Heuristic Methods for Coalition Structure Generation

by

Amir Aatieff Amir Hussin

A Doctoral Thesis

Submitted in partial fulfilment of the requirements for the award of

Doctor of Philosophy of Loughborough University

June 14, 2017

Abstract

The Coalition Structure Generation (CSG) problem requires finding an optimal partition of a set of *n* agents. An optimal partition means one that maximizes global welfare. Computing an optimal coalition structure is computationally hard especially when there are externalities, i.e., when the worth of a coalition is dependent on the organisation of agents outside the coalition.

A number of algorithms were previously proposed to solve the CSG problem but most of these methods were designed for systems without externalities. Very little attention has been paid to finding optimal coalition structures in the presence of externalities, although externalities are a key feature of many real world multiagent systems. Moreover, the existing methods, being non-heuristic, have exponential time complexity which means that they are infeasible for any but systems comprised of a small number of agents.

The aim of this research is to develop effective heuristic methods for finding optimal coalition structures in systems with externalities, where time taken to find a solution is more important than the quality of the solution. To this end, four different heuristics methods namely tabu search, simulated annealing, ant colony search and particle swarm optimisation are explored. In particular, neighbourhood operators were devised for the effective exploration of the search space and a compact representation method was formulated for storing details about the multiagent system. Using these, the heuristic methods were devised and their performance was evaluated extensively for a wide range of input data.

Acknowledgements

After so much hard work I am finally able to submit this thesis which I hope will be at least a particle among the particles of knowledge. All of this would not have been possible without the dedication and persistence of my dear supervisor Shaheen Fatima. I owe you the world and with my deepest sincerity I would like to express a million thanks and gratitude to you, Dr Shaheen Fatima. You helped nurture my research by being supportive and patient with my pace throughout the period of my studies. Thank you for giving me a chance to be an apprentice as a PhD student under your supervision. I am truly proud to be a student of such an outstanding and intelligent academic and utmost expert in multi-agent systems.

Nothing is more important than family, without their support no one can sail through the rough seas of doing a PhD. My dearest wife Syuhada, may Allah bless you always. Your warmth, kind, patient and tender loving care made me cheerful whenever I hit a brick wall with the research. You, Adrianna, Adrian and Aarash sacrificed a lot for me throughout this journey, this leap of faith. Failure would be to fail you all. It was not easy but we did it together. Mama, Dedi, Thank you for believing in me and continuing to support me all throughout this journey, for your endless spiritual, physical and financial support.. To my little brother Aarieff, thank you for helping me apply for this scholarship and my sisters Aalia and Aafaff, you are the best sister a brother could ask for, you literally 'guaranteed' my success, without you I would have no scholarship. Mak, Bapak, thank you for always being there for me and for not losing hope in me. Atif Sana Tarar, thank you for keeping our family strong. Thank you Bidah, for filling in for Aarash. Bengah, Bede, Pik, Afiq, Nana, Ira, Aisyah, all of you add colours to my life.

A special thanks my former masters' supervisor, the late Dr. Mohd Syazwan Abdullah who encouraged me to pursue a PhD. Also to Judith Paulton who has tirelessly supported me throughout the course of my studies at Loughborough University, for always being supportive and always proactive in finding solutions when I needed help.

Thank you to all my compatriots from the MML clan. A personal thanks to those families who has contributed so much to me and my family, my most closest of friends in Loughborough Amran Ahmad, Farraen, Shafizal, Nafiss, Rifqi, Muhaimin, Syahibudil, Safwan, Syuib Rambat, Shahrulizan, Nashrudin, Ezhan Johaniff, Faezz, Zack, Zulkifli Omar and all the others which are too many to mention here.

Table of Contents

List of Figures

List of Tables

Chapter 1 Introduction

This chapter sets the background for this research and lists the main aims and objectives of this research.

1.1 Background

In the modern age of computing, complex systems have evolved to become more independent and self-aware. There is an increasing trend towards designing software systems that work mostly autonomously requiring little or no human input. An *intelligent agent* is an agent that exhibits three main characteristics (Wooldridge, 2009): *pro-activeness*, *reactivity* and *social ability*. An agent is said to be proactive if it exhibits goal-directed behaviour, i.e. is capable of taking initiative to better achieve its intended goal. It is said to be reactive if it can respond to changes in the environment in time for the response to be useful. It is said to be social if it is able to interact with other agents.

A multiagent system (MAS) is comprised of multiple intelligent agents that interact and coordinate with each other. A key characteristic of a MAS is that the individual agents in it have different skills and capabilities and each agent is limited in terms of its capabilities. Cooperation and coordination between agents is therefore necessary in order for them to achieve big and complex goals, i.e., goals that cannot be achieved by an agent individually. Working together in some instances also helps the agents achieve the goals more efficiently (Shehory & Kraus, 1998; Zlotkin & Rosenschein, 1994).

To bring about this cooperation, the individual agents in a MAS must form a group and work together. In this context, the following two terms are useful:

Coalition: It is a subset of the agents that comprise a MAS.

Coalition structure: A typical MAS requires the formation of several coalitions that work simultaneously. In this context, the term *coalition structure* refers to a *partition* of the agents in a MAS.

The following are some applications that require the formation of coalitions:

1. Autonomous sensors networks where coalitions help to improve the coverage of the surveillance area (Glinton et al., 2008).
2. A coalition of buyers working together to get cheaper prices by purchasing in bulk (Sukstrienwong, 2011).
3. Intrusion detection systems (IDSs) to provide secure and dependable cloud computing service (Liu et al., 2015).
4. Distributed vehicle routing applications where delivery companies need to optimally allocate resources to coalitions within a structure (Sandholm & Lesser, 1997).
5. E-commerce systems where buyers group together to obtain discounts (Tsvetovat & Sycara, 2000).
6. E-business systems that need to optimally allocate resources to market partitions (Norman et al., 2004).
7. Distributed grid computing where virtual organisations must optimally share resources (Foster & Kesselman, 2003; Yong et al., 2003).
8. Information gathering systems where clusters of information servers must process queries together by forming optimal coalitions (Klusch & Shehory, 1996).
9. Multi-sensor surveillance networks where coalitions are used to provide better coverage over a large area (Dang et al., 2006).

1.2 Coalition Formation in Multiagent Systems

The success of a MAS is frequently measured in terms of the overall system performance of the system. The performance of a MAS depends on what coalitions form. Coalitions can form in many different ways. If formed correctly, coalitions can increase the productivity of a MAS. Likewise, badly formed coalitions can be counter-productive. It is therefore important for the right coalitions to form. But finding the right coalitions is a difficult problem. This thesis is aimed at finding methods for overcoming this difficulty. More precisely, the aim of this research is to devise computational methods for finding optimal coalitions.

In order to understand the complexity of the problem, consider a system comprised of 3 agents. Let {a, b, c} be the set of agents. There are 7 different ways in which a coalition can form:

1. {a}
2. {b}
3. {c}
4. {a, b}
5. {b, c}
6. {a, c}
7. {a, b, c}

In general for a system comprised of *n* agents, there are 2^{n-1} possible coalitions.

Again consider the system comprised of 3 agents {a, b, c}. There are 5 possible coalition structures, i.e., 5 different ways of partitioning the 3 agents:

1. {{a}, {b}, {c}}
2. {{a, b}, {c}}
3. {{a, c}, {b}}
4. {{a}, {b, c}}
5. {{a, b, c}}

In general, for a system of n agents, the number of possible coalition structures is the Bell number (B_n) (see section 1.2.4 for details).

Different coalition structures yield different levels of performance. The problem is therefore to determine an optimal coalition structure, i.e., a structure that optimizes system performance.

In order to find an optimal coalition structure, we must first use a suitable representation for describing the *worth of a coalition* and the *worth of a coalition structure*. The most commonly used representation comes from the literature in game theory (Chalkiadakis et al., 2012).

A coalition game is defined in terms of the players playing the game (i.e., the agents that comprise a MAS) and the worth of coalitions and coalition structures (Chalkiadakis et al., 2012). There are two types of coalition games *characteristic function games* (CFG) and *partition function games* (PFG).

1. Characteristic function games (Di Mauro et al., 2010; Keinänen & Keinänen, 2008; Rahwan et al., 2012; Rahwan et al., 2013; Sen & Dutta, 2000; Yeh, 1986): These are games in which the worth of a coalition depends on its member agents alone.
2. Partition function games (Banerjee & Kraemer, 2010; Epstein & Bazzan, 2013; Michalak et al., 2008; Rahwan et al., 2012; Thrall & Lucas, 1963): These are games in which the worth of a coalition depends not only on its member agents but also on how the external agents form coalitions.

The difference between PFGs and CFGs is that PFGs take into account externalities from forming coalitions. In PFGs the utility that is obtained from forming a coalition could be influenced by other coalitions that are concurrently being formed. This means that the value of a coalition depends on the coalition structure it is embedded with.

Example

Consider a game comprised of three agents {a, b, c}. For a CFG, the value of a coalition, say {a}, is given in terms of the elements of the set {a}. The value of any coalition is a real number. Thus the value of {a} in the structure {{a}, {b, c}} is equal to the value of {a} in the structure {{a}, {b}, {c}}. However, this is not the case for PFGs. For a PFG, the value of {a} in the structure {{a}, {b, c}} need not be equal to the value of {a} in the structure {{a}, {b}, {c}}.

Most of the existing literature on determining optimal coalition structures has focussed on CFGs (Rahwan et al., 2015). However, in many cases, such as resource allocation (Dunne, 2005), the performance of a coalition is directly influenced how the external players are organized. In such cases, a coalition that increases its consumption of resources in one part of the system can have adverse effects that could impact the effectiveness of agents in another part of the system. Another example of the occurrence of externalities is conspiracy attempt in oligopolies. In these situations, cooperating organisations explore ways to undermine the competitiveness of other companies in the market (Catilina & Feinberg, 2006).

Type of externality

Externalities can be of two types (Rahwan et al., 2009): *positive* or *negative*. Externalities are said to be positive (negative) if the merger of two coalitions has a beneficial (harmful) impact on the external players.

An example of positive externalities is environmental policy arrangement among countries (Plasmans et al., 2006). A decision made by a coalition of countries to cut pollution levels will have a beneficial impact on the countries outside the coalition (Finus, 2003). An example of negative externalities is the merger of technological companies; if say a giant like Microsoft were to merge with Facebook, then this decision will typically have a negative impact on other companies such as Google and Oracle.

Negative externalities also arise when major technology corporations decide to cooperate in order to develop a new technology standard; those that are not part of the coalition see a reduction their competitive position. For instance when a group of companies decided to form Blu-ray Disc Association (BDA), the industry consortium that develops and licenses Blu-ray Disc technology, the others that did not join BDA were recipients of negative externalities. The effects of this can clearly be seen in the collapse of HD DVD developed by the DVD Forum (Edwards et al., 2005). PFGs are necessary to model such systems with externalities.

1.2.1 Coalitions and Coalition Structures

In this section, we will introduce notation and formalize terms. The set of agents in a MAS will be denoted $Ag = \{a1, \ldots., a\boldsymbol{n}\}$. The term *coalition* refers to a non-empty subset of Ag. The term *coalition structure* refers to a set of pairwise-disjoint coalitions. Formally, a coalition structure, CS is a set $CS = \{C_1 \ldots . C_k\}$ of disjoint coalitions such that

$\cup Ci = C$, and $C_i \cap C_j = \varnothing$ for every $i, j \in \{1 \ldots . k\}$ and $i \neq j$.

We will denote the set of all coalition structure over C as Π^C.

For example, for the four agent set $Ag = \{a1, a2, a3, a4\}$, a possible coalition is $C = \{a2, a3, a4\}$ while a possible coalition structure is $\big\{\{a1\}, \{a2\}, \{a3, a4\}\big\}$.

A coalition in a coalition structure will be denoted as C combined with a subscript for example C_1 or C_i and combined with primes either in the form of C' or C''.

In coalitional games where externalities are considered, an **embedded coalition** is a pair consisting of a coalition and a coalition structure over Ag. For example, (C, CS) is an embedded coalition where C represents a coalition and CS represents a coalition structure which includes C, i. e.,

$$CS \in \Pi^{Ag} \text{ and } C \in CS.$$

The space of all embedded coalitions will be denoted $\mathcal{E}C$.

1.2.2 Characteristic Function Games

A characteristic function games is a pair (Ag, v) where Ag denotes a finite set of agents and $v: 2^{Ag} \rightarrow \mathbb{R}$ is the *characteristic function* representing the value $v(C)$ given to coalition $C \in \mathrm{P}(Ag)$ where C is the powerset of Ag excluding the empty set. The value $V(CS)$ of a coalition structure CS is the sum of the values of the coalitions in the structure CS:

$$V(CS) = \sum_{C_i \in CS} v(C_i) \qquad (1.1)$$

1.2.3 Partition Function Games

Games in partition function form were introduced by Thrall and Lucas (Thrall & Lucas, 1963) based on research into cooperative game theory by Neumann and Morgenstern (Neumann & Morgenstern, 1947). For PFGs the value of a coalition is influenced both by the identities of its members together with the ways non-members are partitioned. A PFGs is a pair (Ag, v) where Ag is the set of agents and v, represents the value given to an embedded coalition, (C, CS). Thus the value of a coalition *C* in the coalition structure *CS* is given by $v((C, CS))$. Formally, this is expressed as $v: \mathcal{E}C \rightarrow \mathbb{R}$.

The value of a coalition structure $V(CS)$ is given by:

$$V(CS) = \sum_{C_i \in CS} v(C_i, CS) \qquad (1.2)$$

This means that in a typical CFG, a coalition $C \subseteq Ag$ has only one value, whereas in a PFG C may have as many values as the number of ways to partition the agents in $Ag \backslash C$.

1.2.4 Coalition Structure Generation (CSG)

The coalition structure generation (CSG) problem for both CFGs and PFGs involves finding an **optimal coalition structure** $CS^* \in \Pi^{Ag}$ such that:

$$CS^* \in \underset{CS \in \Pi^{Ag}}{argmax}\big(V(CS)\big).$$

Solving the CSG problem is important because there are numerous applications in various fields including bioinformatics, data mining, semantic web, natural language processing and machine learning (Di Mauro et al., 2014). The problem of finding an optimal coalition structure is computationally hard (Aziz & De Keijzer, 2011). While it is solvable using brute force search, its time complexity is exponential in the number of agents in Ag. This is not feasible with the processing capacity of the current state of the art computers.

For n agents, the number of possible coalition structures is given by the *Bell number* B_n (Graham et al., 1994) or Stirling number of the second kind $S(n, k)$ (Knuth, 1992) where the nth of these numbers counts the number of different ways to partition a set with n elements into exactly k nonempty subsets. This is calculated by the following recursive formula:

$$B_{n+1} = \sum_{k=0}^{n} \binom{n}{k} B_k.$$

Table 1.1 – Number of possible coalition structures.

Number of elements/agents n	Bell number B_n (Possible Number of Coalition Structures)
5	52
10	115,975
15	1,382,958,545
20	51,724,158,235,372
25	4,638,590,332,330,743,949
27	545,717,047,947,902,329,359

Table 1.1 shows how quickly the Bell number grows in n. Given these numbers, it is clear that a brute force method using present computational technology is impractical for finding an optimal coalition structure. Alternative methods that are practically feasible must be devised. This is the goal of this thesis.

