak{T}, planDelibTime)$
12: $assignDelibTime \leftarrow AllocateTime(\mathcal{T}_{ST}, \mathcal{T}_R)$
13: $DelibTaskAssign(\mathfrak{T}, assignDelibTime$
14: $elapsedTime \leftarrow presetTime - startTime$
15: $\mathcal{T}_R \leftarrow \mathcal{T}_R - elapsedTime$
16: **end while**

### 5.2.3 Algorithm to Deliberate Task-Plan

Here, agents have to split the available time to compute each $\mathcal{P}^*_{T,p}$. In the simplest case, where $n$ is equal to $|\mathfrak{T}|$ and $|T|$ is equal for each $T \in \mathfrak{T}$, each agent may be responsible for the task deliberation based on lexicographic order. In the case where $|T|$ is different for each $T \in \mathfrak{T}$, the computation should be distributed evenly [67]. Every computed plan $\mathcal{P}^*_{T,p}$ needs to be distributed to every agent. Although the shortest route from the source of each task to go through all of its destinations can possibly be derived by a single agent, the system is under time constraint as well as the agents themselves are limited by their own capabilities. Therefore, agents need to form coalitions to relax these constraints. On the other hand, forming too large coalitions can lead to poor performance of the overall system because the cost can be too high. Assigning $|T|$ agents, for example, to $T$ simply means the execution cost of the task is as twice as much of the sum of the distance between each pair of the source and each destination. Since there are a large number of possible $\mathcal{P}^*_{T,p}$ to be computed, agents can also apply strategies of time division to the computation of these $\mathcal{P}^*_{T,p}$. One strategy agents can use to reduce the computational time for $\mathcal{P}^*_{T,p}$ is to allocate time based on the size of the search space of the plan and quality of the solution it may achieve. Agents may stop deliberating execution cost on these oversize $P_p$ if it is certain that the best $\mathcal{P}^*_{T,p}$ it can achieve, with respect to time constraint, is worst than the solutions it

has already achieved with smaller plans. Note that we treat the optimization part for $\mathcal{P}^*_{T,p}$ as the underpinning precess. Any algorithm for the Traveling Salesman problem or the Vehicle Routing problem can be applied here. The main focus of this chapter is on the assignment of coalitions of agents to tasks.

The detailed process of allocating time to task-plan is described in algorithm 15. It takes the $remainTime$ as an input. The algorithm initializes two arrays and one variable to store computed weights for $P$s, $T$s, and $\mathfrak{T}$. The algorithm then finds the weight of each $P_p$, total weight for each $T$, and total weight for $\mathfrak{T}$. In each of these processes, we do not show details on how to compute the weights because it is quite straightforward from what we have described in section "time allocation for task-plan solution". The function $computeWeight()$ serves this purpose as a black box that returns just the weight of the respective $P$. Both the total weight of each $T$ and $\mathfrak{T}$ will also be accumulated at the same time. Once all the weights are computed, the algorithm then goes to each $T$, finds the exact time for optimizing this $T$. The allocated time for this $T$ will then be allocated to each $P$, which will be optimized for the the best route by the function $optimizePartn()$. This function is the call to the underpinning technology that takes $T$, $P$, and the available time as inputs, and return $\mathcal{P}$ for the given $P$. The detailed processes are shown in algorithm 15.

**Algorithm 15** Optimize for the best possible plans for each task

**Require:** $remainTime$
**Require:** $maxTaskSize$
**Require:** $\mathfrak{T}$
1: init $planWeightArray[|\mathfrak{T}|][maxTaskSize]$
2: init $taskWeightArray[|\mathfrak{T}|]$
3: $TotalWeight \leftarrow 0$
4: **for** each $T \in \mathfrak{T}$ **do**
5: **for** each $P \in T$ **do**
6: $planWeightArray[T][P] \leftarrow computeWeight(\mathfrak{T}, T, P)$
7: $taskWeightArray[T] \leftarrow taskWeightArray[T] + planWeightArray[T][P]$
8: $TotalWeight \leftarrow TotalWeight + planWeightArray[T][P]$
9: **end for**
10: **end for**
11: **for** each $T \in \mathfrak{T}$ **do**
12: $taskAllocTime \leftarrow getTaskAllocTime(remainTime, TotalWeight,$
13: $taskWeightArray[T])$
14: **for** each $P \in T$ **do**
15: $partnAllocTime \leftarrow getPartnAllocTime(taskAllocTime, T, planWeightArray)$
16: $\mathcal{P} \leftarrow optimizePartn(T, P, partnAllocTime)$
17: **end for**
18: **end for**

### 5.2.4 Algorithm to Deliberate Task-Agent

This algorithm repeatedly searches for the next $S^*_{\mathcal{P}^*_{T,p}}$ to be placed in TCS, while meeting with other requirements such as all agents must be assigned to tasks. In principle, it is similar to algorithm 1 that it searches for the next best task-agent assignment. However, there are two main differences from algorithm 1: $i$) the alter and shrink points, and $ii$) the process of choosing best assignment is more complex than choosing the best candidate coalition.

We define an array $TCS$ of size $|\mathfrak{T}|$ to store the assignment of each task. Firstly, it creates the first TCS by calling $chooseBestAssignment()$ to receive the next $S^*_{\mathcal{P}^*_{T,p}}$. This new assignment will be placed in TCS by calling $assignTask()$, which will locate the next available element for the new assignment. This process repeats until all the tasks are assigned with coalitions of agents, i.e. $unassigned(TCS) = false$. We treat the function $unassigned()$ as a black box which can simply scan the all elements of $TCS$ to locate the first empty element and returns true. Once no empty element is found, it returns false.

The algorithm then calculates for the remaining time before it can improve TCS. While the remaining time is greater than zero, the algorithm calls function $chooseTaskToBeImproved()$. The function will locate the task $taskToBeImprove$ which will be the shrinking point and alter the present agent coalition with the next best one. The algorithm goes into another loop to assign agent coalitions to the remaining unassigned task, starting from $taskToBeImprove$.

### 5.2.5 Algorithm to Choose the Best Assignment

For each $T$, all agents will be ranked by their access costs in descending order, instead of scanning and keeping all the coalitions as in chapter 3. The next available coalition with minimal access cost of $P_p$ can be found by collecting $p$ available agents $\mathcal{L} \in \mathfrak{L}$ from the ranking.

The algorithm begins by initiating $totalRatio$ as the benchmark for the best assignment. It then goes through, from large to small $|T|$, all of the remaining tasks. In each task, it goes through all the $P$s whose size is not greater than $(|RL| - (|RT| - 1)$. For each $P$, the algorithm locates the next best coalition, $S$, by calling function $NextAvail()$, which will start looking for the first $|S|$ agents ranked by access cost to $T$. Note that we start scanning from the largest possible $|S|$ down to 1. The access cost $CA_{T,S}$ of $S$ is then aggregated by calling the function $AccessCost(T, S)$. The total cost ,$C_{\mathcal{P}^*_{T,p},S}$, for $S$ executing $T$ is then computed and followed by the reduction contribution $\bar{A}$ of this assignment $\langle T, S\rangle$. If $\bar{A}$ is smaller than $\bar{A}^*$, the value of $\bar{A}$ is kept as the new benchmark as well as $\langle T, S\rangle$ is kept is the

**Algorithm 16** Search for OTCS by assigning recursively the best task-agent into the existing structure.

**Require:** $remainTime$
1: $initTCS[|\mathfrak{T}|]$
2: $elapsedTime \leftarrow 0$
3: $startTime \leftarrow presentTime$
4: **while** $unassigned(TCS) = true$ **do**
5: $\quad assignment \leftarrow chooseBestAssignment(Tasks)$
6: $\quad assignTask(TCS, assignment)$
7: **end while**
8: $elapsedTime \leftarrow presentTime - startTime$
9: $remainTime \leftarrow remainTime - elapsedTime$
10: **while** $remainTime > 0$ **do**
11: $\quad taskToBeImprove \leftarrow chooseTaskToBeImproved()$
12: $\quad unassignTask(taskToBeImproved)$
13: $\quad$ **while** $unassigned(Tasks) = true$ **do**
14: $\quad\quad assignment \leftarrow chooseBestAssignment(Tasks)$
15: $\quad\quad assignTask(Tasks, assignment)$
16: $\quad$ **end while**
17: $\quad elapsedTime \leftarrow presentTime - startTime$
18: $\quad remainTime \leftarrow remainTime - elapsedTime$
19: **end while**

best assignment. After going through all the remaining tasks and valid partitions, the best assignment is returned.

## 5.3 Example

In the following, we will give a simple example and show how the algorithm works. We have just 2 tasks: $\mathcal{T}_1 = \langle \mathcal{S}_1, \{\mathcal{D}_1, \mathcal{D}_2, \mathcal{D}_3\}\rangle$ and $\mathcal{T}_2 = \langle \mathcal{S}_2, \{\mathcal{D}_4, \mathcal{D}_5, \mathcal{D}_6\}\rangle$. Below are the costs between each pair of source and destination in both tasks:

| | $S_1$ | $D_1$ | $D_2$ | $D_3$ | | $S_2$ | $D_4$ | $D_5$ | $D_6$ | | $S_1$ | | $S_2$ |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| $S_1$ | 0 | 10 | 15 | 20 | $S_2$ | 0 | 8 | 12 | 7 | $L_1$ | 6 | $L_1$ | 8 |
| $D_1$ | 10 | 0 | 12 | 13 | $D_4$ | 8 | 0 | 9 | 11 | $L_2$ | 7 | $L_2$ | 9 |
| $D_2$ | 15 | 12 | 0 | 11 | $D_5$ | 12 | 9 | 0 | 13 | $L_3$ | 10 | $L_3$ | 5 |
| $D_3$ | 20 | 13 | 11 | 0 | $D_6$ | 7 | 11 | 13 | 0 | | | | |

Note that destinations in both tasks are ranked in lexicographical order. In $T_1$, all destinations are co-incidentally ranked by their access costs in ascending order. However, in $T_2$, the correct ranking is $D_6, D_4$ and $D_5$.

**Algorithm 17** Choose best assignment

```
1: Ā* ← MAX_DOUBLE
2: for eachT ∈ RT do
3:     for each P ∈ T and |P| ≤ (|RL| − (|RT| − 1)) do
4:         S ← NextAvail(T, |P|)
5:         CA_{T,S} ← AccessCost(T, S)
6:         C_{P*_{T,p},S} ← CA_{T,S} + CE*_{P_p}
7:         Ā ← C_{P*_{T,p},S}/|S|
8:         if Ā < Ā* then
9:             Ā* ← Ā
10:            assignment ← ⟨T, S⟩
11:        end if
12:    end for
13: end for
14: return assignment
```

### 5.3.1 Combinations of Tasks, Plans, Execution and Access Costs

For the sake of simplicity, we will show all the combinations of tasks, plans, execution and access Costs, and assignments. We will go through detailed execution in next section.

For each task, there are 3 partitions, each of which has just 1 partition instance, i.e. 3, 2+1, 1+1+1. In $\mathcal{T}_1.P_{1.1}$, the only alternative combination is $D_1D_2D_3$. In $\mathcal{T}_1.P_{2.1}$, there are 3 alternatives: $D_1D_2 + D_3$, $D_1D_3 + D_2$ and $D_2D_3 + D_1$. In $\mathcal{T}_1.P_{3.1}$, the only alternative combination is $D_1 + D_2 + D_3$. Similarly, $\mathcal{T}_2$ can be partitioned in the same way. Note that each of these alternative combination is actually a plan $\mathcal{P}$. The optimization algorithm will search the best combination for a given time frame $t$ and return $\mathcal{P}^t$ as the optimal solution at that point in time. Below are the tasks and all alternative plans.

| $T_1$ | | | $T_2$ | | |
|---|---|---|---|---|---|
| $P_1$ | $P_2$ | $P_3$ | $P_1$ | $P_2$ | $P_3$ |
| $D_1D_2D_3$ | $D_1D_2 + D_3$ | $D_1 + D2 + D_3$ | $D_4D_5D_64$ | $D_4D_5 + D_6$ | $D_4 + D_5 + D_6$ |
| | $D_1D_3 + D_2$ | | | $D_4D_6 + D_5$ | |
| | $D_2D_3 + D_1$ | | | $D_5D_6 + D_4$ | |

For the sake of simplicity, we assume in this example that the optimization algorithm is merely a brute force algorithm which searches for the solution by lexicographic order. We assume that the logistics providers have to collect the products from the manufacturing sites (sources) and return the receipt dockets from customers to the manufacturing site to confirm that the customers have received the products. We denote $Sol_t$, where $1 \leq t \leq 6$, as the solution received from the optimization tool at time $t$. The best plan $\mathcal{P}^*_{T,p}$ for each $p$ is the best $Sol_t$ achieved so far. The cost for each plan in each task at time $t$ is shown below:

| | | |
|---|---|---|
| $T_1$ | $\mathcal{P}^*_{T_1,1}$ | $Sol_1 = (S_1D_1D_2D_3S_1) = (10+12+11+20) = 53$ |
| | $\mathcal{P}^*_{T_1,1}$ | $Sol_2 = (S_1D_1D_3D_2S_1) = (10+15+11+15) = 51$ |
| | | $Sol_3 = (S_1D_2D_1D_3S_1) = (15+12+15+20) = 62$ |
| | | $Sol_4 = (S_1D_2D_3D_1S_1) = (15+11+15+10) = 51$ |
| | | $Sol_5 = (S_1D_3D_1D_2S_1) = (20+15+12+15) = 62$ |
| | | $Sol_6 = (S_1D_3D_2D_1S_1) = (20+11+12+10) = 53$ |
| | $\mathcal{P}^*_{T_1,2}$ | $Sol_1 = (S_1D_1D_2S_1) + (S_1D_3S_1) = (10+12+15) + (20+20) = 77$ |
| | $\mathcal{P}^*_{T_1,2}$ | $Sol_2 = (S_1D_1D_3S_1) + (S_1D_2S_1) = (10+15+20) + (15+15) = 75$ |
| | $\mathcal{P}^*_{T_1,2}$ | $Sol_3 = (S_1D_2D_3S_1) + (S_1D_1S_1) = (10+12+15) + (20+20) = 66$ |
| | $\mathcal{P}^*_{T_1,3}$ | $Sol_1 = (S_1D_1S_1) + (S_1D_2S_1) + (S_1D_3S_1) = (10+10) + (15+15) + (20+20) = 80$ |
| $T_2$ | $\mathcal{P}^*_{T_2,1}$ | $Sol_1 = (S_2D_4D_5D_6S_2) = (8+9+13+7) = 37$ |
| | | $Sol_2 = (S_2D_4D_6D_5S_2) = (8+11+13+12) = 43$ |
| | | $Sol_3 = (S_2D_5D_4D_6S_2) = (12+9+11+7) = 39$ |
| | | $Sol_4 = (S_2D_5D_6D_4S_2) = (12+13+11+8) = 44$ |
| | | $Sol_5 = (S_2D_6D_4D_5S_2) = (7+11+9+12) = 39$ |
| | | $Sol_6 = (S_2D_6D_5D_4S_2) = (7+13+9+8) = 37$ |
| | $\mathcal{P}^*_{T_2,2}$ | $Sol_1 = (S_2D_4D_5S_2) + (S_2D_6S_2) = (8+9+12) + (7+7) = 43$ |
| | | $Sol_2 = (S_2D_4D_6S_2) + (S_2D_5S_2) = (8+11+7) + (12+12) = 50$ |
| | | $Sol_3 = (S_2D_5D_5S_2) + (S_2D_4S_2) = (12+13+7) + (8+8) = 48$ |
| | $\mathcal{P}_{T_2,3}$ | $Sol_1 = (S_2D_4S_2) + (S_2D_5S_2) + (S_2D_6S_2) = (8+8) + (12+12) + (7+7) = 54$ |

Note that we show all the possible solutions above. We are yet to through the execute of our algorithm below.

Possible assignment of agent coalitions to tasks as well as the access costs for each task-agent pair are shown below:

| LPs Coalition | $T_1$ | | $T_2$ | |
|---|---|---|---|---|
| | Task-Agent | Access Cost | Task-Agent | Access Cost |
| $L_1$ | $L_1T_1$ | $L_1S_1$=6 | $L_1T_2$ | $L_1S_2$=8 |
| $L_2$ | $L_2T_1$ | $L_2S_1$=7 | $L_2T_2$ | $L_2S_2$=9 |
| $L_3$ | $L_3T_1$ | $L_3S_1$=10 | $L_3T_2$ | $L_3S_2$=5 |
| $L_1L_2$ | $L_1L_2T_1$ | $L_1L_2S_1$=13 | $L_1L_2T_2$ | $L_1L_2S_2$=17 |
| $L_1L_3$ | $L_1L_3T_1$ | $L_1L_3S_1$=16 | $L_1L_3T_2$ | $L_1L_3S_2$=13 |
| $L_2L_3$ | $L_2L_3T_1$ | $L_2L_3S_1$=17 | $L_2L_3T_2$ | $L_2L_3S_2$=14 |
| $L_1L_2L_3$ | $L_1L_2L_3T_1$ | $L_1L_2L_3S_2$=23 | $L_1L_2L_3T_2$ | $L_1L_2L_3S_2$=22 |

Appropriate assignments, where both tasks are assigned with agent coalitions, are shown below. Note that the grand coalition of agents can not be assigned to any task because other tasks will be left unassigned.

| Singleton Coalitions | Aggregated Cost | Mixed Coalitions | Aggregated Cost |
|---|---|---|---|
| $L_1T_1$+$L_2T_2$ | 6+9=15 | $L_1L_2T_1 + L_3T_2$ | 13+5=18 |
| $L_1T_1$+$L_3T_2$ | 6+5=11 | $L_1L_3T_1 + L_2T_2$ | 16+9=25 |
| $L_2T_1$+$L_1T_2$ | 7+8=15 | $L_2L_3T_1 + L_1T_2$ | 17+8=25 |
| $L_2T_1$+$L_3T_2$ | 7+5=12 | $L_1L_2T_2 + L_3T_1$ | 17+10=27 |
| $L_3T_1$+$L_1T_2$ | 10+8=18 | $L_1L_3T_2 + L_2T_1$ | 13+7=20 |
| $L_3T_1$+$L_2T_2$ | 10+9=19 | $L_2L_3T_2 + L_1T_1$ | 14+6=20 |

The aggregated costs, the sum of access cost and execution cost of the above assignments, are shown below.

### 5.3.2 Example of Run

We assume the time allocation strategy is $\langle 50, 50 \rangle$. We refer to the allocated time as a number of steps in order to match with this simple example. The available time is 20 steps. The time allocated for task-plan optimization and task-agent assignment are 10 steps each. We assume this simple optimization takes 1 step for each of the plan. We rank destinations in each task based on their access costs to their respective sources below.

| | $S_1$ | $D_1$ | $D_2$ | $D_3$ | | $S_2$ | $D_6$ | $D_4$ | $D_5$ |
|---|---|---|---|---|---|---|---|---|---|
| $S_1$ | 0 | 10 | 15 | 20 | $S_2$ | 0 | 7 | 8 | 12 |
| $D_1$ | 10 | 0 | 12 | 13 | $D_6$ | 7 | 0 | 11 | 13 |
| $D_2$ | 15 | 12 | 0 | 11 | $D_4$ | 8 | 11 | 0 | 9 |
| $D_3$ | 20 | 13 | 11 | 0 | $D_5$ | 12 | 13 | 9 | 0 |

After the task-plan optimization is finished, the plans for each task are shown below:

| $T_1$ | | | $T_1$ | | |
|---|---|---|---|---|---|
| $\mathcal{P}_1$ | $\mathcal{P}_2$ | $\mathcal{P}_3$ | $\mathcal{P}_1$ | $\mathcal{P}_2$ | $\mathcal{P}_3$ |
| 53 | 77 | 90 | 37 | 43 | 54 |

Note that the ranking of access costs in each task is in the following order: $T_1 : L_1 = 6, L_2 = 7, L_3 = 10; T_2 : L_3 = 5, L_1 = 8, L_2 = 9$. Since the valid plan size is $3-(2-1) = 2$, the execution costs in each task are in the following order: $T_1.P_2 = 77, T_1.P_1 = 51, T_2.P_2 = 43, T_2.P_1 = 37$.

The next step is to repeatedly choose the best assignment to place in $TCS$, which is an array of size 2 in this case. The algorithm examines if there is any task yet to be assigned with agents. The function $unassigned()$ returns true because no tasks have been assigned with agents. The algorithm enters this loop and calls function $chooseBestAssignment()$. In $chooseBestAssignment()$, the algorithm initializes $\bar{A}^* = MAX\_DOUBLE$ as the benchmark for the best assignment. The algorithm then scans through each of the valid $P$s. In this

first round, valid $P$s of $T_1$ are those whose sizes are less than or equal to $(3-(2-1))=2$, i.e. $\{L_1, L_2\}$ and $\{L_1\}$. The total cost and $\bar{A}$ for $\langle T_1, \{L_1, L_2\}\rangle$ are then computed as $77+13=90$ and $\frac{90}{2}=45$ respectively. Since this $\bar{A}=45$ is less than the benchmark $\bar{A}^*$, $\bar{A}^*$ is set to $45$ as the new benchmark and $assignment$ is set to $\langle T_1, \{L_1, L_2\}\rangle$. The algorithm tries with the next assignment $\langle T_1, \{L_1\}\rangle$ whose $\bar{A}=59>\bar{A}^*=45$. Hence, $assignment$ remains the same. The algorithm then tries for the best assignment with $T_2$. The first assignment in $T_2$ is $\langle T_2, \{L_1, L_3\}\rangle$, whose $\bar{A}=\frac{43+13}{2}=28$. Since this $\bar{A}=28$ is less than the benchmark $\bar{A}^*$ = 45, $\bar{A}^*$ is set to $28$ as the new benchmark and $assignment$ is set to $\langle T_2, \{L_1, L_3\}\rangle$. The last assignment to try is $\langle T_2, \{L_3\}\rangle$, whose $\bar{A}=37>\bar{A}*=28$: no changes needed. Hence the function $chooseBestAssignment()$ returns $\langle T_2, \{L_1, L_3\}\rangle$ as the best assignment, which will be placed in the first element of $TCS$ in the function $delibTaskAssign()$. Since there is yet another task to be assigned with agents, the function $delibTaskAssign()$ calls $chooseBestAssignment()$ again. The only pair left is obviously, $\langle T_1, \{L_2\}\rangle$, which is returned to $chooseBestAssignment()$. The algorithm then places $\langle T_1, \{L_2\}\rangle$ at the last element of $TCS$, whose total cost is $(43+13)+(53+7)=116$. Since this is the first $TCS$, $TCS^*$ is then set to $TCS$.

Let's assume there is remaining time for the function $delibTaskAssign()$ and the shrinking point $taskToBeImproved$ return from the function $chooseTaskToBeImproved()$ is 1. The function $unassignTask()$ will remove every element, from the position $taskToBeImproved=1$, from $TCS$. The algorithm calls the function $unasignedTask()$, which returns $T_1$. The algorithm enters the loop and calls the function $chooseBestAssignment()$. Here, the first possible assignment is $\langle T_1, \{L_1, L_2\}\rangle$, which is set to $assignment$ again. The next possible assignment is $\langle T_1, \{L_1\}\rangle$, which is not as good as the present $assignment=\langle T_1, \{L_1, L_2\}\rangle$. The third assignment is $\langle T_2, \{L_1, L_3\}\rangle$ which has been used at this point, so it is ignored in this round. The forth possible assignment is $\langle T_2, \{L_3\}\rangle$, whose $\bar{A}=37<\bar{A}*=45$. Hence, the new $assignment$ is $\langle T_2, \{L_3\}\rangle$, which is returned to the function $delibTaskAssign$. Here, $\langle T_2, \{L_3\}\rangle$ is placed in the first element of $TCS$. The algorithm tries with the next best assignment. Obviously, the only option left is $\langle T_1, \{L_1, L_2\}\rangle$ and is placed in the last element of $TCS$, whose total cost is $(37+5)+(77+13)=132$. Since $TCS^*$ offers lower cost $116$, it remains unchanged.

If the remaining time is less than 0. The present $TCS^*$ will be the bounded-rational solution. If the remaining time is greater than zero, the algorithm attempts to optimize plans in each task for the given period of time. The new $\mathcal{P}$s of each task will be used to re-compute the best assignment later.

## 5.4 Experiments

We are interested in conducting experiments to test the algorithm against various data setting, as we have done in the previous chapter. Here, we want to see how efficiently the algorithm responds to various settings. In general, solutions yielded from algorithm for solving hard problems like TSP or VRP usually improve rapidly in the early stages of the execution and improve slowly in later stages. We want to investigate if the solutions are produced differently, how well our algorithm cope with them.

There are two dimensions in the data generation with respect to elapsed time; the quality of the solutions (agent-task assignments) and the number of solutions generated. Since we are interested in the cost, we refer to the cost reduction as the quality of the solution. We assign to both dimensions the distribution patterns presented above. Hence we have

1. NLRP-NLRP for an environment where the solutions are produced heavily in the early stage of the computation but are produced rarely in later stage, and the costs are reduced rapidly in the early stage but are reduced slowly in later stage of the computation.

2. NLRP-NLDL for an environment where the solutions are produced heavily in the early stage of the computation but are produced rarely in later stage, and the costs are reduced slowly in the early stage of the computation but are reduced rapidly in later stage of the computation.

3. NLDL-NLRP for an environment where the solutions are produced rarely in the early stage of the computation but are produced heavily in later stage, and the costs are reduced rapidly in the early stage of the computation but reduced slowly in later stage of the computation.

Among the aforementioned settings, NLRP-NLRP is the most realistic setting because most of the algorithms for solving hard problems like TSP behave like this. Other setting are merely invented to test the robustness of the algorithm as well as to explore the effectiveness of different time strategies. With respect to the coalition value distribution patterns used in Chapter 3, it is limited to the CCD pattern because it is reflected naturally from the setting, i.e. too small coalitions cannot finish tasks in time (due to time constraints imposed by customers) while too large coalitions incur too much overhead and are not profitable. Other scenarios used in Chapter 3 are not sufficiently realistic for this chapter and we consider them inappropriate for conducting experiments (which we have done quite a number of them already in Chapter 3). CVD, for example, is unlikely to happen because it contradicts the

scenario, i.e. small and large coalitions are profitable in CVD, which is not true as we have just discussed above. We emphasize here again that various data distributions (realistic or unrealistic) in Chapter 3 are invented in order to test the robustness of the principle of CH algorithm which we follow throughout the thesis. Therefore, we decided not to have too many of these unrealistic settings in this chapter.

We allowed 3,600,000,000 milliseconds (1 hour) for the agents to compute OCSs. The total cost at the beginning of each data distribution pattern is around 100000. The time frame will be allocated to both deliberations according to the time allocation strategies. We used strategies already specified earlier in this chapter, i.e. $\langle 50, 40\rangle, \langle 40, 40\rangle, \langle 40, 30\rangle, \langle 30, 30\rangle, \langle 30, 25\rangle, \langle 25, 25\rangle$, which would recursively leave spare time for deliberations in later rounds as $10\%$, $20\%, 30\%, 40\%, 50\%$, respectively. We have 50 agents (logistics providers) for 25 customers, each of which has at least 26 destinations. In the ultimate case, there will 24 customers, whom will be served by one agent, and there will be one customer, whom will be served by 26 agent, each of which will do just one site. The distance between each pair of locations is in the range of 100 to 1000 units.