1.2.5 Methods for Optimal Coalition Structure Generation

Existing methods for solving the coalition structure generation problem can be divided into three categories (Service & Adams, 2011):

(i) Anytime algorithms (Changder et al., 2016a; Rahwan et al., 2009; Rahwan et al., 2012; Sandholm et al., 1998; Service & Adams, 2010) – the quality of the solution generated improves with their execution time. Their disadvantage is that, in the worst case, they need to check all coalition structure requiring $O(n^n)$ time complexity.

(ii) Design-to-time algorithms (Yeh, 1986) – These methods guarantee returning an optimal solution, however can only do so if allowed to run to completion. DP (Yeh, 1986) belongs to this category and has a time complexity of $O(3^n)$.

(iii) Heuristics algorithms (Di Mauro et al., 2010; Guo & Wang, 2006; Keinänen & Keinänen, 2008; Sen & Dutta, 2000; Sukstrienwong, 2011) – These algorithm prioritise speed over solution quality. However, when used, it is impossible to provide any form of guarantees.

For CFGs, and more so for PFGs, finding an optimal coalition structure is computationally hard. In the existing literature, a number of methods have been studied for CFGs but the optimal coalition structure determination problem for PFGs has only recently become the focus of attention (see Chapters 3 and 4 for a review of existing methods). A number of deterministic methods have been developed for PFGs but they have exponential time complexity. This presents the need for developing effective heuristic methods for finding a good enough solution as quickly as possible, especially for settings with a large number of agents. Such methods are important for example in mission critical systems where a group of agents representing emergency responders need to partition their resources so the emergency situation is handled optimally. In these systems the agents need to react quickly and time lost looking for the absolute optimal can severely impact on handling the emergency. A quick locally optimal solution would be better than a delayed globally optimal one because the situation may have changed during the time (Di Mauro et al., 2014).

Against this background, the objectives of this research are as follows.

1.3 Research Objectives

The main objective of this research is to develop effective heuristic solutions to the coalition structure generation problem. To this end, four different heuristic search methods will be explored. Specifically, these four methods are as follows:

1. Tabu search method
2. Simulated annealing method
3. Ant colony search method
4. Particle swarm optimization method

A key consideration in the design of heuristics will be their suitability to CFGs and also to PFGs. Other considerations are their running time memory space requirement, and their scalability.

Given the complexity of the problem, it is especially important and desirable when designing an algorithm, to balance the running time and the amount of memory required to execute a method. This can be achieved, perhaps by using efficient memory management which has been proven successfully adaptable in other combinatorial optimisation problems (Galinier et al., 2008). For example, having a tabu list which is relatively small could potentially be a more efficient way of using memory compared to the memory usage of other algorithms such as Dynamic Programming (Yeh, 1986). It is also worth noting there are no existing heuristic algorithms solving the optimal coalition structure generation problem for PFGs.

Thus, the aim of this thesis is to explore the above list heuristic approaches for solving the coalition structure generation problem and conduct a comparative analysis of their performance. Performance will be measured in terms of the above mentioned desirable characteristics.

1.4 Research Contributions

Heuristic methods have been applied to related optimisation problems such as the travelling salesman problem (Fiechter, 1994), bin packing problem (Lodi et al., 2004) and scheduling problems (Nonobe & Ibaraki, 2002). In solving the CSG problem **without** externalities i.e. CFGs, there are several heuristic methods such as greedy (Di Mauro et al., 2010), ant colony (Sukstrienwong, 2011), simulated annealing (Keinänen & Keinänen, 2008) and genetic algorithms (Sen & Dutta, 2000) that have been applied with promising results. However, there are no existing heuristic methods for generating optimal coalition structures for PFGs. This research contributes to the state of the art in the following ways:

1) Devising a range of heuristic methods for solving the coalition structure generation problem for CSGs and for PFGs.
2) Devising neighbourhood operators for effective exploration of the search space.
3) Devising compact representations for PFGs.
4) Analysing the performance of each heuristic method.
5) Conducting a comparative analysis of the heuristic methods.

Publications from this research

A. Hussin and S. Fatima (Hussin & Fatima, 2016), Heuristic methods for optimal coalition structure generation, Lecture Notes in AI, 10207, pages 1 – 16, 2017. (DOI: 10.1007/978-3-319-59294-7 11)

1.5 Thesis Structure

The remainder of the thesis is organized as follows. Chapter 2 provides background information on coalitional games. Chapters 3 and 4 review the existing algorithms for solving the coalition structure generation problem. The methods reviewed cover all three types of methods present in literature, namely anytime algorithms, design to time algorithms and heuristic algorithms. Chapter 3 reviews the existing literature for CFGs and Chapter 4 covers existing methods for solving the CSG problem for PFGs. These including centralised and distributed methods.

Chapter 5 describes the four heuristic search methods: tabu search, simulated annealing, ant colony search, and particle swarm search methods.

Chapter 6 is a description of the set-up for conducting simulations for performance evaluation.

Chapter 7 provides the result of the simulations. Performance is evaluated both in terms of the time taken to generate a solution and the quality of solution. The heuristic methods are also evaluated in terms of their scalability and how well they handle externalities. This chapter analyses the performance of each of the four heuristic methods for six different probability distribution data.

Chapter 8 presents an analysis of the average performance of each heuristic method across all the data sets. Scalability and the impact of externalities on performance are also analysed.

Chapter 9 lists the main conclusions of this research and provides pointers for further research.

Chapter 2 Coalitional Games

This chapter is a background on coalitional games. Section 2.1 introduces characteristic function games and provides details regarding the definition of values of coalitions. Section 2.2 does the same for partition function games.

2.1 Characteristic Function Games

In some multi-agent systems, each coalition pursues its own goal with little or no interaction with other coalitions. A lack of interaction between coalitions means that the value generated by a coalition is independent of the external coalitions. Games with no externalities are known as *characteristic function games* (CFGs).

A CFG is represented as a pair (Ag, v) where Ag is the set of agents and v is the *characteristic function* that gives the value of any coalition (Neumann & Morgenstern, 1947; Rapoport, 1970).

$$v: 2^A \rightarrow \mathbb{R}.$$

Figure 2.1 is an illustration of values for an example game of 3 agents.

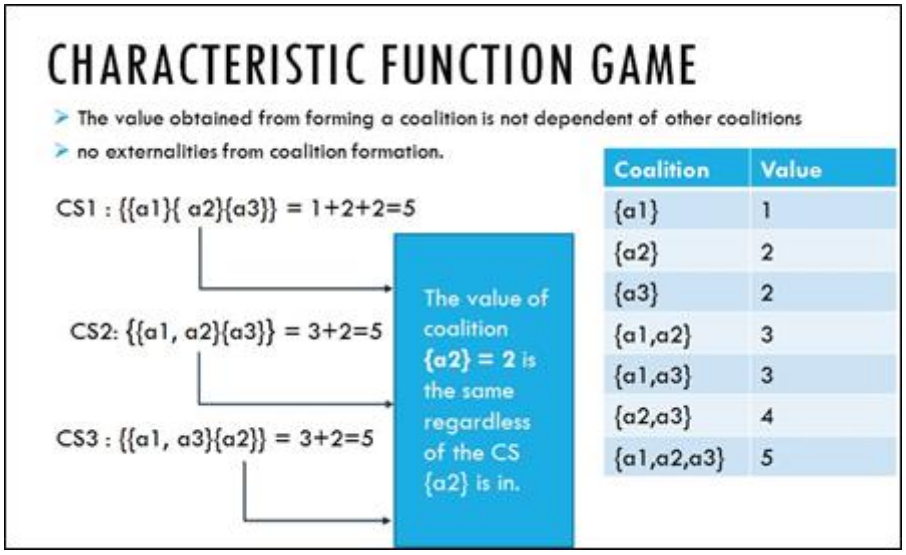

Figure 2.1 – A typical characteristic function game and its input.

Definition 1. A coalition structure is an exhaustive partition of a set of agents Ag into pairwise-disjoint (or non-overlapping) coalitions. For example, for $Ag = \{a1, a2, a3\}$, there are exactly 5 possible coalition structures:

$\{\{a1\},\{a2\},\{a3\}\}$, $\{\{a1, a2\},\{a3\}\}$, $\{\{a1, a3\},\{a2\}\}$, $\{\{a1\},\{a2, a3\}\}$ and $\{\{a1, a2, a3\}\}$

For n agents, there are Bell (B_n) $\sim O(n^n)$ possible coalition structures, i.e., the number of coalition structures is exponential in n.

The value of a coalition structure is the sum of the values of its coalitions:

$$V(CS) = \sum_{C_i \in CS} v(C_i) \qquad (2.1)$$

The precise details about how the value of a coalition is defined will be the subject of Chapter 6.

2.2 Partition Function Games

For partition function games (PFGs), the value of a coalition can be effected by the way the agents external to the coalition are organised (Thrall & Lucas, 1963).

2.2.1 The General Partition Function Game

A coalition structure CS is a partition of the set of agents in Ag. Any coalition $C \subseteq Ag$ that is a member of a coalition structure CS is embedded in CS. If the value assigned to a coalition C is influenced by other coalitions in that structure, then each coalition may have different values depending on which structure it is embedded in. Let $(C; CS)$ denote an embedded coalition. Let $\mathcal{E}C$ represent the set of all embedded coalitions, and Π the set of all coalition structures.

A PFG is comprised of:

- A set of agents, $Ag = \{a_1, .., a_n\}$
- A partition function w that takes an embedded coalition $(C, CS) \in \mathcal{E}C$ as input and assigns a real number value to coalition C in the coalition structure CS.

$$w: \mathcal{E}C \rightarrow \mathbb{R}$$

The value of a coalition structure CS is given by:

$$V(CS) = \sum_{C_i \in CS} w(C_i, CS) \qquad (2.2)$$

This is illustrated in Figure 2.2. A mentioned in Chapter 1, externalities are two main types: positive and negative. However, in some games, externalities may occur in both positive and negative forms. These are called *mixed externalities* games.

A mixed externalities game consists of a set of agents, Ag and a partition function which takes, as input, every feasible coalition structure (CS), and for each coalition in each structure, outputs a numerical value that reflects the performance of the coalition in that structure which can either have positive (increase) or negative (decrease) value when moving from one structure to the other.

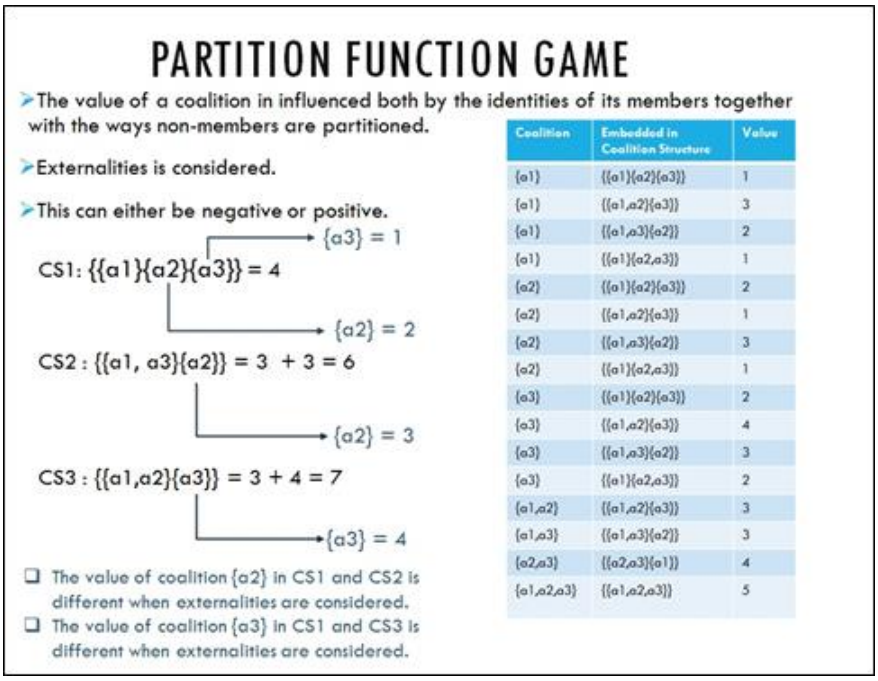

Figure 2.2 – A typical partition function game and its input.

Example

Given two coalition structures CS and CS' where $CS = \{\{C_1\}, \{C_2\}, \{C_3\}\}$ and $CS' = \{\{C_1\}, \{C_2 \cup C_3\}\}$ the value of $\{C_1\}$ may be different in CS' compared to CS as a result of the merger between C_2 and C_3. This condition effecting C_1 is usually attributed as an externality implied on C_1 by the formation of coalition $\{C_2 \cup C_3\}$ resulting from the merger of C_2 and C_3 (Michalak et al., 2008).

Types of PFGs

Three types of Partition Function Games (PFGs) have been studied in literature:

- Games with positive externalities (Rahwan et al., 2012).
- Games with negative externalities (Rahwan et al., 2012).
- Games with mixed externalities (Banerjee & Kraemer, 2010).

A brief description of each is as follows:

1. Games with positive externalities – In games with positive externalities, the merger of some coalitions in a coalition structure adds value to the players who are external to that coalition, making them better off. In other words, the merger of any two coalitions gives a positive impact to other existing coalitions in the system (Rahwan et al., 2012). An example of environmental policy arrangement maybe the decision of a coalition of countries to cut pollution levels. This decision will impact other countries leading to positive externalities (Finus, 2003).

2. Games with negative externalities – For games with negative externalities, the merger of any two coalitions does not benefit the other existing coalitions (Rahwan et al., 2012). For example, collusion in oligopolies where cooperating organisations explore ways to undermine the competitiveness of other companies in the market (Catilina & Feinberg, 2006) gives rise to negative externalities.

3. Games with mixed externalities – In a game with mixed externalities, the formation of a new coalition can either induce a positive or negative externality upon other (non-member) coalitions in the structure. This game is a neither positive only nor negative only externality game as considered by Michalak et al. (2008). A game with mixed

externalities can be modelled using agent types where two types of agents will form coalitions that induces either a positive or negative externality on the other type (Banerjee & Kraemer, 2010).

In general, for games with mixed externalities, it would be very difficult to find a solution to the coalition structure generation problem without exploring the entire space and checking every coalition structure (Rahwan et al., 2015). To put in perspective, consider a coalition structure generation problem with externalities where all the structures have been examined except one single structure. It is possible that this single structure has a higher value compared to all the other coalition structures in the system resulting from the externalities imposed on it.

In order to understand the degree of complexity involved in solving the CSG problem for PFGs, we must compare the size of their search space to the size of the search space for CFGs.

Storing the values of coalitions by exhaustively listing them in memory is infeasible for all but very small PFGs, i.e., those with very few agents. This is because the number of possible partitions grows very rapidly as the number of agent increases. For example, for 20 agents, there are 4×10^{14} possible structures and exhaustive enumeration requires 394 terabytes of memory (Rahwan et al., 2012). For PFGs, any coalition can have as many different values as there are coalition structures. Storing the value of each coalition in memory is therefore infeasible for any but very small games. Figure 2.2 gives an indication of the size of search space for a PFG of 3 agents.