We show results in terms of reduced cost and percentage of reduced cost. As for the reduced cost, we show results only in the very last part of the experiment because it is where we can see the difference more clearly. We show the full progress of the results for the percentage of reduced cost because we want to show the overall performance of the algorithm. Across all the settings, the results are very consistent, i.e., the trends of the results in each time allocation strategy are similar. The 40-30 time allocation strategy apparently is the best strategy because it yields the highest cost reduction percentage among all strategies in all settings, i.e. 31.3171%, 32.9629% and 33.1223%. However, the progress of the improved solutions in each setting is interesting. In NLRP-NLRP, the first solution of 26.5824% is produced at 1800000ms. The second solution of 30.8369% is produced at 3420000ms. Four more results are produced with small improvement before it reaches the final (seventh) result at 3599998ms. In NLRP-NLDL, the first result of 18.1271% is produced at 1440000ms. The second result is quite a stride, i.e. 28.5098% at 2952000ms. Ten more results are produced with small improvement until the final result at 3599999ms. In NLDL-NLRP, the first result of 16.3757% is produced at 1440000ms. The second result is also a stride, i.e. 27.9425% at 2952000ms. Ten more results are produced with small improvements before the final result is produced at 3599999ms. Other time allocation strategies are not as good as the 40-30. Strategies with larger portion of deliberation time yield quite good results originally but the results hardly or slightly improve after that. For example, 50-50 time allocation strategy yields results after half of the elapsed time and the results are among the poorest ones,

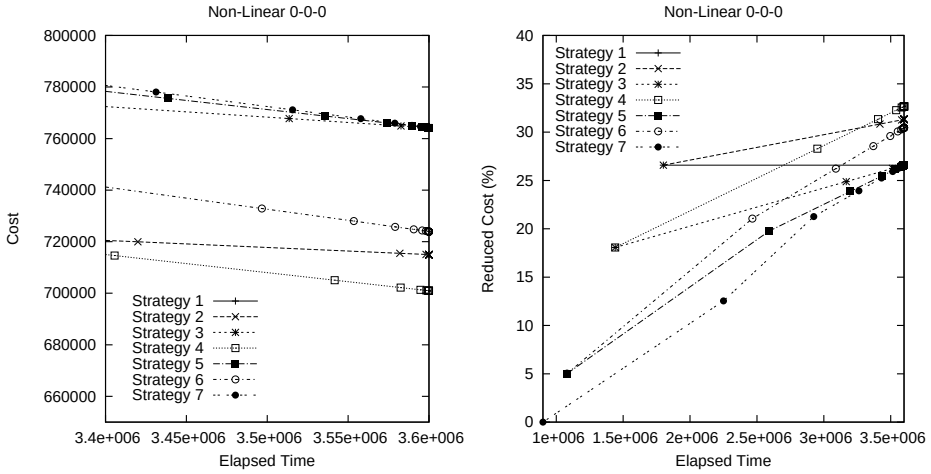


Figure 5.1: **Empirical Results NLRP-NLRP** The graphs show reduced cost in raw figure and percentage achieved from the seven time allocation strategies as per elapsed time.

i.e. solutions are about 25% reduction of original costs. On the other hand, strategies with smaller portion of deliberation time yield results quickly but the final results are relatively poor. Most of them produce results merely around 25% of cost reduction. This tells us that, for solving OCS in non-linear settings which are usually hard problems, we need to allocate time appropriately between deliberation time (for solving the task allocation problems) and solving the OCS itself.

The results from our experiments are shown in Figure 5.1, Figure 5.2 and Figure 5.3 respectively.

## 5.5 Conclusion

The work presented in this chapter has achieved the third objective of this research, namely, to adapt the algorithm presented in Chapter 3 so as: 3. to solve OCS problems in a an NP-hard non-linear environment where coalition values and coalition structure values are not known a priori but must be calculated Having successfully met this third objective, our goal is now to deploy the best-first anytime algorithm to solve OCS problems in which coalitions involve more than two types of stakeholders, such as the supply chain domain

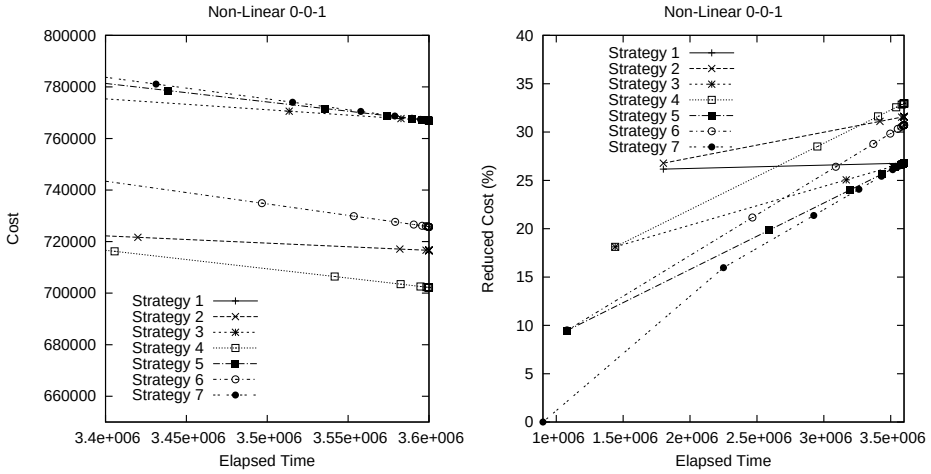


Figure 5.2: **Empirical Results NLRP-NLDL** The graphs show reduced cost in raw figure and percentage achieved from the seven time allocation strategies as per elapsed time.

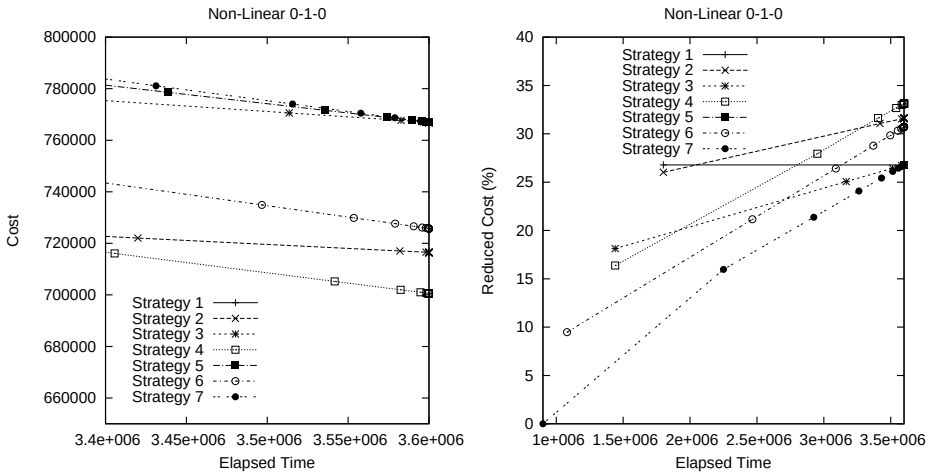


Figure 5.3: **Empirical Results NLDL-NLRP** The graphs show reduced cost in raw figure and percentage achieved from the seven time allocation strategies as per elapsed time.

# Chapter 6

# Coalition Formation in Dynamic Supply Networks

## 6.1 Introduction

The previous 3 chapters have achieved the first three objectives of this research by developing a best-first, anytime algorithm that is an efficient solution for OCS problems in environments where coalition values are known a priori, and adapting that algorithm for use in both linear and non-linear environments, which are much more computationally complex. We have extensively experimented our algorithms against various data settings and taking into account realistic factors. In this Chapter, we will use algorithms we have developed so far to tackle the OCS problem in a complex environment such as supply chain domains, there by achieving the fourth objective of this research. Furthermore, we will investigate the performance of the system from economic perspective, in addition to algorithmic one.

It is widely recognized that the current business environment is characterized by large, complex supply networks that are often global in reach and that are highly adaptive, being frequently re-configured to respond to dynamic business contexts. Collaboration across the supply networks is widely recognized as a key prerequisite for supply networks efficiency. Collaboration can be any form of cooperation/coordination among firms (and these two alternatives will be used interchangeably hereafter). Collaboration in supply networks can take various forms. Suppliers might collaborate to increase selling power or to aggregate capacity. Buyers might collaborate to increase buying power or to reduce logistics costs. LPs might collaborate to increase efficiency in services. From business perspective, it is interesting to know how collaboration among various parties in complex supply networks can bring about a wealthy system. Furthermore, it also interesting to know whether such a wealth can be achieved in such a way that it is distributed *fairly* and *efficiently* to all parties in the system.

As a potential tool to bring about collaboration, coalition formation has been widely studied in supply networks research [14, 49, 31, 45, 54, 75, 108]. A common scenario is

coalition formation among buyers, who form coalitions in order to increase their bargaining power and gain some discounts. Such coalitions can be as simple as buying a single good or can be as complex as combinatorial goods [54]. Each agent attempts to seek the most likely formable coalitions and the most attractive payoffs for coalition members in order to reduce the complication in negotiation [88]. These studies solely address the problem of how the discounts achieved from sellers can be distributed among buyers without addressing the delivery cost of the goods. Another common scenario is in logistic domains, where agents are fully cooperative and commonly seek the maximal utility for the system [77, 75]. Based on these previous works, we have extended our research to tackle even more complex problems in characteristic function game, linear and non-linear settings, where different types of agents can form coalitions.

However, we argue that there are missing links in recent coalition formation research in supply networks. Firstly, delivery cost is a serious issue for forming coalitions. Although it is quite common in practice that sellers can provide free delivery for buyers that are located at neighboring locations, dispersed geographic proximity of buyers in a coalition can incur an additional delivery cost to those members who are not in such a condition. Furthermore, the goods may be sold by themselves, without free delivery service being offered at all. In this case, buyers seriously need to take into account such a cost while negotiating with others or they suffer a total high cost. Secondly, the problem of forming coalitions among sellers in response to a large order is an important issue but has been minimally studied [18]. A seller may not be able to cope with the requirement on its own. Thus it needs to form a coalition to increase it's selling power. This creates a new model of coalition formation between buyers and sellers that brings various issues into perspective. Lastly, the new model has brought a more complex and challenging problem to which LPs need to react in order to increase their service efficiency. They can seek collaboration with others when they want to increase service or economical efficiency. Hence, collaboration across supply networks can be characterized as highly complex coalitions of buyer(s), seller(s) and LP(s).

This chapter offers rich support for dealing with this complexity [99] in two levels: coalitions and coalition structures. For the coalition level, we propose a mechanism for agents to form coalitions. This can be done in two steps. Firstly, buyers form primary coalitions and send their requests to sellers and LPs. Up on receiving the requests, sellers and LPs try to form primary coalitions. Once formed, they can respond to buyers to form secondary coalitions. This makes the model more dynamic and reflects the reality more correctly. For the coalition structure level, we propose a new concept, which take into account fairness and efficiency of the system.

In the followings, we describe coalitions in dynamic supply networks. We explain coalition settings in our work and how agents form primary coalitions, which include negotiation among agent of the same sectors and the deliberation mechanism. After that is the explanation of how agents form secondary coalitions across sectors. We then propose the new concept which will be examined and reported in experiment section.

## 6.2 Coalitions in Dynamic Supply Networks

A *coalition* is a group of agents who unanimously agree to cooperate, e.g., to buy, sell or deliver. Agents are divided according to their roles into buyers, sellers and LPs. The roles of buyer and seller are interchangeable depending on their present activity. An agent can act as a seller in one transaction while becoming a buyer in another transaction. Although it is easy to view LPs as sellers, the role of LPs are quite distinct from other sellers that they sell delivery services. Furthermore, the effect of LPs' role can be captured more easily.

A simple example of coalition formation is a small market game [60], where a seller has a product on sale and two buyers are bidding for it. The seller has a value for the product in mind and is unwilling to accept offers below this price, referred to as the seller's *reserve price*. Each buyer also has a private evaluation of the value of the product and will not pay higher than this value, called the buyer's *reserve price*. Buyers compete with each other by offering a price for the product to the seller. When a buyer offers a price that the others cannot compete with, that buyer then can buy the product from the seller at that price. This can be viewed as their having reached an agreement to cooperate, i.e., they have formed a coalition to trade the product. The value of a coalition is the difference between the buyers and the sellers reserve prices. The payoffs for both agents are the differences between their reserve prices and the selling price. A simple coalition in supply networks is one where a buyer forms a coalition with a seller and an LP. A more complex form of such a coalition involves multiple agents in each sector.

There are multiple distinct drivers for coalition formation. Several small buyers with similar needs coming together to obtain greater bargaining power. Sellers form coalitions with other sellers to aggregate selling power. In some settings, there are legal impediments to certain forms of such coalitions (specifically cartels) in the form of anti-trust laws. However, such coalitions are common for small sellers, such as in agricultural cooperatives (e.g. for micro-producers of dairy products). Sellers sometimes form coalitions to aggregate/augment capabilities. LPs may want to form coalitions to aggregate their service capacities or efficiency. We note that we have only listed the basic drivers. Most real-life supply network

coalitions tend to have more than one motivating factor driving their formation. Multiple factors play a role in coalition formation decisions. In the current work, we focus on quantitative factors, e.g., price, delivery cost and lead time. Most decisions typically involve trade-offs among several attributes. One may be willing to pay more for a shorter lead time and vice versa. One may, in some settings, be willing to accept longer lead times for the purpose of getting higher quality. Several other instances of such trade-off are common, but we do not list them all here.

Although negotiation can be either bilateral or multilateral, we can in most instances reduce negotiation to the bilateral case without loss of generality. An agent that stands to benefit the most from forming a coalition usually acts as the coalition *leader*. Firstly, a coalition leader negotiates bilaterally with multiple agents of its own sector to establish a sectoral coalition. An agent might find itself forming a singleton primary coalition because it cannot find suitable coalition members nor there is need to do so. We call this type of coalitions *primary coalition*. Firstly, buyers form primary coalitions and the coalition leaders send request to sellers and LP(s). Interested sellers and LP(s) then form primary coalitions among themselves. The coalition leaders then negotiate across sectors to establish a cross-sectoral coalition involving a coalition of buyer(s), seller(s) and LP(s). We call the type of coalitions *secondary coalition*. This secondary coalition is where the real trade is finalized. The buying leader collects money from its members and pays to the selling leader and LP leader, who in turn distribute among their members. Then the coalitions will break. Members can form coalitions later on should they need to do so. We note that these negotiations must occur within stringent time constraints.

## 6.3 Coalition Formation

Supply networks are very complex systems. In order to study coalition formation in such systems, we create a small system composed of three kinds of agents: buyers, sellers, and logistics providers. Each of these agents is merely a small economic entity of a small economy. A buyer can be a typical consumer. A seller can be a small producer, whose number of member can be just one. A logistics provider is, as in the previous chapter, merely a small company, whose only resource is a truck. However, the model can be scalable. Our main aim is to use the model as a testbed to study coalition formation among buyers, sellers and logistics providers by applying techniques presented in previous chapters.

### 6.3.1 Setting

Let $\mathfrak{B}$, $\mathfrak{S}$ and $\mathfrak{L}$ be set of agents who act as buyers, sellers and LPs respectively. Buyers order products, $\mathfrak{F}$, from sellers. LPs distribute the products over a transportation network, $\mathcal{G} = \{\mathcal{V}, \mathcal{E}\}$, where $\mathcal{V}$ is a set of vertices and $\mathcal{E}$ is a set of edges, each of which connects a pair of vertices, for them. The location of each buyer $b \in \mathfrak{B}$, seller $s \in \mathfrak{S}$, and LP $l \in \mathfrak{L}$ are specified by function $Loc_b : \mathcal{B} \rightarrow \mathcal{V}$, $Loc_s : \mathcal{S} \rightarrow \mathcal{V}$, and $Loc_l : \mathcal{L} \rightarrow \mathcal{V}$ respectively. These three functions are accessible to all agents. Agents have individual goals and constraints. A goal is represented by the product that an agent may seek to buy or sell. Constraints are the time/deadline and the price at which they wish to conclude the transaction. The amounts of specific products required by specific buyers are given by $Dem : \mathfrak{B} \times \mathfrak{F} \rightarrow \mathbb{N}^+$. The budgets of buyers and the due times by which the buyers want the products are determined by the function $Bud : \mathcal{B} \rightarrow \mathbb{R}^+$ and $Due : \mathfrak{B} \times \mathfrak{F} \rightarrow \mathbb{N}^+$, respectively. Here the output of the time function denotes the elapsed times from a commonly agreed upon origin). The amounts of certain products that sellers can produce are given by $Prod : \mathfrak{F} \times \mathfrak{S} \rightarrow \mathbb{N}^+$. The times which sellers require to produce products are determined by $ProdTime : \mathfrak{F} \times \mathfrak{S} \rightarrow \mathbb{N}^+$. The costs of producing products are governed by $ProdCost : \mathfrak{F} \times \mathfrak{S} \rightarrow \mathbb{R}^+$. The load capacities of LPs are determined by $Load : \mathfrak{L} \rightarrow \mathbb{N}^+$. For the sake of simplicity, we assume the load capacity is measured by weight. The costs of delivery between locations are determined by $DelCost : \mathcal{V} \times \mathcal{V} \rightarrow \mathbb{R}^+$. The time of traveling between locations are determined by $TraTime : \mathcal{V} \times \mathcal{V} \rightarrow \mathbb{N}^+$.

We define a requirement as a tuple $Req = \langle f, q, t \rangle$, where $f \in \mathfrak{F}$ is a product an agent wants to buy, $q \in \mathbb{N}^+$ is the quantity of the product which is needed, and $t \in \mathbb{N}^+$ is the (elapsed) time by which the buyer wants the product be delivered. We define a pickup requirement as a tuple $Pick = \langle f, q, v, t \rangle$, where $f \in \mathfrak{F}$ is the product to be picked up, $q \in \mathbb{N}^+$ is the quantity of the respective product, $v \in \mathcal{V}$ is the location from which the product is to be picked up, and $t \in \mathbb{N}^+$ is the elapsed time by which the product need to be picked up. Corresponding to this, we define a delivery requirement as a tuple $Deli = \langle f', q', v', t' \rangle$, where $f' \in \mathfrak{F}$ is product to be delivered, $q' \in \mathbb{N}^+$ is the quantity of the respective product, $v' \in \mathcal{V}$ is the location from which the product is to be picked up, and $tt \in \mathbb{N}^+$ is the elapsed time by which the product need to be picked up. We define a job descripte as a tuple $Job = \langle \langle f, q, v, t \rangle, \langle f', q', v', t' \rangle \rangle$, composed of $Pick$ and $Deli$

### 6.3.2 Forming Primary Coalitions

In our setting, buyers need to form coalitions in order to gain the benefit from bulk buys. Sellers also form coalitions to increase their producing capabilities.

#### Buyer to Buyer Negotiation

A buyer, who wants to buy products and foresees an opportunity to gain some discounts from buying a large quantity, acts as a coalition leader by negotiating with other buyers. We shall call it the leading buyer. The following is the protocol for forming a coalition of buyers.

1. Every buyer $b$ sends a message $(b, \{\langle f, q, t\rangle\}, v, p)$, where $b$ is the identifier of the agent, $\{\langle f, q, t\rangle\}$ is a collection of requirements vector, $v \in \mathcal{V}$ is the location where the product will be delivered, $p$ is the price, and $t$ is the due time it wants the product to be delivered.

2. Based on its deliberation, any buyer can act as a leading buyer, $b_0$, and initiates the process of forming a primary coalition of buyers. (See section Deliberation for Joining Primary Coalition for more details).

3. Leading buyer $b_0$ sends a message $(msg, b_0, \langle f, q, t\rangle, v, p, x)$, where $msg$ is the message id, $p$ is the price, and $x$ is the expiry time to potential buyers in its current best potential coalition.

4. Any buyer, who is satisfied with the offer, sends a message $(msg, b, ACK, x)$ back to $b_0$ stating that it will wait for a confirmation until time $x$.

5. Leading buyer $b_0$ sends a message $(msg, b_0, ACK)$ to inform all interested buyers that the new primary coalition has now been formed.

6. Leading buyer $b_0$ sends a message $B2S(msg, b_0, \langle f, q, t\rangle, p)$ to sellers, where $\langle f, q, t\rangle$ is the order details compiled from its members, and $t$ is the available time.

7. At the same time, the leading buyer sends message $B2L(id, b_0, \langle f, q, v, t\rangle, \langle f', q', v', t'\rangle, p)$, where $\langle f, q, v, t\rangle$ and $\langle f', q', v', t'\rangle$ specifies job description for picking up from sellers and delivering the products to buyers, to LPs.

The leading buyer may find itself being a singleton coalition, i.e., it can not agree with other buyers or there is no need to do so. Once the coalition is formed, $b_0$s can negotiate with sellers and LPs to form secondary coalitions by sending messages to them.

**Seller to Seller Negotiation**

Upon receiving the message from a leading buyer, interested sellers react. If the order is within it's capability, it may form a singleton coalition on its own. If the order is beyond its capability, it can act as a coalition leader and try to form a coalition of sellers in order to increase it's selling power. We shall call it a leading seller. The following is the protocol for forming coalitions among sellers.

1. Interested sellers send a message $(msg, s, \langle f, q, \mathfrak{p}, \mathfrak{r}, v\rangle)$, where $msg$ is the message id sent by $b_0$, $\mathfrak{p}$ is specified by function $prod$, $\mathfrak{r}$ is specified by function $Dem$, and $\langle f, q, \mathfrak{p}, \mathfrak{r}, v\rangle$ is the list of capability vectors, to all other sellers.

2. Any seller can act as a leading seller, $s_0$, and try to form a primary coalition with other sellers. It selects a number of sellers, who might help increase its selling power (See section Deliberation for Joining Primary Coalition for more details).

3. Leading seller $s_0$ sends a message $(msg, s_0, \langle s, f, v\rangle, x)$, where $\langle s, f, v\rangle$ is a collection of preferred capability vectors, to selected sellers.

4. Any seller, who is satisfied with the offer, replies with the message $(msg, s, ACK, x)$ and waits for the confirmation by time $x$.

5. Agent $s_0$ confirms that a primary coalition is now formed with the message $(msg, s_0, ACK)$.

The leading seller may find itself being a singleton coalition because it cannot agree with other sellers. It can begin negotiation with the leading buyer to form secondary coalition.

**LP to LP negotiation**

Upon receiving the message from leading buyer $b_0$, interested LPs react. Any interested LP may try to form a primary coalition with other LPs. We shall call such an LP the leading LP. Following is the protocol for forming coalitions among logistics providers.

1. Every interested LP sends a message $(msg, l, \mathfrak{l}, v)$, where $msg$ is the message id sent by $b_0$, $\mathfrak{l}$ is the load capacity specified by function $\mathcal{L}$, to all other LPs.

2. Any interested LP can act leading LP, $l_0$, and try to form a primary coalition with other LPs. It selects a number of LPs, who might help increase its capacity and service efficiency. (See section Deliberation for Joining Primary Coalition for more details).

3. Leading LP $l_0$ sends a message $(msg, l_0, v, x)$, to selected LPs to ask if they would like to form a primary coalition.

4. Any LP, who is satisfied with the offer, replies with the message $(msg, l, ACK, x)$ and waits for the confirmation by time $x$.

5. Agent $l_0$ confirms to its primary coalition members with the message $(msg, l_0, ACK)$.

The leading LP may find itself being a singleton coalition. It then can negotiate with the leading buyer $b_0$ on behalf of it's members to form a secondary coalition.

**Deliberation for Joining a Primary Coalition**

Agents have a common view when forming primary coalitions: Quality of Coalition (QoC). QoC indicates the likelihood to achieve the goal of the coalition. The common goal of every agent is to maximize its payoff. Buyers need members who can help increase their bargaining power. Sellers need members who can help increase selling power, which in turn helps maximize their payoffs. LPs needs member who can help increase service efficiency, which in turn helps maximize their payoffs. Based on its status, an agent can create an ideal coalition, the one with the highest QoC. The agent creates a template, which is an ideal coalition member of the ideal coalition. Each agent measures the suitability of another agent to the the ideal coalition in terms of the distance between that agent and this template. It uses the distance to rank other agents in a table, which will be used to help select appropriate agents for forming high QoC coalitions. Let $A = \{a_1, a_2, \ldots, a_m\}$ be a set of attributes. Let $B = \{L_1, U_1, L_2, U_2, \ldots, L_m, U_m\}$ be a set of lower bounds and upper bounds of quantitative attributes $a_1, a_2, \ldots, a_m$. A template is a vector $t = \langle q_1, q_2, \ldots, q_m, w_1, w_2 \ldots, w_m \rangle$, where $q_i$ is a quantitative attribute $q_j$, and $w_k$ is the weight for each $a_k$ of $m$ attributes. The agent $a_0$, who wants to form a primary coalition, uses the attributes passed over in the first step of the negotiation to measure the distance between other agents and its templates. The closer distance signifies the more suitability for the coalition of the agents. The distance can be considered as the sum of the difference of each pair of corresponding attributes in the template and the agents' attributes. For each attribute, the distance should be 0 if the attribute's value are equal. Otherwise the distance is the multiplication of the difference between the values of corresponding attributes, and the weight for that attribute. Let $y_j$ be the value of attribute $j$ of the template, the distance between an agent and a template can be derived as follows:

$$d = \sum_{j=1}^{m} \mathit{Diff}(q_j, y_j) w_m$$

where

$$\mathit{Diff}(q_j, y_j) = \begin{cases} \infty & \text{if } y_j \rangle U_j \text{ or } y_j \langle U_j \\ \frac{q_j - y_j}{q_j - L_j} & \text{if } q_j \rangle y_j \text{ and } y_j \text{ is } \geq L_j \\ \frac{y_j - q_j}{U_j - qj} & \text{if } q_j \langle y_j \text{ and } y_j \text{ is } \leq U_j \\ 0 & \text{if } q_j = y_j \end{cases}$$

Let us consider an example of how this mechanism works. Suppose there are four attributes, $A = \{f, q, p, t\}$, to describe a quality of primary coalition of buyers. Let $B = \{1, 10, 10, 20, 10, 30, 0, 100\}$ be the set of upper and lower bounds for $f, q, p$ and $t$, respectively. Let $T = \langle 1, 15, 20, 30, \infty, 10, 1, 5 \rangle$ be a template of buyer $b_0$, who is considering forming a primary coalition. There are two buyers, $b_1$ and $b_2$ being considered whether they should be invited to join the coalition. Their attributes, achieved from the messages, are $\{1, 13, 18, 28\}$ and $\{1, 17, 23, 34\}$, respectively. The distance between agent $b_1$ and template $T$ is

$$\begin{aligned} d_1 &= (0 * \infty) + (2/5) * 10 + (2/10) * 1 + (2/300) * 5 \\ &= 0 + 4 + 0.2 + 0.33 \\ &= 4.53 \end{aligned}$$

The distance between agent $b_2$ and template $T$ is

$$\begin{aligned} d2 &= (0 * \infty) + (2/5) * 10 + (3/10) * 1 + (4/70) * 5 \\ &= 0 + 4 + 0.3 + 0.57 \\ &= 4.87 \end{aligned}$$

Since 4.53 is less than 4.87, agent $b_1$ is obviously closer to the template than $b_2$. Hence, agent $b_1$ sits higher in the ranking table.

Each agent uses its own ranking table to compute potential primary coalitions, those it might try to form, and keeps them in a list. The agent sets up a period of time for deliberation, i.e., generating coalitions and adding them to the list. It computes, for each potential coalition, the aggregated distance, which is the sum of of all members' distance to the template. It then ranks those coalitions by the coalitions' aggregated distance in ascending order. In each potential coalition, the agent computes potential payoffs for the members. The agent considers each potential coalition one by one from the top of the list. For the best potential coalition being considered, the agent decides to be a leading agent if it's potential payoff is the greatest. It then follows the primary protocol to negotiate with others. If the agent's potential payoff is not the greatest, it will wait for some time. Upon receiving an invitation message, it computes its potential payoff and compares it to the potential payoff in the second best potential coalition of its own list. If the the inviting potential payoff is not lower,

the agent accepts the offer, or rejects it otherwise. In the case the coalition cannot be formed. The next potential coalition will be considered. More potential coalitions will be computed as needed.

### 6.3.3 Secondary Coalitions

A secondary coalition is composed of three primary coalitions: buyer, seller and LP primary coalitions. The negotiation occurs among coalition leaders and within primary coalition members as an ongoing process. They try to agree on their payoffs and tasks (to supply product for sellers and to deliver products for LPs). Agents who are not satisfied with payoffs being offered may decide to deviate from their present primary coalitions and try to form new ones.

#### Buyers to Sellers and LPs

Firstly, leading sellers and leading LPs try to agree on an offer for the corresponding buyer. The negotiation involves their payoffs and tasks. The protocol for forming such coalitions is as follows:

1. Each member seller evaluates its present capability and sends it's $s_0$ a message $S2S(msg, b, \langle f, q, t\rangle, p)$ bidding for the supply task.

2. Leading seller $s_0$ may have to negotiate on product, quantity, time and price with member sellers with message $S2S(id, b, \langle f, q, t\rangle, p)$ until every member seller is satisfied and the total quantity, available time and price is satisfied with the request.

3. Leading seller $s_0$ accumulates the bids from its member sellers and creates an offer. It then sends message $S2B(msg, s_0, \langle f, q, v, t\rangle, p)$ back to $b_0$.

4. Similarly, member LP $l$ bids for the task by sending message $L2L(id, \langle f, q, v, t\rangle, \langle f', q', v', t'\rangle, p)$ to its leading LP $l_0$.

5. Leading LP $l_0$ may have to negotiate over quantity, pickup location, delivery location, pickup time, delivery time, and price with its members.

6. The leading LP $l_0$ accumulate all the bid from its members in order to create an offer for the delivery job.

7. It sends a message, i.e., proposal, $(id, s_0, \langle f, q, v, t\rangle, p)$ back to $b_0$.

8. The leading buyer $b_0$ accumulates the proposals from the leading sellers and leading LPs and finds the best combination of a selling and a LP proposal. It then creates a proposal to its members and sends message $(id, s_0, \langle f, q, v, t\rangle, p)$ to the members.