We therefore need more concise methods for dealing with this problem. The following section provides details regarding our approach for dealing with this problem. Note that Section 2.2.2 is our contribution and does not form part of literature review.

2.2.2 A compact representation for PFGs

In order to define coalition values compactly, we used the following approach. The value of a coalition in a structure is calculated in terms of two components:

- an *externalities-free value*, and
- an *externalities factor*.

The externalities-free value of a coalition *C* is defined in the same way as for a CFG, i.e. $v: 2^A \rightarrow \mathbb{R}$. The externalities factor is defined in terms of two components:

1. The size of the coalition, i.e., the number of member agents.
2. The size of the structure it is embedded in, i.e., the number of coalitions in the structure.

For a coalition C, this factor is denoted *ef* and is calculated as follows:

$$ef = \frac{|C|}{d} \qquad (2.3)$$

where d is the number of coalitions in the structure in which *C* is embedded. As described below, this factor can be used to represent both positive and negative by adding and subtracting the externality free value using the formula given in Table 2.5.

2.2.2.1 Positive Externalities

In a PFG with positive externalities, the merger of any two coalitions decreases their joint value (or keeps it constant), and increases the values of other coalitions in the structure (or keeps them constant). The value $v(C)$ of any coalition C is calculated as follows. Let RV denote a randomly drawn value from any probability distribution. This random value is **increased** by a factor of ef to obtain $v(C)$ as follows:

$$v(C) = RV + (RV \times ef) \qquad (2.4)$$

Therefore, we have the following:

1. If external coalitions merge, the number of coalitions in the structure, i.e. d decreases. As a result, ef increases.
2. If ef increases, then $v(C)$ increases as well.

Examples of games where only positive externalities occur includes projects to reduce deforestation in a group of countries benefits other countries environmentally. Another example is the decision by one group of countries to reduce pollution, which has a positive impact on other countries or regions, it induces positive externalities. The sharing of cars between people in a community, if a car is a coalition then merging two coalitions of people into a single car (coalition) benefits the other coalitions as there will be less cars in the structure reducing the traffic resulting in a positive externality on other coalitions in the structure.

2.2.2.2 Negative Externalities

Externalities are said to be negative if the merger of two coalitions reduces the value (or keeps them constant) of the other coalitions in the structure. Again, RV denotes a randomly drawn value from any probability distribution. This random value is **decreased** by a factor of ef to obtain $v(C)$ as follows:

$$v(C) = RV - (RV \times ef) \qquad (2.5)$$

Therefore, we have the following:

1. If external coalitions merge, number of coalitions in the structure, i.e.,d decreases and ef increases, and so will the value to be deducted from $v(C)$.
2. If ef increases, then $v(C)$ decreases as the original value is subtracted by $RV \times ef$

Situations where negative externalities occur are for example, when high-tech companies decide to cooperate in order to develop a new technology standard, other companies lose some of their competitive position, i.e., they are subject to negative externalities. Research & Development coalitions among pharmaceutical companies, when two companies decide to jointly develop a new drug, the market position of other companies is likely to decrease. Collusion in oligopolies, exogenous coalition formation in e-market places, as well as multi-agent systems with shared resources and/or conflicting goals all invoke negative externalities.

A PFG with mixed externalities is represented as follows. Recall that when a coalition is effected by positive externalities, its value is calculated using Equation (2.4) and when it is effected by negative externalities, its value is calculated using Equation (2.5). In order to incorporate mixed externalities, we take the following approach which is similar to the approach taken by Banerjee and Kraemer (Banerjee & Kraemer, 2010):

1. The agents in a game are divided into different types.
2. Then, based on the types of member agents, coalitions are divided into different types.
3. Finally, based on the types of coalitions, coalition structures are divided into different types.

The underlying idea is that the type of a coalition is correlated to the type of externality it can impose on external players.

Details regarding the types of agents, the types of coalitions and the types of coalition structures are the subject of the following section.

2.2.2.3 A Representation for Mixed Externalities

Using this concept, agents are divided into two distinct types with the restriction that any game must contain both types of agents. The two types of agents are:

i) Type A
ii) Type B

Given that there are two type of agents, there can be the following three types of coalitions:

i) Type AA – All agents in the coalition are Type A.
ii) Type BB – All agents in the coalition are type B.
iii) Type MX – Some agents in the coalition are type A while others are type B.

Example:

In a 3-agent comprised of 2 Type A agents and 1 Type B agent, the following types of coalitions are possible (supposing that agents a1 and a2 are Type A, and agent a3 is Type B):

Table 2.1 – Possible Coalition Types for 3-agents.

Coalition	Coalition Type
{a1}	Type AA
{a2}	Type AA
{a1, a2}	Type AA
{a3}	Type BB
{a1, a3}	Type MX
{a2, a3}	Type MX
{a1, a2, a3}	Type MX

Three types of coalitions give rise to the following 5 types of coalition structures:

i) Type AABB – the structure consists of type AA and type BB coalitions
ii) Type AAMX – the structure consists of type AA and type MX coalitions.
iii) Type BBMX – the structure consists of type BB and type MX coalitions.
iv) Type AABBMX – the structure consists of type AA, type BB and MX coalitions.
v) Type MXMX – the structure consists only of type MX coalitions.

Example:

In a 3-agent system comprised of 2 Type A agents and 1 Type B agent, the following types (see Table 2.2) of coalition structures are possible (supposing that agents a1 and a2 are Type A, and agent a3 is Type B):

Table 2.2 – Possible Coalition Structure Types.

Coalition Structures	Coalition Structure Type
{{a1, a2, a3}}	Type MXMX (Grand Coalition)
{{a1, a2}, {a3}}	Type AABB
{{a1, 3}, {a2}}	Type AAMX
{{a1}, {a2, a3}}	Type AAMX
{{a1}, {a2}, {a3}}	Type AABB

In PFGs with only one agent of a type, certain types of coalition structure will not exist. The grand coalition will always be Type MXMX and the coalition structure of singletons is always type AABB if there is at least one agent of each type.

Example:

Number of Agents by Type	Coalition Structure Types that Do Not Exist
1 Type A agent, $n-1$ Type B agents	There will be no Type AAMX coalition structure
1 Type B agent, $n-1$ Type A agents	There will be no Type BBMX coalition structure

The effect of externalities

We defined the effect of externalities in terms of the types of coalitions. More precisely, the impact on a coalition when other coalitions form is defined as follows:

1) When a Type MX coalition forms as a result of a merger of a Type AA and a Type BB coalition, it will give positive externalities to Type AA coalitions but negative externalities to Type BB. The collusion will always benefits Type AA coalitions but always harms Type BB coalitions.
2) When a Type MX coalition forms and no Type AA or Type BB are present (i.e. the coalition structure consists only of Type MX coalitions), all coalitions in the structure are effected negatively. The idea is that Type MX coalition is a collusion between Type A and Type B agents. Therefore, if the coalition structure consists only of colluders that are colluding against each other, everyone loses.
3) When a Type AA coalition forms as a result of a merger of two Type AA coalitions, all other coalition types that are not singletons have their value increased due to positive externalities.
4) When a Type BB coalition form as a result of a merger of two Type BB coalitions, all other coalition types that are not singletons have their value increased due to positive externalities.

Special case for singletons (working alone may not be beneficial):

1) The effect of the formation of a structure on a singleton Type AA coalition is defined as follows. If the structure is Type AAMX or AABBMX, then the externality on the singleton is positive. If the structure is Type AABB, then the externality on the singleton is negative.
2) The effect of the formation of a structure on a singleton Type BB coalition is defined as follows. Regardless of the type of the structure, the externality on the singleton is negative.

The effect of externalities for coalition structures with no singletons is shown in Table 2.3.

Table 2.3 – Externalities for each coalition type in a structure type.

Coalition Structure Type	Coalition Structure Membership	First Coalition Type (Value)	Second Coalition Type (Value)	Third Coalition Type (Value)
AABB	{{AA}{BB}}	{AA} + (Increased)	{BB} + (Increased)	N/A
AAMX	{{AA}{MX}}	{AA} + (Increased)	{MX} + (Increased)	N/A
BBMX	{{BB}{MX}}	{BB} – (Decreased)	{MX} + (Increased)	N/A
AABBMX	{{AA}{MX}{BB}}	{AA} + (Increased)	{MX} + (Increased)	{BB} – (Decreased)
MXMX	{{MX}{MX}}	{MX} – (Decreased)	{MX} – (Decreased)	N/A

Different consideration is given for singletons. Working alone is always bad for Type BB coalitions. However for Type AA coalitions when there is a Type MX coalition in the structure, it benefits from this coalition type (see Table 2.4).

Table 2.4 – Externalities for singletons.

Coalition Structure Type	Coalition Structure Membership	First Coalition Type (Value)	Second Coalition Type (Value)	Third Coalition Type (Value)	Fourth Coalition Type (Value)	Fifth Coalition Type (Value)
AABB	{{A}{AA}{BB}{B}}	[A] – (Decreased)	[AA] + (Increased)	[BB] + (Increased)	[B] – (Decreased)	N/A
AAMX	{{A}{AA}{MX}}	[A] + (Increased)	[AA] + (Increased)	[MX] + (Increased)	N/A	N/A
BBMX	{{B}{BB}{MX}}	[B] – (Decreased)	[BB] – (Decreased)	[MX] + (Increased)	N/A	N/A
AABBMX	{{A}{AA}{MX}{BB}{B}}	[A] + (Increased)	[AA] + (Increased)	[MX] + (Increased)	[BB] – (Decreased)	[B] – (Decreased)

The calculation of the value of a coalition for each of the two types of externalities is summarized in Table 2.5.

Table 2.5 – Formula for calculating positive/negative externalities.

Equation	Externality	Formula
(2.4)	Positive Externality	$v(C) = RV + (RV \times ef)$
(2.5)	Negative Externality	$v(C) = RV - (RV \times ef))$

The calculation of the value of a coalition based on the structure it is embedded in is shown in Table 2.6.

Table 2.6 – Externalities from Other Coalitions in CS on Coalition C.

Coalition C	Other Coalitions in *CS*	Value Calculation
Type AA	Single Type	(2.4)
Type BB	Single Type	(2.4)
Type AA	At least one Mixed Type	(2.4)
Type BB	At least one Mixed Type	(2.5)
Type MX	Single Type	(2.5)
Type MX	At least one Type AA or BB	(2.4)
Type AA Singleton	At least one Mixed Type	(2.4)
Type AA Singleton	Single Type	(2.5)
Type BB Singleton	At least one Mixed Type	(2.5)
Type BB Singleton	Single Type	(2.5)

The type of externality (i.e., positive or negative) imposed on a coalition in a structure is summarised in Table 2.7.

Table 2.7 – Externalities on Coalition C in Coalition Structure CS.

Non-Singleton Coalitions			
	Coalition (C) Type	Coalition Structure (CS) Type	Externality
S1.	Type AA	Type AABB	POSITIVE
S2.	Type AA	Type AAMX	POSITIVE
S3.	Type AA	Type AABBMX	POSITIVE
S4.	Type BB	Type AABB	POSITIVE
S5.	Type BB	Type BBMX	NEGATIVE
S6.	Type BB	Type AABBMX	NEGATIVE
S7.	Type MX	Type MXMX	NEGATIVE
S8.	Type MX	Type AAMX	POSITIVE
S9.	Type MX	Type BBMX	POSITIVE
S10.	Type MX	Type AABBMX	POSITIVE
Singleton Coalitions			
S11.	Type AA Singleton	Type AABB	NEGATIVE
S12.	Type AA Singleton	Type AAMX	POSITIVE
S13.	Type AA Singleton	Type AABBMX	POSITIVE
S14.	Type BB Singleton	Type AABB	NEGATIVE
S15.	Type BB Singleton	Type BBMX	NEGATIVE
S16.	Type BB Singleton	Type AABBMX	NEGATIVE

This representation is much more compact relative to the representation where a different value is stored for each coalition in each coalition structure. This is because, the externality free value (i.e., RV) is all that is needed to be saved in memory. Externalities are then incorporated on the fly by employing Equation 2.4 for positive and Equation 2.5 for negative.

Let's view the externalities in the context of Producers (Agent Type B) and Consumers (Agent Type A) for each of the scenario in Table 2.7. In a typical setting Producers are traditionally competitors thus the assumption is for each case is:

S1. Suppose a structure of Type AABB has resulted from the merger of two Type AA coalitions. Then the externality imposed on each coalition in the resulting structure is positive. Intuitively, this means that when small consumer coalitions come together to form a bigger coalition of consumers, their purchasing power increases. As a result, trade for products at the top end of the market increases and therefore both producers and consumers benefit (this corresponds to scenarios S1 and S4) except for singleton consumers. Intuitively, this means that, as singletons, they have less purchasing power and are at the bottom end of the market and the prices of products at this end of the market remain unchanged. The singleton consumers correspond to scenario S11.

S2. Suppose a structure of Type AAMX has resulted from the merger of two Type AA coalitions. Then the externality imposed on each coalition in the resulting structure is positive. Intuitively, this means that when small consumer coalitions come together to form a bigger coalition of consumers their purchasing power increases. As a result, trade for products at the top end of the market increases and therefore both producers and consumers (i.e. coalitions of Type AA and MX) benefit everyone (this corresponds to scenarios S2 and S8). Intuitively this means that, even singleton consumers gain because of the existence of Type MX coalitions which may contain just one consumer. Since even single consumers in Type MX coalition benefits, this advantage is passed on to external singleton consumers .The singleton consumers correspond to scenarios S12 and S13.

S3. Suppose a structure of Type AABBMX has resulted from the merger of two Type AA coalitions. Then the externality imposed on each Type AA coalition in the resulting structure is positive. Intuitively, this means that when small consumer coalitions join together to form a bigger coalition of consumer their purchasing

power increases. This translates into real benefit for them and Type AA coalitions gets a positive externality. However the coalitions of Type BB get a negative externality (this corresponds to S6 and S16). Intuitively, this means that the needs of the newly merged Type AA coalition of consumers is met by producers in Type MX coalitions. Therefore Type MX coalitions benefit i.e. get a positive externality. Type AA singletons (singleton consumers) benefit and get a positive externality (this corresponds to S13).

S4. Suppose a structure of Type AABB has resulted from the merger of two Type BB coalitions. Then the externality imposed on each coalition in the resulting structure is positive. Intuitively, this means that when small producer coalitions come together to form a bigger coalition of producers their production power increases. As a result, all coalitions except the smallest of producers gain (are effected positively by externalities). What we mean by the smallest of producers is a singleton producers. The singleton producers corresponds to scenario S14 that get a negative externality.

S5. Suppose a structure of Type BBMX has resulted from the merger of two Type BB coalitions. Then the externality imposed on each Type BB coalition in the resulting structure is negative. Intuitively, this means that when small producer coalitions join together to form a bigger coalition of producers their production power increases, but this does not translate to a benefit but rather results in increased cost of creating a merger. This is because of the presence of Type MX coalitions. Intuitively this means that the consumers in the Type MX coalitions being already together in a coalition with producers prefer to work with those producers and not with the newly form merger of producers. This benefits Type MX coalitions and translates to positive externality on it. This corresponds to scenario S9.