9. Members who are satisfied with the proposal send a message $(id, s, ACK)$ to $l_0$.

10. Members who are not satisfied can negotiate over quantity, time and price by sending message $(id, s, \langle f, q, v, t\rangle, p)$ back to the leading agent.

11. Once all member buyers are satisfied, the leading buyer then sends message $(id, b_0, ACK)$ to the selected leading seller and leading LP.

12. The leading buyer, seller, and LP send message $(id, a_0, CONFIRM)$ to their coalition members confirming that the secondary coalition has been formed.

LPs are to pickup the product from sellers and deliver to buyers. Sellers and LPs get paid when the product is delivered. The negotiation within and among primary coalitions can keep going until agents are satisfied or the time is over.

### 6.3.4 Decision Mechanism

In our setting, each agent has their individual reserve price. For a buyer, the reserve price $r_b$ is the maximum price he is willing to pay for acquiring a bunch of products, i.e., including prices of products and costs of delivery. For a seller, the reserve price $r_s$ is the cost $\mathfrak{c}_p$ of producing products and the minimum profit it expects. For an LP, the reserve price $r_l$ would be the sum of the estimated cost of operation and the minimum profit it expects. Let $\mathsf{B} \subseteq B, \mathsf{S} \subseteq S$, and $\mathsf{L} \subseteq L$ be a set of buyers in a primary coalition, a set of sellers in a primary coalition, and a set of LPs in a primary coalition, respectively.

For a secondary coalition $\mathsf{C} = \{\mathsf{B}, \mathsf{S}, \mathsf{L}\}$, the reserve price of $\mathsf{B}$ is

$$r_{\mathsf{B}} = \sum_{b \in \mathsf{B}} r_b,$$

the reserve price of $\mathsf{S}$ is

$$r_{\mathsf{S}} = \sum_{s \in \mathsf{S}} r_s,$$

and the reserve price of $\mathsf{L}$ is

$$r_{\mathsf{L}} = \sum_{l \in \mathsf{L}} r_l.$$

The coalition value of a secondary coalition $\mathrm{C}$ is

$$V_{\mathsf{C}} = r_{\mathsf{B}} + r_{\mathsf{S}} + r_{\mathsf{L}},$$

which is to be distributed among agents. Let $\mathsf{P_S}$ be the price that sellers charge buyers. Let $\mathsf{P_L}$ be the price that LPs charge buyers. The payoff for buyers in $\mathsf{B}$ is

$$\mathrm{U_B} = r_{\mathsf{B}} - \mathsf{P_S} - \mathsf{P_L}.$$

The payoff for sellers in $\mathsf{S}$ is

$$\mathrm{U_S} = \mathrm{P_S} - r_{\mathsf{S}}.$$

The payoff for LPs in $\mathsf{L}$ is

$$\mathrm{U_L} = \mathrm{P_L} - r_{\mathsf{L}}.$$

From now on, we shall refer to the secondary coalition as a coalition because it is the true coalition in a supply chain domain.

The price each agent uses to negotiate may be higher than their reserve prices. Knowing the bidding price of every agent in its primary coalition, the leading agent can use a fair division of their coalition payoff by using the Shapley value [42] concept. Agents in the primary coalition may find that the shares offered by the leading agent are below than their own reserve price. Hence they may have to deviate from their present primary coalitions. Although Shapley value does exist in every coalition structure, it does not guarantee system's stability nor efficiency. In addition to Shapley value, there are several solution concepts in coalition formation theory [42], e.g., core kernel, etc. These solutions are based the assumption that agents are self-interested. Kernel, for example, guarantees stability to the system and individual satisfaction to agents. However, it is computationally complex and does not guarantee system's efficiency. The Core guarantees stability to the system and efficiency in both the system and individual level.

Here, we propose to use a compromising solution. We invent a novel concept in order for agents to settle down their negotiation based on two major concepts *fair* and *efficient*:

1. **Fair** To ensure that each agent has a fair share of such a solution. By saying "fair share", we mean the difference of each pair of agents' payoffs across the whole system is minimal.

2. **Efficient** To maximize the social welfare, i.e. the utility of the whole system should be maximized.

As a very first step of this novel fair and efficient concept, we adopt the Shapley value (for the fairness) and our OCS algorithms (for efficiency) we have developed throughout this thesis. Shapley value guarantees fairness in a given coalition. We extend it to another level

by taking into account the fairness across the CS. Given a coalition $S$ with coalition value $V_S$, we define the *average payoff*

$$AV_S = \frac{\sum V_{a_i}}{n},$$

where $a_i \in S$, and $V_{a_i}$ is the agent's payoffs achieved by applying the Shapley value. We define *average payoff difference*

$$DV_S = \sum |AV_S - V_{a_i}|.$$

This $DV_S$ reflects the fairness in each coalition that the less value it has, the less difference among agents' payoff. Further, it tells us how much this coalition is valued in terms of fairness and efficiency. This brings us a new coalition value, which we name *fairly efficient* value,

$$FEV_S = AV_S - DV_S.$$

With respect to a CS, we then have the CS's fairly efficient value,

$$FEV(CS) = \sum_{S \in CS} FEV_S.$$

We are interested to find the optimal fairly efficient coalition structure,

$$FECS^* = argmax_{CS \in L} FEV(CS).$$

We follow the fairness principle provided by Shapley because it is the only solution concept in cooperative game that does exist in every CS.

### 6.3.5 Algorithm

So far, we have developed a couple of solid algorithms for optimal coalition structures. In Chapter 3, we have shown that with the concept of choosing *best* coalition as the new coalition to place in coalition structure, we can reach optimality very quickly in characteristic function game. However, we are merely limited to 26 agents. In Chapter 4, we have shown that for larger number of agents, i.e. 50 agents, and in more pragmatic settings such as a linear production domain, we can choose only a small number of agents (by applying a heuristic approach) and can reach (near) optimal coalition structures, quickly. In Chapter 5, we have shown that for very hard problem, such as non-linear logistics domain, which is usually intractable for even a small number of agents, we have appropriate time allocation strategy to search for optimal coalition structure.

In this Chapter where we aim at optimal coalition structure in a supply chain domain, we need not an absolutely new algorithm but we can simply use the set of algorithms we

have developed so far. This is because these algorithms cover all the needs to solve a hard problem in relatively larger scale and have proved themselves with empirical results. We will follow the principle (proposed in Chapter 3) of choosing a relatively small number of *best* coalitions from available candidates and placing it at each level of CS. We will follow the heuristics (proposed in Chapter 4) to locates appropriate coalition members. We will follow the time allocation strategy (proposed in Chapter 5) to solve this hard problem of searching for optimal coalition structure in constrained time environment. We have also proposed in this Chapter the new type of coalition in supply chain domain as well as the new principle for considering optimal coalition structure.

Therefore, the only thing that is needed (as the condition for forming coalition has changed: a coalition is composed of buyers, sellers, and logistics providers) is the condition of *disjointness* of agents in a coalition structure. That is we need to satisfy this condition in all types of agents. To cope with this, we add three variables to represent remaining buyers, sellers and logistics providers. When the new coalition is placed in CS, respective buyers, sellers and logistics providers will be removed from the variables. Similarly, respective agents will be added to the variables when a coalition is removed from CS. The criteria for choosing the next *best* coalition is also changed according to the new concept we proposed in previous section.

However, for the sake of completeness, we show the algorithms below Algorithm 18-22). Note that we denote remaining buyers, sellers and logistics providers by $R.B, R.S, R.L$, respectively (and vice versa for $CS[l]$).

## 6.4 Experiments

We are interested in comparing the typical OCS and the new concept we proposed that how much different could it be between the system's utility, the agents' payoffs, and the elapsed time from convergence to termination.

We experimented for 100 agents which composed of 3 sets of agents. We designed 7 settings, each of which specifies the number of buyers, sellers, and logistic providers, as follow: 60-30-10, 50-40-10, 40-50-10, 30-60-10, 50-30-20, 40-40-20, and 30-50-20. These settings will be referred to as setting 1, 2, 3, 4, 5, 6 and 7, respectively. We used these figures based on the experiments in previous chapters that we do not allow too many or too few logistics providers that will not be profitable for all parties. For each of these settings, we ran experiment 100 times. In each time, we search for the solution both by V(CS) and FEV(CS). The experiments were conducted in AMD Turion 64 machine with 2GB RAM. We recorded

**Algorithm 18 Main** Construct coalition structures by choosing best coalition (of buyers, sellers and logistics providers) and place it into the present $\mathcal{CS}[l]$

```
1:  l ← 1                                              ▷ the present layer l is set to 1
2:  S* ← chooseBestS(l)                                ▷ the best coalition S* is placed into CS[l]
3:  while S* ≠ ∅ do                                    ▷ while S* exists
4:      CS[l] ← S*                                     ▷ place S* in CS[l]
5:      R.B ← R.B \ S*
6:      R.S ← R.S \ S*
7:      R.L ← R.L \ S*                                 ▷ update remaining B, S, and L with S*
8:      S* ← ∅                                         ▷ reset S* to ∅
9:      if R.B = R.S = R.L = ∅ then                    ▷ no agents left outside CS
10:         print "new CS generated: "+CS;             ▷ output new CS
11:     end if
12:     S* ← Extend()                                  ▷ attempt to extend CS
13:     if S* != null then                             ▷ CS can be extended if S* is found
14:         l ← l + 1;                                 ▷ CS is to be extended
15:     else
16:         S ← Alter()                                ▷ try for altering when cannot extend
17:         if S* = ∅ then                             ▷ cannot alter, nothing to substitute
18:             S* ← Shrink()                          ▷ then try to shrink
19:         end if
20:     end if
21: end while
```

**Algorithm 19** ChooseBestS Function

```
1:  function CHOOSEBESTS(l)
2:      bestS ← ∅
3:      ā* ← 0
4:      for c = 1 to |R| do                            ▷ for each valid cardinality
5:          if B[l][c] > 0 then                        ▷ if there is a candidate coalition
6:              ā ← V_{C[c][B[l][c]]}/c                ▷ compute the candidate's ā
7:              if ā > ā* then                         ▷ if the new ā is better than ā*
8:                  bestS ← C[c][B[l][c]]              ▷ set the new best coalition
9:                  ā* ← ā                             ▷ set ā* to the new value
10:             end if
11:         end if
12:     end for
13:     return bestS
14: end function
```

**Algorithm 20** NextS Function

```
1: function NEXTS(c,p)
2:     for j = p to C[c].length do        ▷ starting from p towards the last element of C[c]
3:         if C[c][j].B ⊆ R.B and
4: C[c][j].S ⊆ R.S and
5: C[c][j].L ⊆ R.L then                   ▷ if the coalition at p is valid
6:             return j                   ▷ return its position
7:         end if
8:     end for
9:     return 0
10: end function
```

**Algorithm 21** Extend Function

```
1: function EXTEND
2:     if l < n then
3:         for c = 1 to |R| do
4:             p ← B[l][c]              ▷ set the beginning position for searching for candidate
5:             if p > 0 then            ▷ only cardinalities that have coalitions left
6:                 if c = |CS[l]| then  ▷ for candidate of cardinality of CS[l]
7:                     p ← p + 1        ▷ begin the search at the next position
8:                 end if
9:                 B[c][l + 1] ← NextS(c, p) ▷ search for the next candidate of cardinality c
10:            end if
11:        end for
12:        return ChooseBestS(l + 1)    ▷ acquire the best coalition and return it
13:    end if
14:    return ∅
15: end function
```

**Algorithm 22** Alter Function

```
1: function ALTER
2:     R.B ← R.B ∪ CS[l].B
3:     R.S ← R.S ∪ CS[l].S
4:     R.L ← R.L ∪ CS[l].L               ▷ return the last coalition of CS to R
5:     p ← B[|(CS[l])|][l] + 1 ▷ start to search for the another candidate at the next position
6:     B[|CS[l]|][l] ← NextS(|CS[l]|, p) ▷ retrieve the alternative candidate
7:     CS[l] ← ∅                         ▷ reset CS[l]
8:     return ChooseBestS(l)             ▷ acquire the best coalition and return it
9: end function
```

**Algorithm 23** Shrink Function

```
1: function SHRINK
2:     if l > 1 then
3:         l ← l − 1                                   ▷ shrink by decreasing value of l by 1
4:         R.B ← R.B ∪ CS[l].B
5:         R.S ← R.S ∪ CS[l].S
6:         R.L ← R.L ∪ CS[l].L                          ▷ return members to R
7:         p ← B[|CS[l]|][l] + 1   ▷ set the starting point to search for alternative candidate
8:         B[|CS[l]|][l] ← NextS(|CS[l]|, p)            ▷ search for the alternative candidate
9:         CS[l] ← ∅;                                   ▷ reset CS[l]
10:        return ChooseNextS(l);                       ▷ acquire the best coalition and return it
11:    end if
12:    return ∅
13: end function
```

the convergence and termination times as well as the V(CS) and FEV(CS) of each run. The average values of recorded data for each setting are then crafted out.

We show the results in 6.1 with two graphs. The first graph show convergence against termination time for each of the settings achieved from searching by V(CS) and FEV(CS), respectively. This is to show the performance of the algorithm to generate OCS as we have done in previous chapters. The second one shows V(CS) against FEV(CS) for each of the settings, achieved from searching for the solution by V(CS) and FEV(CS), respectively. This is to show the efficiency of the algorithm from economic perspective, i.e. to see how agents can be affected from V(CS) and FEV(CS).

From the first graph, the results show that termination times of V(CS)-oriented and FEV(CS)-oriented runs are not much different. This tells us that differences in numbers of buyers, sellers and logistics providers do not affect much on termination times. However, there are significant differences in convergence times (which is more important than termination time as we have discussed in previous chapters). In setting 1, convergence time of V(CS)-oriented run is closest to termination time but is furthest from termination time in setting 2 and 3. It gets closer to termination time (but still not the closest) and is quite consistent in setting 4, 5, 6 and 7. The implication from V(CS)-oriented runs is that the convergence time is shorter when the number of buyers and sellers are not much different, i.e. 50 buyers versus 40 sellers and vice versa for 10 logistics providers. When the number of logistics providers increases to 20, the convergence times are also not much different for 50 buyers versus 30 sellers and vice versa, and 40 buyers versus 40 buyers.