S6. Suppose a structure of Type AABBMX has resulted from the merger of two Type BB coalitions. Then the externality imposed on each Type BB coalition in the resulting structure is negative (this corresponds to S6 and S16). Intuitively, this means that when small producer coalitions join together to form a bigger coalition of producers their production power increases, but this does not translate to a benefit but rather results in increased cost of creating a merger. This is because the needs of consumers are met by producers outside the newly formed merger. However, existing coalitions of consumers i.e. Type AA coalitions gain from the

merger of producers and this translates into positive externality on Type AA coalitions (this corresponds to S3 and S13). In the same way Type MX coalitions are affected by positive externality.

S7. Suppose a structure of Type MXMX has resulted from the merger of two Type MX coalitions. Then the externality imposed on each Type MX coalition in the resulting structure is negative. Intuitively, this means the merger is unnecessary and counterproductive.

S8. Suppose a structure of Type AAMX has resulted from the merger of two Type MX coalitions. Then the externality imposed on each Type MX coalition in the resulting structure is positive.

S9. Suppose a structure of Type BBMX has resulted from the merger of two Type MX coalitions. Then the externality imposed on each Type MX coalition in the resulting structure is positive. However Type BB coalitions are affected negatively.

S10. Suppose a structure of Type AABBMX has resulted from the merger of two Type MX coalitions. Then the externality imposed on each Type MX and each Type AA coalition in the resulting structure is positive. However Type BB coalitions are affected negatively.

The above discussion is just a possible way of viewing the definition of positive and negative externalities. However our aim is not to consider a specific PFG such as that for producers and consumers. Rather it is to define a general coalitional game with all three externalities i.e. positive, negative and mixed inherent in the game.

2.3 Chapter Summary

This chapter introduced the key concepts that underlie coalitional games. A brief introduction to Characteristic Function Games and Partition Function Games was given. Finally, we introduced our approach for a compact representation of PFGs.

Chapter 3 Coalition Structure Generation for Characteristic Function Games: A Review of Literature

As outlined Chapter 1, algorithms for finding an optimal coalition structure can be classified into 3 categories (Service & Adams, 2011):

- design-to-time algorithms,
- anytime algorithms, and
- heuristic algorithms.

This chapter provides a review of the existing methods in each of these three categories. Section 3.1 is about design-to-time algorithms, Section 3.2 about anytime algorithms, and Section 3.3. about heuristic algorithms.

3.1 Design-to-Time Algorithms

These algorithms are guaranteed to provide an optimal solution. However, a solution can only be provided when the algorithm terminates. Since they are not anytime, it is impossible to return any form of solution before completion. These algorithms have mostly been designed using dynamic programming (DP).

3.1.1 Dynamic Programming

Yeh (1986) developed a method based on dynamic programming (Bellman, 1952). Their algorithm maintains two tables f_1 and f_2 that contain an entry for every possible coalition. For every coalition $C \subseteq N$, $f_1[C]$ and $f_2[C]$ are computed as follows: A best possible split (if any) for C is stored in $f_1[C]$ and its evaluation in $f_2[C]$. If it is best not to split C then $f_1[C]$ contains C and $f_2[C]$ the value of C. The value of every splitting C', C'' of C is evaluated as $f_2[C'] + f_2[C'']$. This method does not evaluate the splitting of size s until it has finished computing f_2 for the coalition of sizes 1 to $s-1$. Table 3.1 shows an example of how f_1 and f_2 are computed for N={a1,a2,a3,a4}. Once f_1 and f_2 are computed for every coalition, the optimal

structure CS^* is computed recursively. In the example below, this is done by first setting CS^*={a1,a2,a3,a4}. Then looking at f_1{{a1,a2,a3,a4}} we find it is beneficial to split it into {a1,a2} and {a3,a4}. Similarly, it is beneficial to split {a1,a2} into {a1} and {a2}, but it is better to keep {a3,a4} as it is. The optimal structure is therefore {{a1}, {a2}, {a3,a4}}.

Although DP gives an exact optimum, it requires a large amount of memory for storing the three tables v, f_1 and f_2 (see the example in Table 3.1).

Table 3.1 – An example showing how f_1 and f_2 are calculated.

Input value	Size	Coalition	Evaluations to determine splitting is beneficial		f_1	f_2
v({a1}) = 35	1	{a1}	v({a1}) = 35		{a1}	35
v({a2}) = 45		{a2}	v({a2}) = 45		{a2}	45
v({a3}) = 30		{a3}	v({a3}) = 30		{a3}	30
v({a4}) = 50		{a4}	v({a4}) = 50		{a4}	50
v({a1,a2}) = 55	2	{a1,a2}	v({a1,a2}) = 55	$f_2[\{a1\}] + f_2[\{a2\}]$= 80	{a1} {a2}	80
v({a1,a3}) = 65		{a1,a3}	v({a1,a3}) = 65	$f_2[\{a1\}] + f_2[\{a3\}]$= 65	{a1,a3}	65
v({a1,a4}) = 85		{a1,a4}	v({a1,a4}) = 85	$f_2[\{a1\}] + f_2[\{a4\}]$= 85	{a1,a4}	85
v({a2,a3}) = 60		{a2,a3}	v({a2,a3}) = 60	$f_2[\{a2\}] + f_2[\{a3\}]$= 75	{a2} {a3}	75
v({a2,a4}) = 75		{a2,a4}	v({a2,a4}) = 75	$f_2[\{a2\}] + f_2[\{a4\}]$= 95	{a2} {a4}	95
v({a3,a4}) = 85		{a3,a4}	v({a3,a4}) = 85	$f_2[\{a3\}] + f_2[\{a4\}]$= 80	{a3,a4}	85
v({a1,a2,a3}) = 95	3	{a1,a2,a3}	v({a1,a2,a3}) = 95	$f_2[\{a1\}] + f_2[\{a2,a3\}]$= 95	{a2} {a1,a3}	105
v({a1,a2,a4}) = 125			$f_2[\{a2\}] + f_2[\{a1,a3\}]$= 105	$f_2[\{a3\}] + f_2[\{a1,a2\}]$= 85		
v({a1,a3,a4}) = 105		{a1,a2,a4}	v({a1,a2,a4}) = 125	$f_2[\{a1\}] + f_2[\{a2,a4\}]$= 110	{a2} {a1,a4}	130
v({a2,a3,a4}) = 120			$f_2[\{a2\}] + f_2[\{a1,a4\}]$= 130	$f_2[\{a4\}] + f_2[\{a1,a2\}]$= 105		
v({a1,a2,a3,a4}) = 145		{a1,a3,a4}	v({a1,a3,a4}) = 105	$f_2[\{a1\}] + f_2[\{a3,a4\}]$= 120	{a1} {a3,a4}	120
			$f_2[\{a3\}] + f_2[\{a1,a4\}]$=115	$f_2[\{a4\}] + f_2[\{a1,a3\}]$= 105		
		{a2,a3,a4}	v({a2,a3,a4}) = 120	$f_2[\{a2\}] + f_2[\{a3,a4\}]$= 130	{a2} {a3,a4}	130
			$f_2[\{a3\}] + f_2[\{a2,a4\}]$=105	$f_2[\{a4\}] + f_2[\{a2,a3\}]$= 110		
	4	{a1,a2,a3,a4}	v({a1,a2,a3,a4}) = 145	$f_2[\{a1\}] + f_2[\{a2,a3,a4\}]$= 155	{a3} {a1,a2,a4}	155
			$f_2[\{a2\}] + f_2[\{a1,a3,a4\}]$= 150	$f_2[\{a3\}] + f_2[\{a1,a2,a4\}]$= 155		
			$f_2[\{a4\}] + f_2[\{a1,a2,a3\}]$= 145	$f_2[\{a1,a2\}] + f_2[\{a3,a4\}]$= 140		
			$f_2[\{a1,a3\}] + f_2[\{a2,a4\}]$= 140	$f_2[\{a1,a4\}] + f_2[\{a2,a3\}]$= 145		

Figure 3.1 illustrates how this method traverses the coalition structure graph. The nodes of the graph represent coalition structures. Nodes are grouped in levels. The coalition structures at a level are obtained from the ones at the level below by splitting a coalition in the structure below. We start at level L_1 (i.e., the node {a1,a2,a3,a4}) and move upwards in the graph until an optimal structure (i.e. {{a1},{a2},{a3,a4}}) is found. In this figure, there are 3 paths from the initial node to the optimal one indicated as dotted, dashed and bold lines.

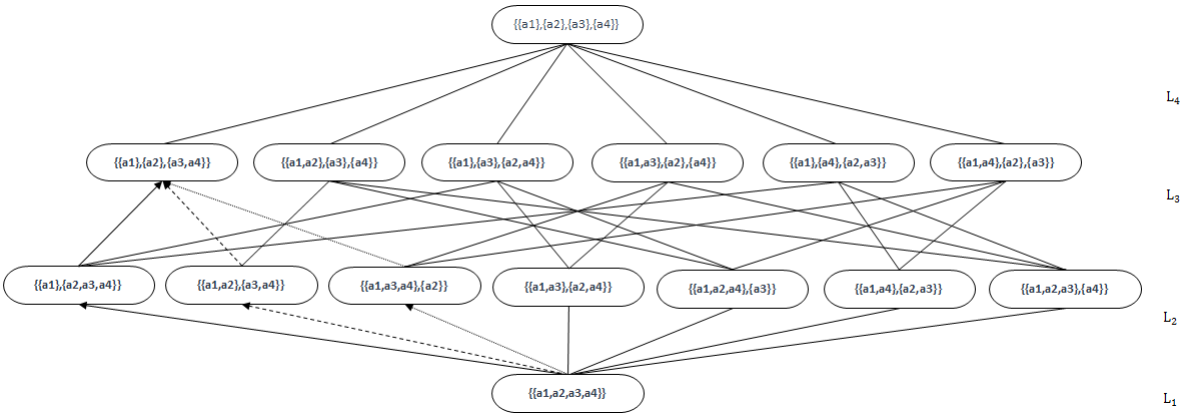

Figure 3.1 – Example DP movements for 4-agents.

Finding an optimal structure does not require evaluating every possible splitting of every possible coalition. It is possible to avoid evaluation of some splittings and still be able to find an optimal structure.

Nodes are divided into n levels where n is the number of agents in the system. In the example of Figure 3.1, $n = 4$. The set of coalition structures at level L_k consists of nodes with k coalitions. The method begins at L_1 with a coalition structure CS consisting of {a1,a2,a3,4}. Splitting the coalitions $C \in CS$ into $\{C', C''\}$ is shown for example by the dotted line with the arrow indicating the movement from the node consisting of CS to the node consisting of $(CS \backslash C) \cup \{C', C''\}$. DP evaluates all possible splits starting at the bottom node and moving upward path through the connected nodes until an optimal node is reached shown as the dashed line in Figure 3.1.

3.1.2 Improved Dynamic Programming

After evaluating the movements of DP, Rahwan & Jennings (2008a) concluded that it was possible to avoid evaluation of some splittings and still be able to find an optimal structure. In their method, they exploited this characteristic to more efficiently find an optimal structure. This method known as *improved dynamic programming* (IDP) (Rahwan & Jennings, 2008a) improves upon the memory usage (by storing less information from the splittings) and speed of DP (by avoiding some splittings). However, IDP, like the original DP is still not anytime and has the same time complexity of $O(3^n)$.

IDP was designed to perform fewer operations by avoiding unnecessary evaluations of as many splitting as possible while maintaining the guarantees of finding the optimal coalition structure. It is therefore necessary to eliminate as many unnecessary edges from the coalition structure graph as possible while avoiding the removal of those edges that lead to an optimal node. The method first ensures that each node in the coalition structure graph is connected to another node by at least one path. In a coalition structure graph, any coalition structure that has more than two coalitions will have more than one path leading to it (shown in Figure 3.2 as dotted, dashed and bold lines).

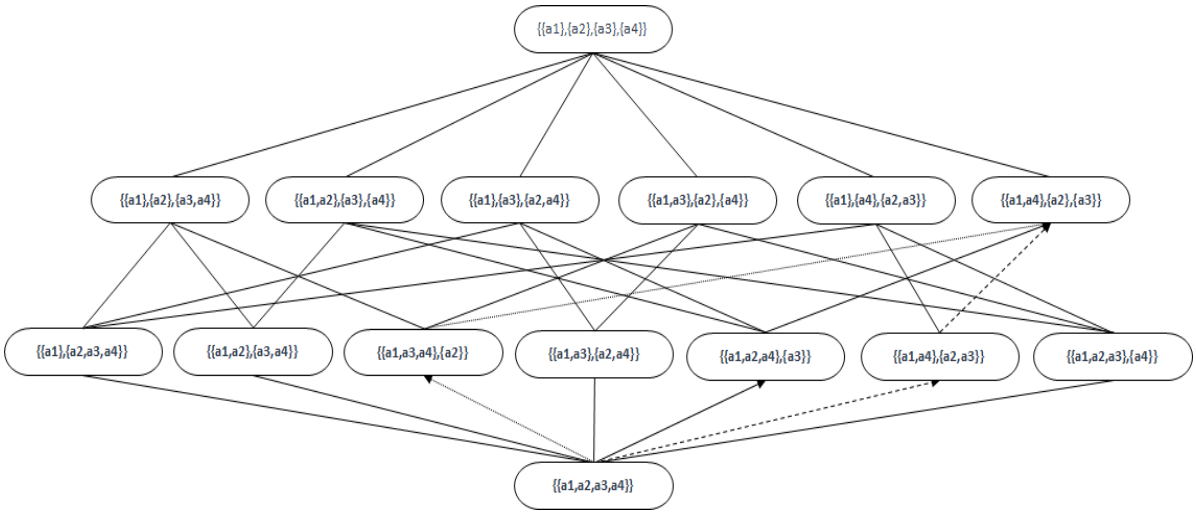

Figure 3.2 – Multiple paths leading to each CS with more than two coalitions.

IDP involves movements through the edges of a coalition structure graph where certain splits are not evaluated. This is represented by removing these edges from the graph while preserving a path leading to any possible optimal node. The remaining edges that are not removed should be sufficient for every node to have a path leading to it, thus IDP only evaluates those edges to find the optimal coalition structure. This is shown in Figure 3.3 where the dotted lines represent some of the redundant edges removed with every CS with more than 2 coalitions still having more than one edge connected to it. The removal of these edges means IDP performs fewer operations that DP. Simulation results showed that IDP evaluated around 38% of the splits evaluated by DP (Rahwan & Jennings, 2008a). However, owing to the tremendous complexity of dynamic programming, alternative methods with *anytime* property were developed.

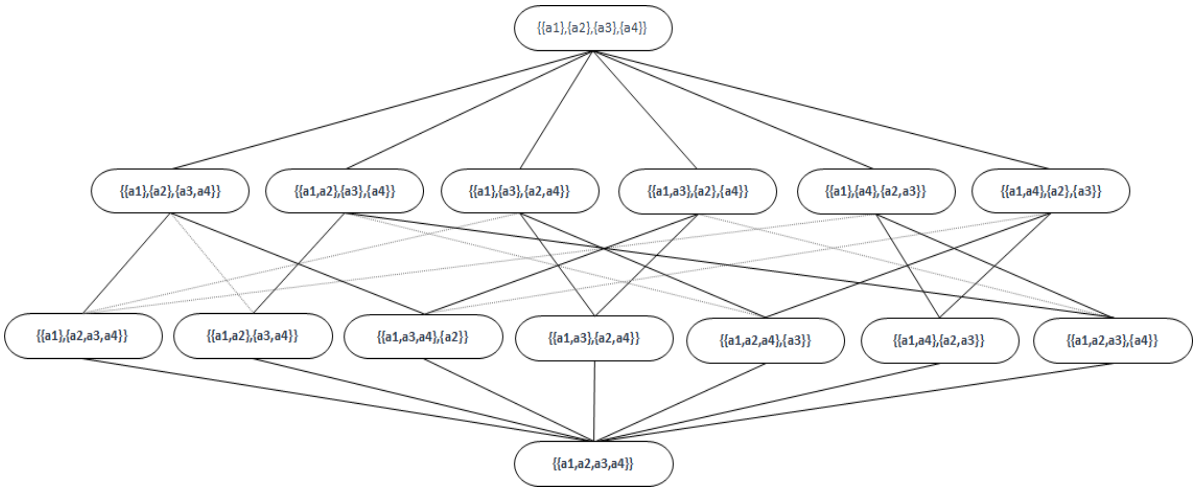

Figure 3.3 – Redundant edges removed as dotted lines.

Both design-to-time algorithms are able to return an optimal solution with a time complexity of $O(3^n)$. However IDP has some advantages over DP (see Table 3.2).

Table 3.2 – Comparison between DP and IDP.

Method	Advantages	Disadvantages
Dynamic Programming	• Returns absolute optimal.	• Requires large amount of memory to store the v, f_1 and f_2 tables. • Not Anytime, requires completion to return solution
Improve Dynamic Programming	• Returns absolute optimal • Avoids exploring redundant splittings.	• Requires large amount of memory despite evaluating only 38% of splits explored by DP. • Not Anytime, requires completion to return solution

In the next section, we will be looking at anytime algorithms.

3.2 Anytime Algorithms

The key feature of an anytime algorithm is the ability to allow early termination whilst still providing some guarantees on the solution. The disadvantage however is that most anytime algorithms have a worst case run time of $O(n^n)$. This means that, in the worst-case, they will end up exhaustively searching the entire space of all coalition structures.

Anytime methods can be classified into two types: coalition structure graph search methods and integer partition based search methods. In the following two subsections, we will look at each of these two types of methods.

3.2.1 Coalition Structure Graph Search

The first anytime algorithm for coalition structure generation was proposed by Sandholm et al. (1999). Their algorithm guarantees incremental improvements of the worst-case bound as the search progresses. The search space is divided into levels L_k, $1 \leq k \leq n$. A node at each level represents a coalition structure with k coalitions. For $n = 4$, this is shown in Figure 3.4.

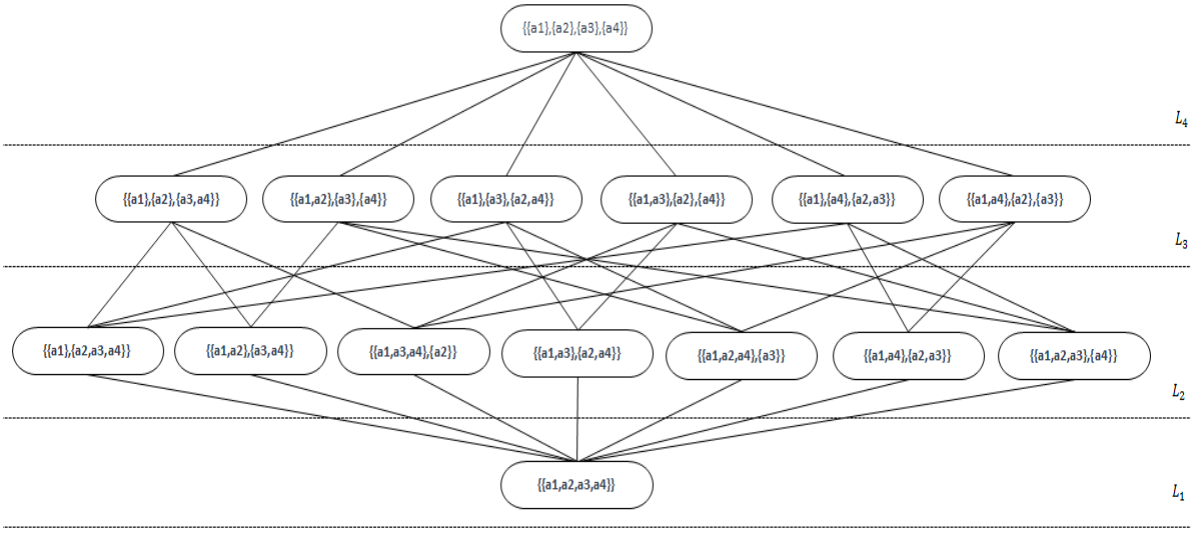

Figure 3.4 - Coalition structure graph for 4-agents (Sandholm et al., 1999).

Each level L_k consists of nodes with k coalitions. The algorithm moves through levels $L_1, L_2 \ldots \ldots L_n$ in a breadth-first search manner and searches through all coalition structures at level L_k. This algorithm starts by calculating the value of the structure at level L_1. Once this is completed it stores both the coalition structure and its value into memory as the current optimal. It then performs an exhaustive search of level L_2 i.e. it computes the value of every coalition structure on this level.

If the value of a structure at this level is found to be greater than the current optimal value (which before this was the highest value found in level L_1), this information is updated and

both the structure and its value is stored in memory as the current optimal. The levels $L_n, L_{n-1} \ldots \ldots L_3$ are then searched sequentially in an exhaustive manner as in level L_2 until some stopping conditions are met i.e. the running time has expired or all of the search space has been explored. Upon termination, the current optimal value is returned as the output.

As more levels of the space are searched, this bound drops. The biggest hurdle is that the search space at each level is still large, which means an exhaustive search will take exponential time. Although this algorithm is anytime and can produce an approximate solution within a bound of the optimal, its operations usually takes longer due to the exhaustive search needed at each level of the graph. Given that, with any characteristic function game (Ag, v) no less that 2^{n-1} coalition structure needs to be evaluated to obtain a bound an initial bound. Thus this algorithm only guarantees to return an optimal coalition structure after all the coalition structure values has been computed (that is in the worse-case it may end up searching the entire space).

This may not be practical for when the value of n i.e. the number of agent is large. It can also be argued that the bounds obtained by this algorithm may not be substantial enough to warrant the amount of computation required. For example, if $n = 12$, then to attain a bound of $\frac{n}{4}$ the algorithm needs to search levels $L_1, L_2, L_{12}, L_{11}, L_{10}$ and L_9 of the coalition structure graph. It can be argued that, the number of structures that needs to be explored in these levels is too large to for obtaining this relatively small bound.

Following the concepts proposed above, Dang & Jennings (2004) proposed an improved anytime algorithm. Their anytime algorithm performs the first two steps i.e. the value of the structure at level L_1 then performs an exhaustive search of level L_2. However the remaining search spaces are searched by computing the value of a particular structure as opposed to the exhaustive search for Sandholm et al. (1999) algorithm. This means that after computing all the values of coalition structures of level L_1 and of level L_2, instead of continuously and reviewing all the coalition structures of level L_n to L_3 it computes the value of particular coalition structures instead. The set of structures chosen can be a size that is chosen by an arbitrary decision, based on random choice, rather for any particular reason. When a new structure is found with a value that is bigger that the current optimal, the structure and its value is kept in memory as the new optimal. It then improves on this estimate by exploring more of the search space calculating a new set of structures of different sizes. Whatever

structure that is in memory will be returned as the optimal structure when the algorithm terminates.

Rahwan et al. (2007) extended Dang & Jennings (2004) work to produce an improved near-optimal anytime coalition structure generation algorithm. By examining only a minute portion of the space (approx.$3 \times 2^{n-1}$) it was still possible to come up with near optimal solution. This method employed a novel representation of the search space according to configurations of coalition structures, thus allowing it to cycle through the list of coalition structures by avoiding redundant coalition structures. This method was proven to use 33.3% less memory that Sandholm's.

3.2.2 Integer Partition-Based Search

Rahwan et al. (2007) developed an algorithm that uses integer programming (IP) with anytime property that utilizes pre-processing techniques. They represent the search space by partitioning it into sub-spaces according integer partitions so that an upper and lower bound on the best coalition structures found in them can be computed. These bounds are used to determine which sub-space have the least chance of containing the optimal coalition structure and can thus be pruned. With useless sub-spaces eliminated, search is focussed on the remaining sub-spaces. An exhaustive search is avoided by using a branch and bound technique. Like other anytime algorithms, the worst case situation can still require searching the entire space which results in time complexity $O(n^n)$. The following is a more detailed description of this method.

Representing the search space of possible coalition structures as partitions of smaller, disjoint sub-spaces allows their independent exploration. In this representation, coalition structures are categorised into sub-spaces based on the size of the coalitions they contain instead of the number of coalitions (as represented in a coalition structure graph). The advantage with this representation is the average values of the coalition structures within each sub-space can be computed easily. This allows an upper and lower bound to be calculated based on the value of the best coalition structure that is found in each sub-space. Comparing these bounds makes it possible to identify those sub-spaces that have a higher potential of containing an optimal coalition structure while those that cannot contain a solution can be pruned from the search to

avoid redundant explorations. Branch-and-bound is then applied to the remaining sub-spaces to further reduce the amount of search.

The algorithm starts by partitioning the space into sub-spaces that contain coalition structures based on the integer partition of the number of agents. An integer partition of n represents a multiset of positive integers that amount to the exactly the value of n. For example, in a system of four agents ($n = 4$), there are five partitions representing the sub-spaces: [4], [3,1], [2,2], [2,1,1] and [1,1,1,1].

An integer partition can be visualised as a sub-space containing coalition structures as shown in Figure 3.5. The sub-spaces are categorised into levels according to their integer partitions. Level P_i corresponds to an integer partition with i parts, for example level P_2 corresponds to integer partitions with two parts such as [3, 1] and [2, 2]. This allows the partitioning of the space into smaller, disjoint sub-spaces to be independently explored to look for optimal solutions. Once the space has been partitioned, IP proceeds with the following steps:

1) Scan the input and compute the bounds for every sub-space and at the same time:
   i) Identify the best coalition structures within each subspace by:
      a) calculating the upper bound of the subspace
      b) computing the lower bound of the subspace
   ii) Prune the remaining sub-spaces based on their upper bounds by:
      a) excluding subspaces whose upper bounds are lower than the lower bounds in the other subspaces.
   iii) Set a worse-case bound on the quality of the best-known solution (found so far)

2) Search within the remaining sub-spaces (after pruning). Pruning allows the search to be more focussed, and, in particular, it:
   i) avoids making unnecessary comparison to subspaces with lower value upper bound thus unlikely to contain the optimal.
   ii) avoids calculating the value of the same coalition structure repeatedly.
   iii) enables the application branch-and-bound to reduce the search time even further.

The remaining sub-spaces to be searched after scanning the input are those that have been pruned where the upper bound is lower that best one found so far. This pruning process is repeated with the remaining sub-spaces until the condition to terminate is reached. This condition is:

1) the best coalition structure found so far fits within the best bounds found, and

2) all sub-spaces remaining has either been searched or pruned.

Depending on the CFG, IP may be able to quickly find an optimal solution.

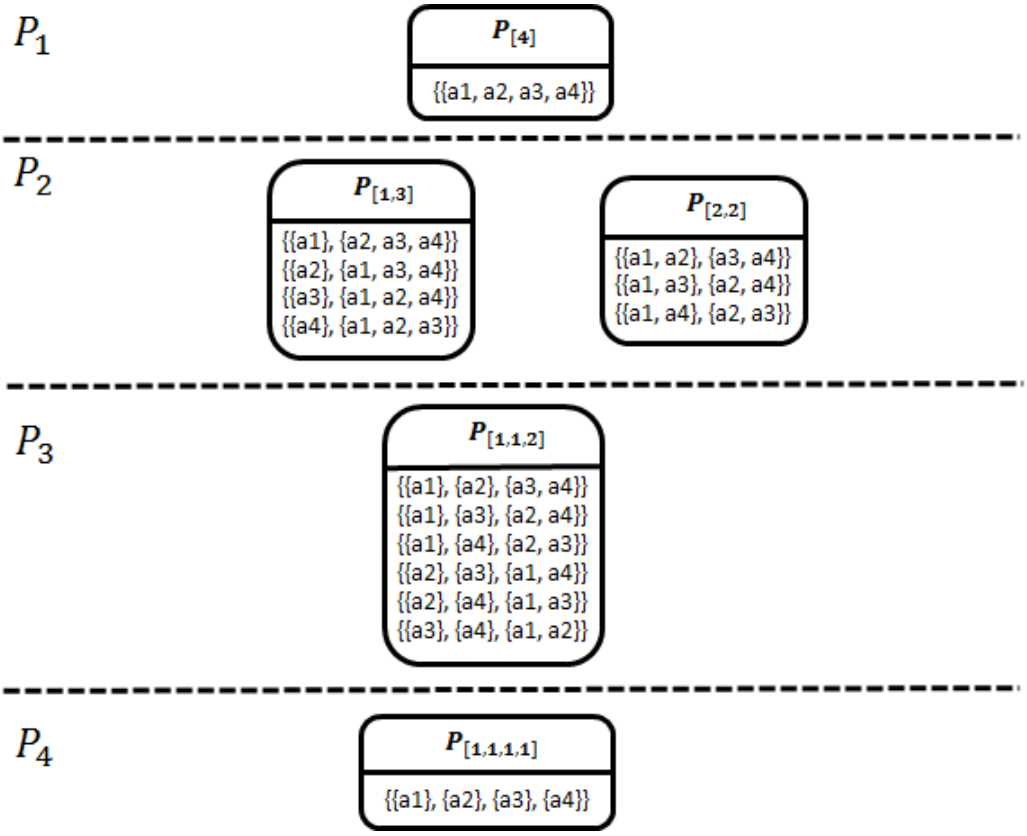

Figure 3.5 – An example IP search space and sub-spaces given 4 agents.

The method proposed by Sandholm et al. (1999) and that of Rahwan et al. (2007) both have advantages and disadvantages, the highlights are shown in Table 3.3

Table 3.3 – Comparison of the Anytime Algorithms.

Method	Advantages	Disadvantages
Coalition Structure Graph Search	• Anytime – able to return good quality solutions if stopped before completion.	• Requires searching at least the two bottom level of the coalition structures graph to establish initial bound bounds. • Requires exhaustive search at each level to improve bound. • In the worst case the algorithm ends up searching the entire space.
Integer Partition-based (IP) Search	• Anytime – able to return good quality solutions if stopped before completion. • Better pruning by removing non-promising integer partitions based on bound.	• In the worst case the algorithm ends up searching the entire space.

In the next section, we will be looking at heuristic algorithms.

3.3 Heuristic Algorithms

Heuristic methods spend little computational effort to find solutions that are not necessarily optimal but are close to optimal (Blumenfeld, 2009). In general, it is difficult to guarantee that a method produces an optimal solution. However they provide solutions that are adequate for practical use. Their main aim is to find acceptable solutions as quickly as possible. This section reviews some of the existing heuristic methods for solving the coalition structure generation problem for CFGs.