However, the convergence times of FEV(CS)-oriented runs are very interesting. For 10 logistics providers, the convergence times of FEV(CS)-oriented runs are opposite to those

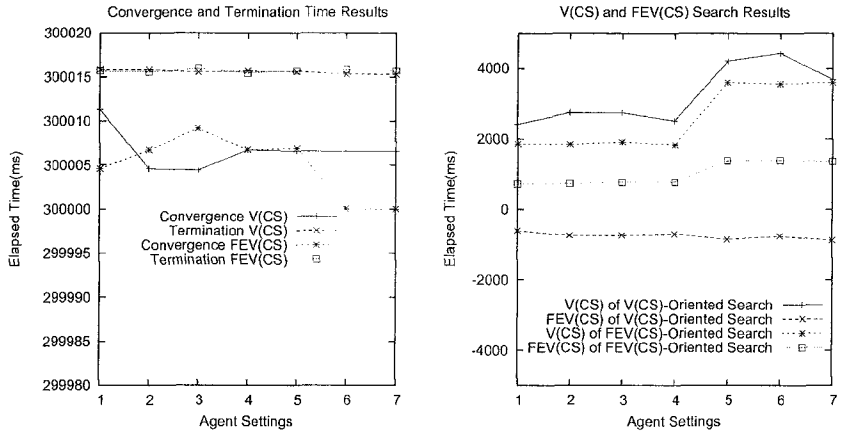


Figure 6.1: **Empirical Results of V(CS) against FEV(CS)** The graphs show convergence versus termination time, and V(CS) versus FEV(CS) of the V(CS)-Oriented versus FEV(CS)-Oriented search.

of V(CS)-oriented runs, i.e. the convergence times of 60 buyers versus 30 sellers and vice versa are shorter than that of 50 buyers versus 40 sellers and vice versa. In other words, for a small number of logistics providers, settings with greater difference in number of buyers and sellers can reach convergence earlier. For larger number of logistics providers, i.e. 20, convergence times decrease when number of buyers decrease, i.e. dropping from 50, 40, to 30. Note that FEV(CS)-oriented runs are driven by the value of FEV(CS) which are less than V(CS). This causes the search to converge quicker (based on the principle of *best* coalition used throughout this thesis.

From the second graph, the results show that V(CS)-oriented runs yield higher-valued solutions. The V(CS) values are quite consistent in setting 1, 2, 3 and 4. In setting 5, 6, and 7, the values are increased, although they are slightly different (with the value of setting 7 is the lowest). This implies that, for higher number of logistics providers (20), the solutions are also better. However, the FEV(CS) values are very low, i.e. minus values. This reflects the fact that accumulated difference between each agent's payoff and the average payoff is high. In other words, the agents' payoffs vary very much although V(CS) is high. This leads to the situation where some agents earn high payoffs while some agents earn low payoffs.

With FEV(CS)-oriented runs, the FEV(CS) values of settings with 10 logistics providers (setting 1, 2, 3 and 4) are quite consistent but are lower than that of settings with 20 logistics providers (setting 5, 6 and 7). Similarly, the V(CS) values of settings with 10 logistics providers (setting 1, 2, 3 and 4) are consistent but are lower than that of settings with 20

providers (setting 1, 2, 3 and 4) are consistent but are lower than that of settings with 20 logistics providers (setting 5, 6 and 7). Note that the FEV(CS) values of FEV(CS)-oriented runs are higher than that of V(CS)-oriented runs. On the other hand, the V(CS) values of FEV(CS)-oriented runs are lower than that of V(CS)-oriented runs. A simple implication of this is that with FEV(CS)-oriented run, agents' payoff are less different compared to that of V(CS)-oriented runs. Although the system's utility, i.e. V(CS), of FEV(CS)-oriented are not as high, the difference of agents' payoff is less, which implies that every has better chance survive in highly competitive environment.

Note that none of other solution concepts consider this fairness from the system perspective, i.e., the more agents survive the better for the system. This is important because the well being is distributed as much as possible among all agents. This brings security to the system that agents can make their own livings and will not be burdens of the system or other agents.

## 6.5 Conclusion

This chapter proposes a model of agents-based coalitions as well as optimal coalition structure in dynamic supply networks. In our setting, agents take two steps to form coalitions: $i$) agents in each sector form loosely-coupled coalitions in order to decrease the complexity of the negotiation, and, $ii$) agents form coalitions across sectors in order to deliver products to end customers. We propose a framework, which involves negotiation protocol and decision mechanism. The negotiation protocol allows thorough communication, i.e., buyers to buyers, buyers to sellers, sellers to sellers, buyer to LPs, and LPs to LPs. With respect to optimal coalition structure, we propose a new approach for achieving a more compromising solution, taking into account both fairness and efficiency to the system.

The work presented in this chapter has achieved the fourth objective of this research, namely, to adapt the algorithm presented in Chapter 3 so as: 4. to solve OCS problems in complex environments such as those in which coalitions involve 3 types of stakeholders, such as in the supply chain domain.

# Chapter 7

# Conclusion and Future Work

## 7.1 Introduction

Optimal coalition structure is a hard problem. It is known that its complexity is NP-hard. For a small input, the problem can be intractable. Research conducted so far in this area has dealt with considerably small number of agents in characteristic function form, where coalition value is known a priori. However, multiagent systems can involve a larger number of agents. Furthermore, coalition value is not known a priori in real world setting and solving optimization problems for coalition values can be hard by itself. Therefore, it is a challenge to deal with larger number of agents and more realistic settings.

We then set out the objectives of this research as follows:

1. to develop a best-first, anytime algorithm that is an efficient solution for OCS problems in environments where coalition values are known a priori. Then, to adapt the algorithm developed in objective 1:

2. to solve OCS problems in a linear environment where coalition values and coalition structure values are not known a priori but must be calculated

3. to solve OCS problems in an NP hard, non-linear environment where coalition costs and coalition structure values are not known a priori

4. to solve OCS problems in which coalitions involve more than two types of stakeholders, such as the supply chain domain.

The work in the preceding 4 chapters has achieved these objectives as set out in the following section.

## 7.2 Contribution

In Chapter 3, we lay down a foundation work for the whole thesis. We consider the characteristic function environment where coalition value is known a priori. We argue that the

state-of-the-art in the area can be misled by some types of data distribution. We want to achieve an algorithm which performs consistently and is not misled by any type of data distribution. We define a novel concept of best coalition which is most likely valuable to coalition structure. We invent a best-first search algorithm that repeatedly chooses the best candidate coalition from available coalitions to place in CS at its present level. We invent 5 extra data distribution types, in addition to existing standard ones existing in the literature, to evaluate the performance consistency. We are interested in achieving an algorithm that can reach (near) optimality in timely fashion, which is even more desirable than termination when solving NP-hard problem. We consider environments where the number of agents is ranged up to 26 agents, by which most typical computers can have maximal resources. We benchmarked the performance of our algorithm with the state-of-the-art approach of Rahwan et al [111], and show that our approach compares favourably, i.e. our algorithm reach optimality more quickly in all settings. We conducted experiments extensively to be certain that the algorithm is robust even though some of the data settings are unrealistic. This work has achieved our first objective:

1. to develop a best-first, anytime algorithm that is an efficient solution for OCS problems in environments where coalition values are known a priori

Having achieved a generic algorithm which performs consistently well in all data distributions, we move forward to deal with more realistic settings in Chapter 4. We consider linear production domain where coalition value is not known a priori-the coalition values need to be computed before optimal coalition structure can be computed. We also consider a larger number of agents, which typical computers do not have enough resources to handle. Therefore, it is desirable to quickly search for a small number of good coalitions, from which best coalitions can be determined and be placed in CS. We invent a heuristic for the agents to quickly locate good coalitions, and only a relatively small number of them (compared to all the possible coalitions). From these coalition, we proposed a distributed branch-and-bound algorithm for computing optimal coalition structure for linear production domains. We extended our previous algorithm in the deliberation process in order to improve the performance. The non-profitable coalitions are not generated by the deliberation algorithm. Then the information of remaining coalitions will be exchanged among agents. Lastly, each agent uses an existing algorithm [86] to compute optimal coalition structures. The empirical results show that our algorithm helps generate the optimal coalition structures much faster than an exhaustive search. Our algorithm dramatically reduces the number of coalitions generated hence reducing the number of coalition structures. As a result, the elapsed time of generating the coalition structures is relatively small. In doing so, we have

achieved objective 2 of this research, namely:

2. to solve OCS problems in a linear environment where coalition values and coalition structure values are not known a priori but must be calculated

We move one step further in Chapter 5, i.e. we consider optimal coalition structure in non-linear logistics domains. We retain the assumption we had in Chapter 4, i.e., coalition value is not known a priori and the number of agents is about the same. Therefore, the problem involves two steps: $i$) achieving coalition values, and $ii$) solving optimal coalition structure problem. Since optimizing non-linear logistics domains is a hard problem by itself, (near) optimal solution for the coalition is expected to achieve in timely fashion. However, solving optimal coalition structure is also a hard problem. Therefore, we are facing the problem of time allocation for solving both problem under time constraint. We investigated the performance of different time allocation strategies. The empirical results show that the strategy that allows more time to solve both problems yields better results. In doing so, we have achieved objective 3 of this research, namely

3. to solve OCS problems in an NP hard, non-linear environment where coalition costs and coalition structure values are not known a priori.

Finally, we use invented knowledge (algorithms) to solve more complex problems. In Chapter 6, we proposed a model for agents to form coalitions in supply chain networks. We also consider a new concept for considering the wealth of the system and whether it is distributed fairly and efficiently. For forming coalitions, agents take two steps: i) agents in each sector form loosely-coupled coalitions in order to decrease the complexity of the negotiation, and, ii) agents form coalitions across sectors in order to deliver goods to end customers. We proposed a framework, which involves negotiation protocol and decision mechanism. The negotiation protocol allows thorough communication, i.e., buyers to buyers, buyers to sellers, sellers to sellers, buyer to LPs, and LPs to LPs. For the fairness and efficiency, we propose a new solution, which measure the value of coalition in both respects. Based on the algorithms we have developed, we conducted experiments to investigate the results of various runs, each of which is to compare the results achieved by either choosing best coalition based on typical coalition values or the new concept. The empirical results show that proportion of number of various types of agents does affect both the performance of the algorithm and the quality of the solutions. In doing so, we have achieved objective 4 of this research, namely:

4. to solve OCS problems in which coalitions involve more than two types of stakeholders, such as the supply chain domain

## 7.3 Significance of the Research

Clearly, this work is important to researchers in the multi-agent systems (MAS) research field, particularly to those focusing on Optimal Coalition Structures (OCS), as it presents a number of algorithms which are significantly better than the current state-of-the-art solutions in that field. The main contribution of this research is the proposition of defining *best* coalition, which is used to determine the next best coalition to be placed in CS, and algorithm to generate coalition structures in a value-oriented fashion. This concept is unique and different from other existing algorithm which is mostly based on certain pattern, such as configuration. This novelty has laid down a concrete foundation for research in other chapters of this thesis as well as the research in multiagent system or cooperative game theory. This novel approach has been shown to be effective in four domains in which MAS and OCS approaches offer efficient solutions to often intractable problems. Within those real-world domains, the research has also experimented with novel pruning, deliberation and negotiation techniques, thereby extending the set of solutions available to MAS and OCS researchers. Finally, the research has established a new benchmark against which other OCS solutions can be tested.

Because the research has been applied in a number of domains, it is also of significance to researchers in those domains as well as. The application of our algorithm to the linear production game offers new insights into the solution of such games. Application of the algorithm to the non-linear problems in the logistics domain has demonstrated that the best-first anytime approach could be used to solve some instances of classic problems such as the Vehicle Routing Problem or the Traveling Salesman Problem. Further developing this work on logistics, we have applied the algorithm to more complex logistics tasks involving stakeholders from 3 separate sectors. This work significantly advances the ability of logistics researchers to solve real-world problems which are of particular interest in extended supply chains, and indeed supply networks. Finally, the research is significant to researchers in broader domains, i.e. qualitative optimal coalition structures, which has more applications in the real world. Some of these research communities have their industrial counterparts and the current research could be immediately used by those industries. Both of the logistics solutions presented in this research could be applied by Logistics Providers (LPs) or by supply chain managers in a wide range of industries.

## 7.4 Limitations

While our algorithm significantly increases the number of agents that can be studied in OCS problems, it too has its limitations. Because multi-agent systems are rationally bounded, our

algorithm cannot deal with huge numbers of agents. Furthermore, other specific domains involving large number of agents may need additional thoughts in order to take into account additional details that are relevant to the performance of the algorithms. To deal with large number of agents, it is very important that high-valued coalitions must be known before the generation of coalition structure can be generated. These high-valued coalitions need to be stored in memory as many as possible because the value of coalition structures depend on their values.

## 7.5 Future directions

Since the most important algorithm is presented in Chapter 3, improving it means improving other algorithms. The algorithm can still be improved significantly. At the moment, it has to search to the last available coalitions in CS even though it may not be able to finish CS. This leads to too many wasteful repetition. The other improvement can also be achieved by using a better pruning approach. The linear production algorithm, presented in Chapter 4, helps reduce the number of coalitions involved in generating optimal coalition structures, but it needs to be able to deal with larger numbers of agents if it is to be truly useful in real-world settings. Ideally, it should be able to deal with hundreds of agents. In addition, this algorithm shows much promise for use in areas such as integer programming and non-linear programming. Additional work in the area of supply chain management is suggested. Since one of the most important issues in supply chains is to minimize logistics costs, future work could focus on finding a mechanism that helps buyers to control the logistics process rather than leaving it to the sellers or to coalitions of sellers. We believe that with efficient negotiation protocol and decision making mechanism, the logistics costs can be significantly reduced.

## 7.6 Conclusion

Notwithstanding the limitations described above, the current research has achieved all 4 of its objectives and in doing so has made a significant contribution to the fields of MAS/OCS research. It has demonstrated the efficacy of a best first, anytime approach to MAS/OCS problems. Moreover, it has done so in a number of real-world settings which are significantly more complex than those used in many previous studies. The work is significance to both academia and to industry and has opened up what promises to be a very fruitful field for further research.

# Bibliography


[1] Joseph Abdou and Hans Keiding. *Effectivity Functions in Social Choice*. Springer–Verlag, Berlin / Heidelberg, Germany, 1991.

[2] David Allsopp, Patrick Beautement, Jeffrey Bradshaw, Edmund Durfee, Michael Kirton, Craig Knoblock, Niranjan Suri, Austin Tate, and Craig Thompson. Coalition agents experiment: Multiagent cooperation in international coalitions. *Intelligent Systems*, 17(3):26–35, May-June 2002.