3.3.1 Genetic Algorithms

Evolutionary computation is a type of technique used in computational and artificial intelligence to solve continuous optimization and combinatorial optimization problems. It is a form of heuristic or stochastic optimization (Fogel, 1998). In general, such methods iteratively progress through the generations in the evolution of a population of individuals. A population is selected using a randomized guided search, sometimes with parallel processing, to simulate the survival of the fittest individuals (Jones, 2002). A population for the next generation is created through recombination, mutation, and selection of relevant individuals (Goldberg, 1989).

Most of the genetic operators are inspired by biological structures similar to those found in natural evolution. Genetic algorithms (GA) have been found to be effective in solving function optimization and combinatorial optimization problems that are NP-complete (Dasgupta & Michalewicz, 1997), as long as there is some regularity in the search space.

One of the earliest heuristic techniques to solve the coalition structure generation based on this approach was proposed by Sen and Dutta (Sen & Dutta, 2000). This is a form of stochastic search process to discover the optimal coalition structure. The algorithm employs an order-based genetic algorithm (OBGA). The advantages of OBGA are its scalability to larger problem sizes (Sen & Dutta, 2000). The drawback of OBGA is it has no performance guarantees.

The OBGA algorithm starts by randomly generating an initial population of coalition structures. This is followed by a repetition a set of three steps; evaluation, selection and recombination (Deb, 2001). First, every member (i.e., a coalition structure) of the current population is evaluated. In the selection phase, individuals are chosen based on their fitness. These form the next generation. The final step is recombination, in which new members are constructed by revising and swapping combination of individuals (Goldberg, 1989). The search covers several levels of the coalition structure graph (see Figure 3.5) at the same time. This means, the next set of coalition structures that will be evaluated are constructed based on the current set in the population.

Regarding crossover, the standard position based uniform, single, and two-point crossover operators result in unacceptable coalition structures. Unacceptability arises from multiple occurrences of the same agent in a structure. To overcome this, the *edge recombination* (Starkweather et al., 1991) crossover and mutation operators were used to ensure that each agent is only represented once in every coalition structure. This is achieved through the order preservation capability of the operator which produces a coalition structure with all the agents but without any duplication.

Results of experiments by Sen & Dutta (2000) showed that there was negligible performance differences across the coalition sizes. This is due to regularity in the initial population. For example, in a set of agents with a group size of n, coalition structures whose sizes are biggest and smallest will be poorly evaluated allowing the algorithm to quickly find the groups with the optimal size. Once the coalition structures with the right size is found more resources can be devoted to finding the optimal composition.

3.3.2 Simulated Annealing

Simulated Annealing, a stochastic local search method (Kirkpatrick et al., 1983), has also been applied to solve the coalition structure generation problem. (Keinänen & Keinänen, 2008). This method combines neighbourhood search with simulated annealing. The algorithm works by first setting an initial temperature, an initial value which is reduced by a factor of alpha α where $0 < \alpha < 1$. At the start an initial random solution is generated. It then continuously loops until a certain termination condition is met. This condition means that the system has cooled sufficiently i.e. when a minimum temperature value is used as the exit parameter or the maximum number of iterations has been reached and an approximately good solution has been found. The intermediate step is to decide whether the current solution is better than the initial solution. If it is better, then move to set the new solution as the current solution. Otherwise, it is accepted with a certain acceptance probability. Once this is decided, the algorithm will move to decrease the temperature and continue looping.

In more detail, an initial random coalition structure CS is generated. Another random coalition structure CS' is then generated inside the neighbourhood of CS. If CS' is superior to CS, then CS' is the new *CS*. If not, *CS'* is *CS* by calculating the acceptance probability using the

formula $e^{\frac{v(cs')-v(cs)}{\tau}}$. The parameter τ is a *temperature* parameter and e, Euler's number which has the value of 2.71828. τ decreases in value after each iteration. The decrease in value follows a defined *annealing schedule* $\tau = \alpha\tau$. Tests conducted using this algorithm demonstrated that for certain neighbourhoods with appropriate characteristics, a good enough solution can be found within a short runtime.

3.3.3 Greedy-Based Methods

Di Mauro et al. (2010) proposed a new algorithm called GRASP-CSG. This method combines two methods. One is the greedy method called Greedy Randomised Adaptive Search Procedure (GRASP), a heuristics approach first proposed by Feo & Resende (1995). The other method is a stochastic local search method. It involves greedily constructing a candidate solution (coalition structure), and then conducting local search to improve upon the solution.

The GRASP-CSG algorithm uses a randomised constructive search procedure to compute a solution and subsequently performs a local search on this solution. The search is conducted to further improve upon the solution provided from the greedy construction. A new candidate solution is constructed by adding a randomly selected solution component iteratively based on a uniform distribution. The selection comes from a special set known as the *restricted candidate list* or RCL consisting of a number of best ranked solutions. High value components i.e. some coalitions with high value from the coalition structures stored in this list are randomly selected to form a new coalition structure.

Once an RCL is constructed, a local search is done. If a neighbour is found to have a higher value than any coalition structure in the RCL, it will be added to the RCL. In the next iteration, the components from the newly added structure are chosen when a new candidate solution is constructed. It was assumed by Di Mauro et al. (2010) that, over time (as the algorithm runs for a higher number of iterations), with more high value structures added to the RCL, the choice of selection for high value components (member coalitions of the structures in the list) will lead to the construction of higher value solutions.

3.3.4 Particle Swarm Optimisation

An algorithm for coalition structure generation based on swarm intelligence (Kennedy & Eberhart, 1995) was proposed by Guo & Wang (2006). The particle swarm optimisation (PSO) algorithm is designed to find the best subsets of the search space with high quality solution candidates. This is achieved by interaction among particles that represents agents in coalitions. Particle group together as swarms. PSO works by initializing a random group of particles, i.e., solutions. From this group, it searches for the best solution iteratively. At each iteration, every particle is assigned two values **Pbest** and **gbest**. **Pbest** is a private best which denotes the best fitness value (coalition structure value) of every particle (solutions) in the population based on the initial performed search. The variable **gbest** represents the global best value obtained thus far by any particle in the population. While going through each iteration, the individual particles make continuous velocity which determines their rate of exploration changes in the solution space to figure out the best direction and distance to cover the areas which **Pbest** and **gbest** points to.

Compared to GAs, PSO has no genetic operations such as crossovers and mutation which are not relevant to the way PSO works. Instead, it focuses on changing the speed of exploration for each particle in the search space. PSO has fewer parameters to be adjusted and can thus be easier to optimise. It shares information in a completely unique way. As opposed to the approach taken by GA where chromosomes share information with each other resulting in the whole swarm moving to the best area in a concurrent path, with PSO information moves in one direction where only **gbest** transfers information to other particles making the subsequent iterations follow the current best fitness. In the simulations conducted by Guo & Wang (2006), PSO was shown to converge to a solution and obtain the best fitness much quicker than GA using information from the history of previous iterations.

The advantage of this method includes execution speed through parallel processing, robustness and ease of managing the search. This is suitable for application in coalition formation in a MAS where the number of agents is larger due to its scalable characteristics (Dos Santos & Bazzan, 2012).

3.3.5 Ant Colony Optimisation

Ant Colony optimisation (ACO) first introduced by Dorigo et al. (1996) is a stochastic metaheuristics that mimics the behaviour of ants walking along a path looking for sources of food. The main idea behind ACO is based on the idea of artificial agents behaving as ants. To apply ACO, a suitable representation of the problem in the form of a weighted graph is needed as the movements of the ants is akin to finding the best path over this graph. The ants build new solutions by exploring the paths of the graph and using a pheromone model to keep track of promising paths. The pheromone can either be nodes or edges in the graph.

A variation of the ACO method was formulated for the formation of buyer groups in electronic marketplaces. The goal is to form groups of buyers (coalitions) whose utilities will maximise the value of the entire coalition structure (Sukstrienwong, 2011). The authors devised an algorithm called BCF_2ACO (BuyerCoalitionFormation_2AntColonyOptimisation). They applied the concept of coalition structure for fairer distribution of utility to buyer coalitions. In their research the values of coalitions are not known priori, i.e. each coalition needs to be constructed and its value calculated before a coalition structure can be built. To do this, their approach deploys two ant colonies. The first one finds the best way to form buyer coalitions according to the total reward received from the sellers, i.e., the utility gained by each buyer by finding the best disjoint subsets of all buyers. Once the buyer coalitions have been formed, a second ant colony then finds the best way to partition those buyer coalitions into coalition structures that are able to achieve the highest total.

Although like other heuristics, ACO is not guaranteed to return the optimal solution. However, BCF_2ACO was shown to perform better compared to the GA method proposed by Hyodo et al. (2003) for forming buyer coalitions (Sukstrienwong, 2011).

Each of the heuristics reviewed have their own strength and weaknesses. Table 3.4 provides a summary of the advantages and disadvantages of the heuristic methods reviewed.

Table 3.4 – A Summary the Heuristic Algorithms.

Method	Advantages	Disadvantages
Genetic Algorithm	• Scalable to larger problems.	• No performance guarantees • Requires strict regularity in input.
Simulated Annealing	• Able to move to worst solution in the hope of escaping local optima.	• No performance guarantees • Highly depended on neighbourhood definition.
Greedy (GRASP-CSG)	• Greedily construct an initial solution then local search helps to explore neighbouring solution.	• Requires memory to store high value coalitions, candidates for greedy construction. • No performance guarantees.
Particle Swarm Optimisation	• Parallel exploration of the search space.	• No performance guarantees.
Ant Colony Optimisation	• History of best path taken using pheromones model	• No performance guarantees • Requires space representation with weighted graphs.

3.4 Chapter Summary

This chapter provides a review of existing algorithms for generating an optimal coalition structure for CFGs. These algorithms were divided into three classes: design-to-time, anytime, and heuristic. Each of these classes of algorithms has its own strengths and weaknesses. The fastest design-to-time algorithms for CFGs have time complexity $O(3^n)$ (Rahwan & Jennings, 2008a; Yeh, 1986). Although they generate an exact solution, they however must to be allowed to run to completion. This can be very time consuming, especially for large games. In contrast, anytime algorithms have the key feature that the quality of solution improves with time they can be terminated at any time. To get a good may still be time consuming. Unlike design-to-time and anytime methods, heuristic methods are designed to quickly return a good enough solution. However, in most cases, they cannot provide guarantees on the quality of solutions.

The next Chapter will focus on the literature review for coalition structure generation for partition function games i.e. games with externalities.

Chapter 4 Coalition Structure Generation for Partition Function Games: A Review of Literature

Chapter 3 focused on methods for determining optimal coalition structures for characteristic function games. In these types of games, the value of a coalition is independent of how other coalitions are formed. However, in some multiagent systems, the value of a coalition depends on the how external coalitions are formed. Such systems are modelled as games in partition function form (Thrall & Lucas, 1963).

An example where the formation of an external coalition impacts the value of an existing coalition is when a group of companies working together tries to erode the market share or competitiveness of other companies in the market. When these companies form alliances and develop a new standards that is proprietary which cannot be used by others outside the coalition, this reduces the competitiveness of other companies which does not have access to these standards. An example of this is the Very Large Scale Integration (VLSI) consortium and well as the Fifth Generation Computer Systems (FGCS) project which subject negative externalities to other companies who were not members (Yi, 2003).

Recall that, in a PFG, the value of a coalition may be influenced by the creation of another coalition in the system. Consider an example of two coalition structures *CS* and *CS'* where $CS = \{C_A, C_B, C_C\}$ and $CS' = \{C_A, C_B \cup C_C\}$. C_A Might have a different value in *CS* compared to *CS'* as result of the union between $C_B \cup C_C$. This effect is known as *externalities* and in this example, it is imposed on C_A by the merger between C_B and C_C to form coalition $C_B \cup C_C$.

Externalities are vital in multiagent systems where coalitions have either contradictory or overlapping goals. This chapter reviews the existing methods for coalition structure generation in partition function games. Despite the importance of externalities in many applications, less attention has been given to the computational aspects of games with externalities. At the time of writing, there were no existing heuristic methods for solving the CSG problem for the general classes of PFGs. This chapter reviews the existing non-heuristic methods.

The chapter is organised as follows. Section 4.1 outlines methods based on the Integer Partition-based (IP) algorithm. Section 4.2 describes a **distributed** approach. All the methods in sections 4.1 and 4.2 deal with games with either exclusively positive or exclusively negative externalities. Section 4.3 describes a method which considers **mixed externalities** where both externalities exists in the same game.

4.1 Integer Partition-based Algorithm for PFG

One of the early works that considered coalition structure generation in games with externalities was by Michalak et al. (2008). They used the same representation as Rahwan et al. (2007) for dividing the space into integer partitions and calculating the bounds for each coalition structure size. This is represented by the partition of integers. In an example for 4-agents, the integer partition [1,1,2] represents a coalition structure with two coalitions of size 1 and one coalition of size two i.e. $CS = \{\{a1\}, \{a2\}, \{a3, a4\}\}$. Each integer partition is known as a **configuration**.

Initially, their research considered four types of PFG settings:

- those that are super-additive and have positive externalities (PF_{sup}^{+}),
- those that are super-additive and have negative externalities (PF_{sup}^{-}),
- those that are sub-additive and have positive externalities (PF_{sub}^{+}) and
- those that are sub-additive and have negative externalities (PF_{sub}^{-}).

They proved that, in each of these four PFG settings it was possible to bound the values of every coalition. Of the four games considered above, they showed that for PF_{sup}^{+}, the optimal CS is always the grand coalition while for (PF_{sub}^{-}) the optimal will always be the coalition structure of singletons. This would make these two settings trivial when studying the CSG problem as the computing of the optimal structure is straightforward. Therefore, they only ran simulations and tested their algorithm on the PF_{sup}^{-} and PF_{sub}^{+} settings.

The primary contribution of their research was devising a way to construct bounds on the value of every coalition for every coalition structure in a PF_{sup}^{-} or PF_{sub}^{+} games. Once the bounds value of each coalition size is known, this can be used to construct the upper and lower bound for each configuration where a configuration is the integer partition denoting the size of the coalition structure i.e. [1,1,1,1],[1,1,2],[1,3],[2,2] and [4] for a set of 4-agents. Being

able to construct bounds on the value of every coalition in games with externalities gave them the ability to adapt the state-of-the-art IP algorithm for CFG for use in a PFG setting.

Their algorithm for the PFG setting does the same kind of partitioning and pruning of the search space as Rahwan et al. (2007) algorithm for CFGs. It also uses a similar way of searching through promising subspaces. Important techniques deployed by the IP algorithm for CFGs such as avoiding redundant calculation where no CS is considered more than once is directly transferred into their adaptation of the algorithm for the PFG settings. The branch-and-bound rule however required modifications to avoid exploring hopeless routes when constructing candidate solutions for PFGs. Their simulation results found that in some cases for the PF_{sub}^{-} settings, the pruning technique used was less effective and almost all of the space needed to be searched to find the optimal. They argued that this is due to the nature of PF_{sub}^{-} setting where the value of the structure in a configuration is dependent on the value of the structures in the previous levels (see dashed line in Figure 4.1).