[3] John Anderson, Brian Tanner, and Jacky Baltes. Dynamic coalition formation in robotic soccer. In *Proceedings of the AAAI-04 Workshop on Forming and Maintaining Coalitions and Teams in Adaptive Multiagent Systems*, pages 1–10, San Jose, CA, USA, July 2004. AAAI Press.

[4] David Applegate, Robert Bixby, Vasek Chvatal, and William Cook. *The Traveling Salesman Problem: A Computational Study*. Princeton University Press, Princeton, 2006.

[5] Frederick Asselin and Brahim Chaib-draa. Toward a protocol for the formation of coalitions of buyers. In *Proceedings of the 5rd International Conference on Electronic Commerce Research (ICERC-5)*, pages 1–10, Montreal, Canada, 2002.

[6] Frederick Asselin and Brahim Chaib-draa. Coalition formation with non-transferable payoff for group buying. In *Proceedings of the 2nd International Joint Conference on Autonomous Agents and Multiagent Systems (AAMAS 03)*, pages 922–923, Melbourne, Australia, 2003. ACM Press.

[7] Frederick Asselin and Brahim Chaib-draa. Performance of software agents in non-transferable payoff group buying. *Journal of Experimental and Theoretical Artificial Intelligence*, 18(1), 2006.

[8] Robert Aumann and Michael Maschler. *The Bargaining Set for Cooperative Games*. Princeton University Press, 1964.

[9] Moshe Babaioff and William E. Walsh. Incentive-compatible, budget-balanced, yet highly efficient auctions for supply chain formation. *Decision Support Systems*, 39(1):123–149, 2005.

[10] Alexander Belenky. Cooperative games of choosing partners and forming coalitions in the marketplace. *Mathematical and Computer Modelling*, 36(11-13):1279–1291, December 2002.

[11] Maria-Victoria Belmonte, Ricardo Conejo, J. L. Pérez-de-la Cruz, and Francisco Triguero. A robust deception-free coalition formation model. In *Proceedings of the 2004 ACM Symposium on Applied Computing (SAC '04)*, pages 469–473, Nicosia, Cyprus, 2004. ACM Press.

[12] Bastian Blankenburg and Matthias Klusch. On safe kernel stable coalition forming among agents. In *Proceedings of the 3rd International Joint Conference on Autonomous Agents and Multiagent Systems (AAMAS 04)*, pages 580–587, New York, USA, 2004. ACM Press.

[13] Bastian Blankenburg, Matthias Klusch, and Onn Shehory. Fuzzy kernel-stable coalitions between rational agents. In *Proceedings of the 2nd International Joint Conference on Autonomous Agents and Multiagent Systems (AAMAS 03)*, pages 9–16, Melbourne, Australia, 2003. ACM Press.

[14] Silvia Breban and Julita Vassileva. A coalition formation mechanism based on inter-agent trust relationships. In *Proceedings of the 1st International Joint Conference on Autonomous Agents and Multiagent Systems (AAMAS 02)*, pages 306–307, Bologna, Italy, 2002. ACM Press.

[15] Philippe Caillou, Samir Aknine, and Suzanne Pinson. A multi-agent method for forming and dynamic restructuring of pareto optimal coalitions. In *Proceedings of the 1st International Joint Conference on Autonomous Agents and Multiagent Systems (AAMAS 02)*, pages 1074–1081, Bologna, Italy, 2002. ACM Press.

[16] Georgios Chalkiadakis and Craig Boutilier. Bayesian reinforcement learning for coalition formation under uncertainty. In *Proceeding of the 3rd International Joint Conference on Autonomous Agent and Multi Agent Systems (AAMAS 04)*, pages 1090–1097, New York, USA, 2004. IEEE Computer Society.

[17] Eve Cohen, Roshan Thomas, William Winsborough, and Deborah Shands. Models for coalition-based access control (cbac). In *Proceedings of the 7th ACM Symposium on Access Control Models and Technologies*, pages 97–106, Monterey, CA, USA, 2002. ACM Press.

[18] Javier Contreras, Felix Wu, Matthias Klusch, and Onn Shehory. Coalition formation in a power transmission planning environment. In *Proceedings of the 2nd International Conference on Practical Applications of Agents and Multi-Agent Systems (PAAM 97)*, London, UK, April 1997. Practical Application Co Ltd.

[19] David Cornforth, Michael Kirley, and Terry Bossomaier. Agent heterogeneity and coalition formation: Investigating market-based cooperative problem solving. In *Proceedings of the 3rd International Joint Conference on Autonomous Agents and Multiagent Systems (AAMAS 04)*, pages 556–563, New York, USA, 2004. ACM Press.

[20] Jeffrey Cox and Edmund Durfee. Efficient mechanisms for multiagent plan merging. In *Proceedings of the 3rd International Joint Conference on Autonomous Agents and Multiagent Systems (AAMAS 04)*, pages 1342–1343, New York, New York, 2004. IEEE Computer Society.

[21] Viet Dung Dang and Nicholas Jennings. Generating coalition structures with finite bound from the optimal guarantees. In *Proceedings of the 3rd International Joint Conferrence of Autonomous Agent and Multi Agent System (AAMAS 04)*, pages 564–571, New York, USA, 2004. IEEE Computer Society.

[22] Viet Dung Dang and Nicholas Jennings. Generating coalition structures with finite bound from the optimal guarantees. In *Proceedings of the 3rd International Joint Conference on Autonomous Agents and Multiagent Systems (AAMAS 04)*, pages 564–571, New York, USA, 2004. IEEE Computer Society.

[23] Gorge Dantzig. *Linear Programming and Extensions, 10th Edition*. Princeton University Press, Princeton, 1963.

[24] Morton Davis and Michael Maschler. The kernel of a cooperative game. *Naval Research Logistics Quarterly*, 12(3):223–259, 1965.

[25] Joakim Eriksson, Niclas Finne, and Sverker Janson. Evolution of a supply chain management game for the trading agent competition. *AI Communications*, 19(1):1–12, 2006.

[26] Amy Fedyk, Gary Kratkiewicz, Jeff Berliner, Mark Davis, Beth DePass, Rich Lazarus, and Rusty Bobrow. Adaptive optimization of solution time in a distributed multi-agent system. In *Proceedings of the International Conference on Integration of Knowledge Intensive Multi-Agent Systems 2005 (KIMAS 05)*, Waltham, MA, USA, April 2005. IEEE Computer Society.

[27] Mark Fox, Mihai Barbuceanu, and Rune Teigen. Agent-oriented supply-chain management. *International Journal of Flexible Manufacturing Systems*, 12:165–188, April 2000.

[28] Mark. Fox, John Chionglo, and Mihai Barbuceanu. The integrated supply chain management system. Technical report, Deparment of Industrial Engineering, University of Toronto, 1993.

[29] Andreas Gerber and Matthias Klusch. Forming dynamic coalitions of rational agents by use of the dcf-s scheme. In *Proceedings of the 2nd International Joint Conference on Autonomous Agents and Multiagent Systems (AAMAS 03)*, pages 994–995, Melbourne, Australia, 2003. ACM Press.

[30] Donald Gillies. *Some Theorems on n-Person Games*. PhD thesis, Department of Mathematics, Princeton University, Princeton, New Jersey, USA, 1953.

[31] Claudia Goldman, Sarit Kraus, and Onn Shehory. Agent strategies: for sellers to satisfy purchase-orders, for buyers to select sellers. In *Proceedings of the 10th European Workshop on Modeling Autonomous Agents in a Multiagent World (Maamaw '01)*, pages 166–177, Annecy, France, 2001. Springer–Verlag.

[32] Claudia Goldman, Sarit Kraus, and Onn Shehory. On experimental equilibria strategies for selecting sellers and satisfying buyers. *Decision Support Systems*, pages 329–346, 2003.

[33] Nathan Griffiths and Michael Luck. Coalition formation through motivation and trust. In *Proceedings of the 2nd International Joint Conference on Autonomous Agents and Multiagent Systems (AAMAS 03)*, pages 17–24, Melbourne, Australia, 2003. ACM Press.

[34] Yi Han, Wen-Xiang Gu, Yang Li, Ming-Hao Yin, and Jing-Bo Zhang. Flexible graphplan based on heuristic searching. In *Proceedings of The International Conference on Machine Learning and Cybernetics 2006*, pages 160–163, Dalian, China, August 2006. IEEE Computer Society.

[35] Hiromitsu Hattori, Tadachika Ozono, and Toramatsu Shintani. Applying a combinatorial auction protocol to a coalition formation among agents in complex problems. In *Proceedings of the 2nd International Joint Conference on Autonomous Agents and Multiagent Systems (AAMAS 03)*, pages 1008–1009, Melbourne, Australia, 2003. ACM Press.

[36] Masaki Hyodo, Tokuro Matsuo, and Takayuki Ito. An optimal coalition formation algorithm for electronic group buying. In *Proceedings of the Society of Instrument and Control Engineers Annual Conference 2003 (SICE 2003)*, pages 3402–3407, Fukui Japan, August 2003. IEEE Computer Society.

[37] Masaki Hyodo, Tokuro Matsuo, and Takayuki Ito. An optimal coalition formation among buyer agents based on a genetic algorithm. In *Proceedings of the 16th International Conference on Developments in Applied Artificial Intelligence (IEA/AIE'2003)*, pages 759–767, Laughborough, UK, 2003. Springer–Verlag.

[38] Toshihide Ibaraki and Naoki Katoh. *Resource Allocation Problems: Algorithmic Approaches*. MIT Press, Boston, 1998.

[39] Samual Ieong and Yoav Shoham. Multi-attribute coalitional games. In *Proceedings of the 7th ACM conference on Electronic commerce (EC 06)*, pages 170–179, Ann Arbor, Michigan, USA, 2006. ACM Press.

[40] Takayuki Ito, Hiroyuki Ochi, and Toramatsu Shintani. A group buy protocol based on coalition formation for agent-mediated e-commerce. *International Journal of Computer and Information Science (IJCIS)*, 3(1):11–20, 2002.

[41] Charles Phillips Jr., T. Ting, and Steven Demurjian. Information sharing and security in dynamic coalitions. In *Proceedings of the 7th ACM Symposium on Access Control Models and Technologies*, pages 87–96, Monterey, CA, USA, 2002. ACM Press.

[42] James Kahan and Amnon Rapoport. *Theories of Coalition Formation*. Lawrence Erlbaum Associates, Hillsdale, NJ, USA, 1984.

[43] Steven Ketchpel. Forming coalitions in the face of uncertain rewards. In *Proceedings of the 12th National Conference on Artificial Intelligence (AAAI 94)*, volume 1, pages 414–419, Seattle, WA, USA, 1994. AAAI Press.

[44] Majid Khan and Ladislau Boloni. Convoy driving through ad-hoc coalition formation. In *Proceedings of the 11th Real Time and Embedded Technology and Applications*

*Symposium (RTAS 2005)*, pages 98–105, San Francisco, CA, USA, March 2005. IEEE Computer Society.

[45] Matthias Klusch and Andrea Gerber. Dynamic coalition formation among rational agents. *IEEE Intelligent Systems*, 17(3):42–47, 2002.

[46] Matthias Klusch and Onn Shehory. Coalition formation among rational information agents. In *Proceedings of the 7th European Workshop on Modelling Autonomous Agents in a Multi-Agent World*, volume 1038, pages 204–217, Eindhoven, Netherland, 1996. Springer–Verlag.

[47] Matthias Klusch and Onn Shehory. A polynomial kernel-oriented coalition algorithm for rational information agents. In *Proceedings of the 2nd International Conference on Multi-Agent Systems (ICMAS 96)*, pages 157–164, Kyoto, Japan, 1996. MIT Press.

[48] Sarit Kraus, Onn Shehory, and Gilad Taase. Coalition formation with uncertain heterogeneous information. In *Proceedings of the 2nd International Joint Conference on Autonomous Agents and Multiagent Systems (AAMAS 03)*, pages 1–8, Melbourne, Australia, 2003. ACM Press.

[49] Sarit Kraus, Onn Shehory, and Gilad Taase. The advantages of compromising in coalition formation with incomplete information. In *Proceedings of the 3rd International Joint Conference on Autonomous Agent and Multi Agent Systems (AAMAS 04)*, pages 588–595, Washington DC, USA, 2004. IEEE Computer Society.

[50] Donald Kreher and Douglas Stinson. *Combinatorial Algorithms Generation, Enumeration and Search*. CRC Press, FA, USA, 1999.

[51] Kate Larson and Tuomas Sandholm. Anytime coalition structure generation: an average case study. *Journal of Experimental & Theoretical Artificial Intelligence*, 12(1):23–42, January 2000.

[52] Kristina Lerman and Onn Shehory. Coalition formation for large-scale electronic markets. In *Proceedings of the 4th International Conference on Multi-Agent Systems (ICMAS 00)*, pages 167–174, Boston, MA, USA, 2000. IEEE Computer Society.

[53] Cuihong Li, Uday Rajan, Shuchi Chawla, and Katia Sycara. Mechanisms for coalition formation and cost sharing in an electronic marketplace. In *Proceedings of the 5th International Conference on Electronic Commerce (EC 03)*, pages 68–77, Pittsburgh, PA, USA, 2003. ACM Press.

[54] Cuihong Li and Katia Sycara. Algorithms for combinatorial coalition formation and payoff division in an electronic marketplace. Technical Report CMU-RI-TR-01-33, Robotics Institute, Carnegie Mellon University, Pittsburgh, PA, November 2001.

[55] Cuihong Li and Katia Sycara. Algorithm for combinatorial coalition formation and payoff division in an electronic marketplace. In *Proceedings of the 1st International Joint Conference on Autonomous Agents and Multiagent Systems (AAMAS 02)*, pages 120–127, Bologna, Italy, 2002. ACM Press.

[56] Rajiv Maheswaran and Tamer Bacar. Coalition formation in proportionally fair divisible auctions. In *Proceedings of the 2nd International Joint Conference on Autonomous Agents and Multiagent Systems (AAMAS 03)*, pages 25–32, Melbourne, Australia, 2003. ACM Press.

[57] Michael Maschler. An advantage of the bargaining set over the core. *Journal of Economic Theory*, 13(2):184–192, October 1976.

[58] Carlos Merida-Campos and Steven Willmott. Modelling coalition formation over time for iterative coalition games. In *Proceedings of the 3rd International Joint Conference on Autonomous Agents and Multiagent Systems (AAMAS 04)*, pages 572–579, New York, USA, 2004. ACM Press.

[59] John Nash. *Non-Cooperative Game*. PhD thesis, Department of Mathematics, Princeton University, Princeton, USA, May 1950.

[60] John Von Neumann and Oskar Morgenstern. *Theory of Games and Economic Behaviour*. Princeton University Press, Princeton, New Jersey, 1953 (1963 printing).

[61] Mark Nissen. Agent-based supply chain integration. *Information Technology and Management*, 2(3):289–312, July 2001.

[62] Timothy Norman, Alun Preece, Stuart Chalmers, Nicholas Jennings, Michael Luck, Viet Dang, Thuc Nguyen, Vikas Deora, Jianhua Shao, Alex Gray, and Nick Fiddian. Agent-based formation of virtual organisations. *Knowledge-Based Systems*, 17(2-4):103–111, 2004.

[63] Guillermo Owen. On the core of linear production games. *Mathematical Programming*, 9(1):358–370, 1975.

[64] Praveen Paruchuri, Milind Tambe, and Fernando Ordonez. Towards a formalization of teamwork with resource constraints. In *Proceedings of the 3rd International Joint*

*Conference on Autonomous Agents and Multiagent Systems (AAMAS 04)*, pages 596–603, New York, USA, 2004. ACM Press.

[65] Michal Pechoucek, Vladimr Mark, and Jaroslav Brta. A knowledge-based approach to coalition formation. *Intelligent Systems*, 17(3):17–25, May-June 2002.

[66] Jos Potters, Imma Curiel, and Stef Tijs. Traveling salesman games. *Mathematical Programming*, 53:199–211, January 1992.

[67] Talal Rahwan and Nicholas Jennings. Distributing coalitional value calculations among cooperating agents. In *Proceedings of the 25th National Conference on Artificial Intelligence (AAAI 05)*, pages 152–157, Pittsburgh, USA, 2005. AAAI Press.

[68] Talal Rahwan and Nicholas Jennings. An algorithm for distributing coalitional value calculations among cooperating agents. *Artificial Intelligence*, 171 (8-9):535–567, 2007.

[69] Talal Rahwan, Sarvapali Ramchurn, Viet Dang, Andrea Giovannucci, and Nicholas Jennings. Anytime optimal coalition structure generation. In *Proceedings of the 22nd National Conference on Artificial Intelligence (AAAI 07)*, pages 1184–1190, Vancouver, Canada, July 2007. AAAI Press.

[70] Talal Rahwan, Sarvapali Ramchurn, Viet Dang, and Nicholas Jennings. Near-optimal anytime coalition structure generation. In *Proceedings of the 20th International Joint Conference on Artificial Intelligence (IJCAI 2007)*, pages 2365–2371, Hyderabad, India, January 2007. Kaufman Morgan.

[71] Talal Rahwan, Nicholas Jennings Sarvapali Ramchurn, and Adrea Giovannucci. An anytime algorithm for optimal coalition structure generation. 34:521–567, 2009.

[72] Martin Rehk, Premysl Volf, and Michal Pechoucek. Multilevel approach to agent-based task allocation in transportation. In *Cooperative Information Agents X, Proceedings of the 10th International Workshop on Cooperative Information Agents (CIA 2006)*, volume 4149/2006 of *Lecture Notes in Computer Science*, pages 273–287, Edinburg, UK, 2006. Springer–Verlag.

[73] Joaquin Sanchez-Soriano, Marco Lopez, and Ignacio Garcia-Jurado. On the core of transportation games. *Mathematical Social Sciences*, 41(2):215–225, March 2001.

[74] Tuomas Sandhlom and Victor Lesser. Coalitions among computationally bounded agents. *Artificial Intelligence*, 94(1-2):99–137, July 1997.

[75] Toumas Sandholm and Victor Lesser. Coalition formation among bounded rational agents. In *Proceedings of the 14th International Joint Conference on Artificial Intelligence (IJCAI 95)*, pages 662–669, Montreal, Canada, January 1995. Kaufman Morgan.

[76] Tuomas Sandholm. *Negotiation Among Self-Interested Computationally Limited Agents*. PhD thesis, Department of Computer Science, University of Massachusetts (Amherst), September 1996.

[77] Tuomas Sandholm, Kate Larson, Martin Andersson, Onn Shehory, and Fernando Tohm. Coalition structure generation with worst case guarantees. *Artificial Intelligence*, 111(1-2):209–238, 1999.

[78] Tuomas Sandholm and Nir Vulkan. Bargaining with deadlines. In *Proceedings of the 16th National Conference on Artificial Intelligence (AAAI 99)*, pages 44–51, Orlando, FA, USA, 1999. AAAI Press.

[79] David Schmeidler. The nucleolus of a characteristic function game. *SIAM Journal of Applied Mathematics*, 17, 1969.

[80] Sandip Sen and Partha Sarathi Dutta. Searching for optimal coalition structures. In *Proceedings of the 4th International Conference on MultiAgent Systems (ICMAS 00)*, pages 287–292, Boston, MA, USA, 2000. IEEE Computer Society.

[81] Lloyd Shapley and Martin Shubik. On market games. *Journal of Economic Theory*, 1(1):9–25, June 1969.

[82] Loyld Shapley. A value for n-person games. *Contributions to the Theory of Games, volume 2 of Annals of Mathematics Studies*, pages 307–317, 1953.

[83] Onn Shehory, Gal Kaminka, and Eran Shoham. Multi-agent coalition re-formation and league ranking. In *Proceedings of the 3rd International Joint Conference on Autonomous Agents and Multiagent Systems (AAMAS 04)*, pages 1348–1349, New York, USA, 2004. IEEE Computer Society.

[84] Onn Shehory and Sarit Kraus. Coalition formation among autonomous agents: Strategies and complexity. In *From Reaction to Cognition, Selected Papers from the 5th European Workshop on Modelling Autonomous Agents in a Multi-Agent World (MAAMAW 93)*, volume 957/1995, pages 55–72. Springer Berlin / Heidelberg, 1995.

[85] Onn Shehory and Sarit Kraus. Task allocation via coalition formation among autonomous agents. In *Proceedings of the 14th International Joint Conference on Artificial Intelligence (IJCAI 95)*, pages 655–661, Acapulco, Mexico, August 1995. Morgan Kaufman.

[86] Onn Shehory and Sarit Kraus. Formation of overlapping coalitions for precedence-ordered task-execution among autonomous agents. In *Proceedings of the 2nd International Conference on Multiagent Systems (ICMAS 96)*, pages 330–337, Kyoto, Japan, December 1996. AAAI Press.

[87] Onn Shehory and Sarit Kraus. A kernel-oriented model for coalition-formation in general environments: Implementation and results. In *Proceedings of the 13th National Conference on Artificial Intelligence (AAAI 96)*, volume 1, pages 134–140, Portland, Oregon, 1996. AAAI Press.

[88] Onn Shehory and Sarit Kraus. Methods for task allocation via agent coalition formation. *Artificial Intelligence*, 101(1-2):165–200, 1998.

[89] Onn Shehory and Sarit Kraus. Feasible formation of coalitions among autonomous agents in non-super-additive environments. *Computational Intelligence*, 15(3):218–251, August 1999.

[90] Onn Shehory, Sarit Kraus, and Osher Yadgar. Emergent cooperative goal-satisfaction in large-scale automated-agent systems. *Artificial Intelligence*, 110(1):1–55, May 1999.

[91] Onn Shehory, Katia Sycara, and Somesh Jha. Multi-agent coordination through coalition formation. In *Proceedings of the 4th International Workshop on Intelligent Agents IV, Agent Theories, Architectures, and Languages (ATAL 1997)*, number 1365 in Lecture Notes on Computer Science, pages 143–154, Providence, RI, USA, 1998. Springer-Verlag.

[92] Leonid Sheremetov and José Romero Cortés. Agent organizations with utility-based fuzzy coalitions. In *Proceedings of the 1st International Joint Conference on Autonomous Agents and Multiagent Systems (AAMAS 02)*, pages 461–462, Bologna, Italy, 2002. ACM Press.

[93] Mark Sims, Claudia Goldman, and Victor Lesser. Self-organization through bottom-up coalition formation. In *Proceedings of the 2nd International Joint Conference*

*on Autonomous Agents and Multiagent Systems (AAMAS 03)*, pages 867–874, Melbourne, Australia, 2003. ACM Press.

[94] Reid Smith. The contract net protocol: High-level communication and control in a distributed problem solver. *IEEE Transactions On Computers*, C-29(12):1104–1113, 1980.

[95] Leen-Kiat Soh and Xin Li. An integrated multilevel learning approach to multiagent coalition formation. In *Proceedings of the 18th International Joint Conference on Artificial Intelligence (IJCAI 03)*, pages 619–624, Acapulco, Mexico, 2003. Morgan Kaufman.

[96] Leen-Kiat Soh and Costas Tsatsoulis. Allocation algorithms in dynamic negotiation-based coalition formation. In *Proceedings of the Workshop on Teamwork and Coalition Formation, AAMAS 2002*, pages 1–8, Bologna, Italy, 2002. ACM Press.

[97] Leen-Kiat Soh and Costas Tsatsoulis. Learning to form negotiation coalitioins in a multiagent system. In *Proceeding of the AAAI Spring Symposium on Collaborative Learning Agents 2002*, pages 106–112, Standford, CA, USA, 2002. AAAI Press.

[98] Leen-Kiat Soh and Costas Tsatsoulis. Utility-based multiagent coalition formation with incomplete information and time constraints. In *Proceedings of the IEEE International Conference on Systems Man and Cybernatics (SMC 2003)*. IEEE Computer Society, 2003.

[99] Chattrakul Sombattheera and Aditya Ghose. Agent-based coalitions in dynamic supply chains. In *The 9th Pacific Asia Conference on Information Systems*, Bangkok, Thailand, 2005.

[100] Chattrakul Sombattheera and Aditya Ghose. A pruning-based algorithm for computing optimal coalition structures in linear production domains. In *Advances in Artificial Intelligence, Proceedings of the 19th Conference of the Canadian Society for Computational Studies of Intelligence (AI 2006)*, Lecture Notes in Computer Science, pages 13–24, Quebec, Canada, 2006. Springer–Verlag.

[101] Chattrakul Sombattheera and Aditya K. Ghose. A distributed branch-and-bound algorithm for computing optimal coalition structures. In *Advances in Artificial Intelligence, Proceedings of the 4th Helenic Conference on AI*, volume 3955 of *Lecture Notes in Computer Science*, pages 334–344, Crete, Greece, 2006. Springer–Verlag.

[102] Richard Sterns. Convergent transfer schemes for n-person games. *Transactions of the American Mathematical Society*, 134(3), December 1968.

[103] Philip Straffin. *Game Theory and Strategy*. The Mathematical Association Of America, NY, USA, 1993.

[104] Jayashankar Swaminathan and Sridhar Tayur. Models for supply chains in e-business. *Management Science*, 49(10):1387–1406, 2003.

[105] Katia Sycara, Steven Roth, Norman Sadeh-Koniecpol, and Mark Fox. Managing resource allocation in multi-agent time-constrained domains. In *Proceedings of the 1990 DARPA Workshop on Innovative Approaches to Planning, Scheduling and Control*, pages 240–250, San Diego, CA, USA, 1990. Morgan Kaufmann.

[106] Stef Tijs. LP-games and combinatorial optimization games. *Cahiers du Centre d'Etudes de Recherche Oprationelle*, 34(2-3), 1992.

[107] Fernando Tohm and Toumas Sandholm. Coalition formation processes with belief revision among bounded-rational self-interested agents. *Journal of Logic and Computation*, 9(6):793–815, 1999.

[108] Maksim Tsvetovat and Katia Sycara. Customer coalitions in the electronic marketplace. In *Proceedings of the 4th International Conference on Autonomous Agents (ICMAS 05)*, pages 263–264, Barcelona, Spain, 2000. ACM Press.

[109] Maksim Tsvetovat, Katia P. Sycara, Yian Chen, and James Ying. Customer coalitions in electronic markets. In *Agent-Mediated Electronic Commerce III, Current Issues in Agent-Based Electronic Commerce Systems*, pages 121–138. Springer-Verlag, London, UK, 2001.

[110] Sander van der Putten, Valentin Robu, Han La Poutré, Annemiek Jorritsma, and Margo Gal. Automating supply chain negotiations using autonomous agents: a case study in transportation logistics. In *Proceedings of the 5th International Joint Conference on Autonomous Agents and Multiagent Systems (AAMAS 06)*, pages 1506–1513, Hakodate, Japan, 2006. ACM Press.

[111] William Walsh and Michael Wellman. Decentralized supply chain formation: A market protocol and competitive equilibrium analysis. *Journal of Artificial Intelligence Research*, 19:513–567, 2003.

[112] Te-Wei Wang and Suresh K. Tadisina. Simulating internet-based collaboration: A cost-benefit case study using a multi-agent model. *Decision Support System*, 43(2):645–662, 2007.

[113] Michael Wooldridge and Paul Dunne. On the computational complexity of coalitional resource games. *Artifitial Intelligence*, 170(10):835–871, 2006.

[114] Michael Wooldridge and Paul E. Dunne. On the computational complexity of qualitative coalitional games. *Artificial Intelligence*, 158(1):27–73, 2004.

[115] Junichi Yamamoto and Katia Sycara. A stable and efficient buyer coalition formation scheme for e-marketplaces. In *Proceedings of the 5th International Conference on Autonomous Agents (ICMAS 01)*, pages 576–583, Montreal, Quebec, Canada, 2001. ACM Press.

[116] Yiming Ye and Xun Yi. Coalition signature scheme in multi-agent systems. In *Proceeding of the 11th International World Wide Web Conference (WWW2002)*, Sheraton Waikiki Hotel, Honolulu, Hawaii, USA, May 2002. ACM Press.

[117] Yun Yeh. A dynamic programming approach to the complete set partitioning problem. *BIT Numerical Mathematics*, 26(4):467–474, 1986.

[118] Dong Won Yi, Soung Hie Kim, and Nak Hyun Kim. Combined modeling with multi-agent systems and simulation: Its application to harbor supply chain management. In *Proceedings of the 35th Annual Hawaii International Conference on System Sciences (HICSS 2000)*, pages 1615–1624, Hawaii, USA, January 2002. IEEE Computer Society.

[119] Gilad Zlotkin and Jeffrey Rosenschein. Coalition, cryptography, and stability: Mechanisms for coalition formation in task oriented domains. In *Working Notes of the AAAI Spring Syposium on Software Agents*, pages 87–94, Stanford, CA, USA, 1994. AAAI Press.