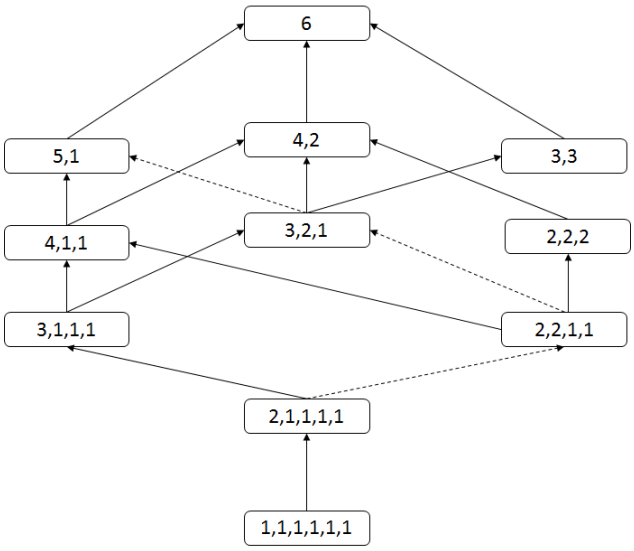

Figure 4.1 – Integer Partition for six-agents.

4.1.1 $IP^{+/-}$ Algorithm

The previous algorithm (Michalak et al., 2008) deals with two specific classes of PFGs known as PF_{sup}^{-} and PF_{sub}^{+}. In PF_{sub}^{+}, the merger between any two coalitions will either decrease their joint value or keep it constant but increases the values of the other coalitions in that structure (or keep it constant as well). That is, the values of the coalitions are weakly sub-additive meaning their value can increase because the externalities are positive or it can remain the

same (externalities equal to zero). In PF_{sup}^{-} The merger between any two coalitions will either increase their joint value or keep it constant but decrease the values of the other coalitions in that structure (or keep it constant). This means that the values of the coalitions are weakly super-additive, thus their value can decrease because the externalities are negative or it can remain the same (where externalities are equal to zero).

PF_{sup}^{-} and PF_{sub}^{+} are widely used in economics and social sciences to model relationships between coalitions which is usually the main focus of the problem studied. An example for PF_{sub}^{+} is in the coordination of environmental policy between countries. When countries decide to join an international coalition to reduce emissions, this may have a negative impact on their economy i.e. slower growth (a sub-additive property). However, their actions could have a positive impact on the other countries outside the coalition thus maximising the overall welfare of everyone i.e. the world. Similarly, in the pharmaceutical industry, when two companies form a coalition to develop a new drug, this can increase their market share (super-additive property) at the expense of the other players (other coalitions of companies in the market) who will likely lose market share. As the internet becomes the de facto marketplace of the future and the transactions becomes more complicated, there may be a need for the formation of ad hoc coalitions which will need to consider externalities in a more extensive manner without the limitations imposed by PF_{sup}^{-} and PF_{sub}^{+} with the assumptions of super- and sub-additivity.

This means that, these two classes (PF_{sup}^{-} and PF_{sub}^{+}) are too rigid (restricted) to be able to represent a number of settings. For example, a merger between coalitions may be beneficial when their sizes are smaller. However, if the newly created group is so large that it results in issues such as communication problems (the merger should have resulted in positive impact for the coalitions), then the cost may far outweigh the benefits. This type of scenario is neither super-additive nor sub-additive.

Due to the strict nature of the PF_{sub}^{+} and PF_{sup}^{-} settings, Rahwan et al. (2009) proposed an algorithm called $IP^{+/-}$ that deals with a wider class of PFGs. These classes of PFGs denoted PF^{+} and PF^{-} which drop the assumptions of super- and sub- additivity, are:

- PF^+ – Games where externalities are weakly positive externalities, i.e. the externalities are always positive (non-negative). Which means the merger of any two coalitions will **never** decrease the value of other coalitions in the system.
- PF^- – Games where externalities are weakly negative externalities, i.e. the externalities are always are negative (non-positive). Which means the merger of any two coalitions will **always** decrease the value of other coalitions in the system.

For these more general type of PFG games, Rahwan et al. (2009) developed a novel anytime algorithm with a new way of bounding the value of any group of coalitions in a PF^+ / PF^- game. Compared to the approach taken by Michalak et al. (2008) which bounds the value of any given coalition C, they bound the value to a given partition of C. Once the bounds for each coalition in each partition can be successfully computed, they apply the same technique as Rahwan et al. (2007) to prune any unpromising subspace (partitions). Although for the general PFG case it is not possible to prune any partition of a subset of agents, they were able to show that pruning was possible for PF^+ / PF^- under specific conditions using their own pre-processing techniques. They devised two algorithms for searching these subspaces. The first one is a pre-processing algorithm which prunes the subspaces by comparing the upper and lower bounds. The second one is $IP^{+/-}$ which is an updated version of the IP algorithm introduced in Rahwan et al. (2009).

To test their algorithm, they needed to provide a suitable way of generating input instances that comply to the PF^+ / PF^- settings. To this end, they provided an equation for generating the input and showed that the input instances generated satisfied the conditions of the PF^+ / PF^- settings. They also showed that although the function they devised consists of $O(n^n)$ values, it only requires $O(2^n)$ values to be stored in memory. This overcomes the limitations encountered in Michalak et al. (2008) where they were unable to test their algorithm for any problem size larger than 10-agents due to the need of storing the value for every pair $(C, CS) : C \in CS$. Using this function to generate the input, it was possible for them to evaluate and provide results PFGs with up to 24-agents. The initial work in (Rahwan et al., 2009) was only tested on the Uniform and Normal distributions. However, they later extended their investigation with $IP^{+/-}$ to include the NDCS distribution (Rahwan et al., 2012).

4.2 Distributed CSG with Externalities

Modern computing brings multiprocessing power to the mass market, in a typical personal computer today the central processor or CPU usually includes at least two or more processors. This allows for multiple threads to be processed and executed at the same time. It would be unwise not to take advantage of this processing power where computers that are capable or simultaneously processing 4 or 8 thread are common. The algorithms for solving the CSG problem in PFG mentioned so far are centralized i.e. single-thread sequentially executed algorithms. Perhaps this was the motivation behind the development of a distributed algorithm for solving the CSG problem in PFGs. Distributing the calculations has the benefit of splitting the processing and memory costs among the agents. Allocating resources by distributing the load so that agents with superior resources and capacity get a bigger chunk of the workload, increases overall efficiency. These and other desirable characteristics of a distributed approach were highlighted by Rahwan & Jennings (2007).

A distributed algorithm for the CSG problem in PFG was introduced by Epstein & Bazzan (2013). They presented DCSGE (Distributed Coalition Structure Generation with Externality) which builds upon the work of Michalak et al. (2010). The original D-IP algorithm introduced in Michalak et al. (2010) is an improvement on the IP algorithm (Rahwan et al., 2009) which added parallel processing and can be used for both distributed and centralized environments. However, D-IP did not take into account externalities from coalition formation.

A contribution made by Rahwan et al. (2012) provided the equation to calculate the externalities for each coalition. The main characteristics of this equation are that the externalities effecting each coalition must be of a single type, either positive only or negative only. This made it possible to set the upper and lower bounds for any coalition and also provide worst case guarantees on solution quality. Using the methods for distributing the calculation of D-IP; and the equation on calculating externalities in $IP^{+/-}$, (Epstein & Bazzan, 2013) combined certain constructs of the two methods to create DCSGE.

The DCSGE algorithm is split into two stages. Stage one involves evaluating the values of the coalitions belonging to selected coalition structures. As explained in Rahwan et al. (2009), it is possible to compute the maximum and minimum values of a coalition in for any PFG in the PF^{+} / PF^{-} settings. Thus once these are computed, the values are then communicated and shared among the agents.

At the second stage, the agents then select which configuration i.e. which integer partition it will choose to begin search for the optimal CS with the information obtained in the first stage. The information obtained in the first stage will guide the agents in the second stage to explore only promising configuration to enable the optimal to be located more quickly. As the search progresses and information on the bounds are exchanged between the agents, more parts of the space can be eliminated. The distributed exploration of multiple configurations enable more space to be covered at any given time. This, it was hoped, will allow for quicker improvements on the bound and thus convergence to the optimal. The authors claim that the distributed computation allows for faster elimination of non-promising configuration thus reducing amount of space that needed to be searched to return the optimal solution. They claim that from simulation results, in some cases just 0.01% of search space needed to be searched before an optimal is found.

4.3 PFGs with Mixed Externalities

The methods reviewed thus far in this chapter deal with PFGs where the externalities are either always positive (PF^{+}_{sub} & PF^{+}) or always negative (PF^{-}_{sup} & PF^{-}) (Epstein & Bazzan, 2013; Michalak et al., 2008; Rahwan et al., 2009; Rahwan et al., 2012). None of these methods considered PFG settings where both negative and positive externalities are present i.e. where the externalities are mixed such that some externalities are negative while others are positive in the same game. Despite being given less attention in the CSG literature, games with mixed externalities reflect many real world situations where a merger of two coalitions may positively impact one coalition but negatively impact another.

For example, A merger between to large conglomerates which will create a giant conglomerate will have a negative impact (negative externality) on smaller organisations in the market At the same time, it will benefit coalitions of stock traders as the shares of the newly merged conglomerate will most likely increase in value (positive externality). This example represents many real life scenarios where not all the players in an environment is of the same type. Thus different types of players merging into different types of coalitions will have different impact on other types of players or coalition in the same domain. Based on this concept, (Banerjee & Kraemer, 2010) introduced a framework to represent coalitional games with mixed externalities i.e. where positive and negative externalities co-exist.

They showed that this framework which models externalities on the concept of competition and complementation conforms to the previous PF^+ and PF^- setting considered in Rahwan et al. (2009) as a **special case**. For generating the input data, they used an identical approach to Rahwan et al. (2009), except when assigning values to each coalition. Whereas in Rahwan et al. (2009), the externalities are solely added or subtracted, their approach needed to consider the relationship between the type of coalition C and each coalition in $CS\backslash\{C\}$. They do this by defining two sets of coalition types. One coalition type is a potential collaborator (externalities on will be positive on C) while the other is a competitor (externalities will be negative on C). Thus the values are either added or subtracted on C depending on the type of other coalitions in the structure. This allows a similar input instance to be generated as in (Rahwan et al., 2009) which only requires $O(2^n)$ values to be stored in memory.

While in Rahwan et al. (2009) it was possible to prune the integer partition space, this is not possible when considering agents of different types as the space now changes into a type-space where each integer partition contains different types of coalition structures containing coalitions with different types of agents. Therefore, they adapted a branch and bound search algorithm to their mixed externalities setting for exploring this type-space which allows significant pruning of the search space. One of the most important findings from their simulations is that their branch and bound algorithm actually increases in performance i.e. the speed of the search is faster for a **larger** number of agents (larger problem instances). Their method however, unlike the other methods reviewed in this chapter only works in a PFG settings and is not applicable to CFGs. Another lacking feature is that their method is not anytime thus requires the search to complete to return a solution.

4.4 Chapter Summary

This chapter presents a review of existing methods for coalition structure generation for partition function games. Most of these methods are based on the IP algorithm (Epstein & Bazzan, 2013; Michalak et al., 2008; Rahwan et al., 2009; Rahwan et al., 2012). Moreover, all these methods considered games with externalities in either exclusively positive or exclusively negative form instead of games where both are present i.e. mixed externalities. The only method to consider mixed externalities is Banerjee & Kraemer (2010) which models the mixed externalities effect caused by the mixing of agent types.

Table 4.1 – Comparison of Methods for CSG in PFGs.

Method	Advantages	Disadvantages
IP for PFG (Michalak et al., 2008) Tested Up to 10-agents.	• Works for CFG and PFG. • Anytime properties of IP retained. • Able to find Optimal CS searching just 1.75% of search space at minimum.	• Centralised • Input requires value stored for every pair (C, CS) i.e. consumes more memory. • Consider games with either positive or negative externalities separately not with both externalities in the same game • Tested only on Uniform Distribution • Larger part space needs to be explored to find optimal in PF_{sup}^{-} setting.
$IP^{+/-}$ (Rahwan et al., 2009; Rahwan et al., 2012) Tested Up to 24-agents.	• Works for CFG and PFG • Anytime. • Less memory required to store input due to pre-computed externalities on each coalition value. • Tested three different probability distributions (Uniform, Normal and NDCS distributions). • Able to find Optimal CS searching just 0.0001% of search space at minimum.	• Centralised • Considers games with either positive or negative externalities exclusively not with both in one game i.e. mixed externalities. • Larger part space needs to be explored to find optimal in PF^{-} setting.
Branch and Bound for Mixed Externalities (Banerjee & Kraemer, 2010) Tested Up to 14-agents.	• Considers games with **Mixed Externalities**. • Works for both CFG and PFG • Uses similar input instance as (Rahwan et al., 2012) thus less memory required. • Able to find Optimal CS searching just 0.048% of search space at minimum.	• Centralised • Tested only on Uniform Distribution • Not Anytime.
DCSGE (Epstein & Bazzan, 2013) Tested Up to 16-agents.	• Distributed method. • Works for both CFG and PFG • Can be run as centralised or distributed search. • Uses similar input instance as (Rahwan et al., 2012) thus less memory required. • More than one distribution type considered (Uniform & Normal). • Able to find Optimal CS searching just 0.01% of search space at minimum.	• Considers games with either positive or negative externalities exclusively not with both in one game i.e. mixed externalities. • Not Anytime.

While the methods proposed in Banerjee & Kraemer (2010), Michalak et al. (2008), Rahwan et al. (2009) and Rahwan et al. (2012) are centralised, the DCSGE algorithm proposed in Epstein & Bazzan (2013) distributes the coalition structure generation process to several agents. DCSGE is a more comprehensive algorithm as it can be run in both centralised as well as distributed manner. Table 4.1 provides a comparison of the methods for solving the CSG problem in PFGs, listing the advantages and disadvantage of each method.

The performance of each method varies in terms of the minimum amount of search space explored before an optimal solution can be found. The method proposed by Rahwan et al. (2012) can find the optimal while exploring just 0.0001% of the search space for 24 agents while Michalak et al. (2008) needed to explore 1.75% of the space to find the optimal for just a 10-agent game.

It should be noted that while the methods in Michalak et al. (2008), Rahwan et al. (2009) and Rahwan et al. (2012) are **anytime**, the methods in Banerjee & Kraemer (2010) and Epstein & Bazzan (2013) are **not**. From the reviewed literature it was found that no design-to-time algorithms exist for CSG in PFG at time of writing. As for heuristic methods, Sen & Dutta (2000) explored genetic algorithms to find optimal coalition structures. Their simulations were conducted for both CFGs and PFGs, however their search space was restricted by stringent regularity unlike the general classes of PFGs considered in this thesis.

Chapter 5 Heuristic Methods for Finding Optimal Coalition Structure

This chapter gives algorithmic descriptions of the following four heuristic methods for determining an optimal coalition structure:

1. A tabu search method we call TACOS (**TA**bu Search for **CO**alition **S**tructure).
2. A simulated annealing method.
3. An ant colony optimization method
4. A particle swarm optimization method.

These four methods form the focus of this research.

These methods were chosen due to their suitability for adaptation to use neighbourhood operators to explore the space. This allows for the same neighbourhood generation operator to be used for all the methods.

5.1 Tabu Search for Coalition Structure Generation (TACOS)

Tabu search (TS), first proposed by Glover & Laguna (1997), is a general purpose heuristic that uses of neighbourhood search combined with tabu memory to quickly return high quality solutions. The tabu memory is used to store a list of coalition structures (points in the search space) already visited by the search. This avoids the same points and points known to be inferior in the tabu list from being revisited again as the search progresses. Avoiding already visited solutions ensures that the search continues from unexplored regions of the search space. It also avoids being stuck at a local maxima enabling better solutions to be found as the search continues.

TS is a highly adaptable method that can be combined with problem-specific heuristics. In some cases, TS has been shown to find solutions very close to optimal (Crainic et al., 2009; Lin et al., 2014; Lodi et al., 2004). It has also been used effectively for solving combinatorial optimization problems (Chang et al., 1999; Rolland et al., 1996).

Tabu search is an improvement on a typical Local Search (LS) in that it enables LS to overcome local optima. The basic principle of tabu search is to perform local search in the neighbourhood of a local optimum. Useful information gained during local search is maintained in a memory called tabu list. This information is used to guide the search toward potentially good directions and prevent visiting a point repeatedly. In other words, this list records the history of search. A typical Tabu search begins by moving to some local optima. To avoid redundancy in making the same move again, moves are stored in one or more tabu lists.

Using the general framework of TS, TACOS was devised to work as follows. To begin, a random coalition structure is generated as the initial starting point (see Algorithm 1). This start point is set as current best solution and the coalition structure is added as the initial entry to the tabu list. The tabu list is an array that keeps those coalition structures that are forbidden from being revisited again in subsequent iterations of the search.

A neighbourhood for the starting coalition structure is generated using a set of *neighbourhood operators* (see section 5.1.1 for details). Three operators are used for neighbourhood generation:

i. *Merge*
ii. *Split*
iii. *Shift*

Within the neighbourhood generated by these operators, a local maximum and minimum is identified. The local minimum is added to the tabu list to prevent future iterations from visiting the coalition structure again. A choice was made to only add the worst solution instead of adding all visited solutions as this would have contributed to a larger tabu list and consumed more memory potentially slow down the search. If a local maxima is found to be better than the current best, the maxima becomes the current best.

Algorithm 1 TACOS: *Tabu* search algorithm for finding an optimal coalition structure

```
1:  tabuList ← [ ] {Tabu list initially empty}
2:  CS ← randomly generated CS; {Generate a random start point}
3:  currentBest ← CS
4:  for iterationCount ← 1, maxIterations do
5:      Generate a neighbourhood N of CS with chosen operators
6:      Find the bestCS and the worstCS in N
7:      if worstCS not in tabuList then
8:          Add worstCS to tabuList
9:      end if
10:     if v(bestCS) > v(currentBest) then
11:         currentBest ← bestCS
12:     end if
13:     if bestCS not in tabuList then
14:         Add bestCS to tabuList
15:         CS ← bestCS
16:     else
17:         CS ← A randomly generated coalition structure that is not in tabuList
18:     end if
19: end for
20: Return currentBest
```

At any iteration during the search where a local maximum is not in the tabu list, it will be added and the search proceeds from this point. Otherwise, if it is already in the tabu list, this means that this point and its neighbourhood have already been explored, thus will not be visited again. When this happens, a random point is generated repeatedly until a point that is not in the tabu list is found and search continues from this newly generated coalition structure. This cycle continues for a fixed number of iterations, and once finished the best solution found in returned as the optimal.

5.1.1 Neighbourhood Generation Operators

As is the case with any general purpose heuristic method, a suitable adaptation to the problem domain is needed. Tabu search requires a suitable neighbourhood generation operator that is problem specific to be defined so that an effective exploration of the search space is possible. One of the key factors that influences performance is the choice of neighbourhood operators. When defining neighbourhood operators, a key consideration is good coverage of the search space. Well-designed neighbourhood operators contribute heavily to the success of tabu search and with this in mind, the three operators, i.e., merge, split, shift were designed as follows.

Merge Operator

The merge operator generates neighbours of a coalition structure by merging two coalitions in the structure. The $immidiate\ neighbour\ i_m(CS)$ of a coalition structure CS if CS is the grand coalition is the grand coalition. Otherwise, $i_m(CS)$ is the coalition structure generated by merging the first two coalitions in CS. The merging is done repeatedly to generate a set of structures that form the neighbourhood N_m of CS. Thus, the neighbourhood is defined as:

$$N_m(CS) = i_m(CS) \cup i_m\big(i_m(CS)\big) \cup \ldots \cup i_m(GrandCoalition)$$

Merge operation is performed until the merger of two coalitions results in the grand coalition where no further merges can be made.

Consider the following example. Let CS = {{a1}, {a2, a3, a4, a6, a8, a12}, {a5}, {a7, a9, a11}, {a10}}.

Neighbouring CS generated using the MERGE operator	
Neighbour 1	{{a1, a2, a3, a4, a6, a8, a12}, {a5}, {a7, a9, a11}, {a10}}
Neighbour 2	{{a1, a2, a3, a4, a5, a6, a8, a12}, {a7, a9, a11}, {a10}}
Neighbour 3	{{a1, a2, a3, a4, a5, a6, a7, a8, a9, a11, a12}, {a10}}
Neighbour 4	{{a1, a2, a3, a4, a5, a6, a7, a8, a9, a10, a11, a12}}

In the structure CS = {{a1}, {a2, a3, a4, a6, a8, a12}, {a5}, {a7, a9, a11}, {a10}}, the first two coalitions are {a1} and {a2, a3, a4, a6, a8, a12}, thus the first neighbour Neighbour 1 results from merging {a1} and {a2, a3, a4, a6, a8, a12} into {a1, a2, a3, a4, a6, a8, a12} resulting in the neighbour Neighbour 1 = {{a1, a2, a3, a4, a6, a8, a12}, {a5}, {a7, a9, a11}, {a10}}. Then, Neighbour 2 is obtained from Neighbour 1 by merging the first two coalitions in Neighbour 1. This process repeats until Neighbour 4, which is grand coalition, is generated. No further merges are possible. The neighbourhood is therefore comprised of four structures Neighbour 1 to Neighbour 4.

Split Operator

The split operator generates neighbours by splitting a coalition in the structure. The $immidiate\ neighbour\ i_{sp}(CS)$ of a coalition structure CS is a CS whose member coalitions consists of singletons. Otherwise, $i_{sp}(CS)$ is the coalition structure generated by splitting the

largest coalition in CS in the middle to generate two coalitions of the same size. If the largest coalition has an even number of agents, it will be split in the middle into two coalitions of equal size. If the largest coalition has an odd number of agents, it will be split in the middle into two coalitions with one coalition containing one more agent than the other. To generate a set of neighbours, splitting is repeated until no further splits are possible, i.e., when a structure is comprised only of singletons. The neighbourhood N_{sp} of a structure CS is defined as:

$$N_{sp}(CS) = i_{sp}(CS) \cup i_{sp}\left(i_{sp}(CS)\right) \cup \ldots . \cup i_{sp}(AllSingletons)$$

Consider the example coalition structure CS = {{a1}, {a2, a3, a4, a6, a8, a12}, {a5}, {a7, a9, a11}, {a10}}.

Neighbouring CS generated using the SPLIT operator	
Neighbour 1	{{a1}, {a2, a3, a4}, {a5}, {a6, a8, a12}, {a7, a9, a11}, {a10}}
Neighbour 2	{{a1}, {a2}, {a3, a4}, {a5}, {a6, a8, a12}, {a7, a9, a11}, {a10}}
Neighbour 3	{{a1}, {a2}, {a3, a4}, {a5}, {a6}, {a7, a9, a11}, {a8, a12}, {a10}}
Neighbour 4	{{a1}, {a2}, {a3, a4}, {a5}, {a6}, {a7}, {a8, a12}, {a9, a11}, {a10}}
Neighbour 5	{{a1}, {a2}, {a3, a4}, {a5}, {a6}, {a7}, {a8, a12}, {a9}, {a10}, {a11}}
Neighbour 6	{{a1}, {a2}, {a3, a4}, {a5}, {a6}, {a7}, {a8}, {a9}, {a10}, {a11}, {a12}}
Neighbour 7	{{a1}, {a2}, {a3}, {a4}, {a5}, {a6}, {a7}, {a8}, {a9}, {a10}, {a11}, {a12}}

The largest coalition in CS is {a2, a3, a4, a6, a8, a12}, so the first neighbour is generated by splitting of {a2, a3, a4, a6, a8, a12} into {a2, a3, a4} and {a6, a8, a12} resulting in the neighbouring Neighbour 1 = {{a1}, {a2, a3, a4}, {a5}, {a6, a8, a12}, {a7, a9, a11}, {a10}}. In the structure {{a1}, {a2, a3, a4}, {a5}, {a6, a8, a12}, {a7, a9, a11}, {a10}} where there are 3 largest coalitions. The first largest coalition, in this case {a2, a3, a4}, will be split. Because it is of odd size 3, it is split into {a2} and {a3, a4}. The resulting neighbouring Neighbour 2 = {{a1}, {a2}, {a3, a4}, {a5}, {a6, a8, a12}, {a7, a9, a11}, {a10}} and this process is repeated until no further splits are possible. Seven neighbours Neighbour 1 to Neighbour 7 are thus generated.

Shift Operator

The shift operator generates neighbours by moving an agent from one coalition to another. The $immidiate\ neighbour\ i_{sh}(CS)$ of a coalition structure CS is CS if it is the grand coalition. Otherwise, $i_{sh}(CS)$ is the coalition structure generated by moving an agent from

one coalition to another coalition in the structure. For moving, each individual agent is considered one by one. For each individual agent within a structure, a neighbour is generated by moving it to an external coalition. Thus for an agent and a structure, |*CS*| - *1* neighbours are generated. This is repeated for each individual agent. The neighbourhood N_{sh} of CS defined as:

$$N_{sh}(CS) = i_{sh}(CS) \cup i_{sh}\big(i_{sh}(CS)\big) \cup \ldots \cup i_{sh}(GrandCoalition)$$

Consider the example with CS = {{a1}, {a2, a3, a4, a6, a8, a12}, {a5}, {a7, a9, a11}, {a10}} and the movement of agents 1 and 2.

Neighbouring CS generated using the SHIFT operator on Agent 1	
Neighbour 1	{{a1, a2, a3, a4, a6, a8, a12}, {a5}, {a7, a9, a11}, {a10}}
Neighbour 2	{{a1, a5}, {a2, a3, a4, a6, a8, a12}, {a7, a9, a11}, {a10}}
Neighbour 3	{{a1, a7, a9, a11}, {a2, a3, a4, a6, a8, a12}, {a5}, {a10}}
Neighbour 4	{{a1, a10}, {a2, a3, a4, a6, a8, a12}, {a5}, {a7, a9, a11}}
Neighbouring CS generated using the SHIFT operator on Agent 2	
Neighbour 1	{{a1, a2}, {a3, a4, a6, a8, a12}, {a5}, {a7, a9, a11}, {a10}}
Neighbour 2	{{a1}, {a2, a5}, {a3, a4, a6, a8, a12}, {a7, a9, a11}, {a10}}
Neighbour 3	{{a1}, {a2, a7, a9, a11}, {a3, a4, a6, a8, a12}, {a5}, {a10}}
Neighbour 4	{{a1}, {a2, a10}, {a3, a4, a6, a8, a12}, {a5}, {a7, a9, a11}}

In the example, CS = {{a1}, {a2, a3, a4, a6, a8, a12}, {a5}, {a7, a9, a11}, {a10}} has five member coalitions. The agent to be shifted cannot shift into its own coalition. Thus, there are $|CS| - 1$ possible shifts. In this example with $|CS| = 5$, the maximum number of shift operations on an agent is 4. This operation is repeated for the remaining agents in the structure until all agents in the shift neighbourhood are generated.

Consider another example where $CS = AllSingletons$. Each agent generates $|CS| - 1$ neigbours. For 27-agents, the shifting of an agent into another coalition generates 26 neighbouring coalition structures. In total, the neighbourhood consists of 702 coalition structures.

5.2 Simulated Annealing for Optimal Coalition Structure

Simulated annealing (SA) is an enhancement of the randomised local search method. SA is analogous to the metropolis algorithm (Metropolis et al., 1953). This method iteratively

generates random solutions similar to gradient descent. The main differentiating factor between SA and standard gradient descent is the provision for accepting inferior solutions with some non-zero probability. This enables escaping local optima and enabling a more extensive search for optimal solution.

The SA method devised in this thesis for solving the CSG problem is given in Algorithm 2. The search begins by constructing a random coalition structure CS as the starting point.

Algorithm 2 SA: Simulated Annealing search algorithm for finding an optimal coalition structure

1: $CS \leftarrow$ randomly generated CS; {Generate a random start point}
2: $bestCS \leftarrow CS$
3: $InitialTemperature \leftarrow 1.0$ {Initialize temperature}
4: $\alpha \leftarrow 0.99$ {Initialize α used to update temperature}
5: **for** $iterationCount \leftarrow 1,\ maxIterations$ **do**
6: Generate a neighbourhood N of CS with shift, merge, split
7: $CS' \leftarrow$ A structure from N chosen uniformly at random
8: **if** $v(CS') \geq v(CS)$ **then**
9: $CS \leftarrow CS'$ {Update CS}
10: **else**
11: $Probabilty \leftarrow e^{\frac{v(CS')-v(CS)}{t}}$ {Set probability of accepting an inferior solution}
12: $CS \leftarrow CS'$ {Update CS}
13: **end if**
14: **if** $v(CS') \geq v(bestCS)$ **then**
15: $bestCS \leftarrow CS$ {Update $bestCS$}
16: **end if**
17: $t \leftarrow t \times \alpha$ {Update t}What is alpha? Where is this defined?
18: **end for**
19: Return $bestCS$

At each iteration a neighbourhood N of CS is generated. From this neighbourhood, CS' is a random structure chosen uniformly at random. If the value of CS' is better than the value of CS, CS' becomes CS. Otherwise, with an acceptance probability of $e^{\frac{v(CS')-v(CS)}{t}}$, CS' becomes CS. If the value of CS is better than $bestCS$, then $bestCS \leftarrow CS$. At the end of each iteration, the temperature is updated. This is repeated for a fixed number of iterations, with the best solution found during the course of the search being returned as the optimal solution.

5.3 Ant Colony Search for Optimal Coalition Structure

Ant colony algorithms (Dorigo et al., 1996) are methods which mimic the movement of ants along a path between a food source and their colony. This method was used in Dorigo et al. (1996) to explore optimal paths in a graph. Ants move around a space searching for food source and laying pheromones along the way. The pheromones help other ants in the colony to identify which paths are best when searching for food. The stronger the pheromone trail, the more likely the other ants will follow this path.

For the CSG problem that we consider, each of the three neighbourhood operators (defined in Section 5.1) is associated with a pheromone level which is a number between 0 and 1. The higher the number, the stronger the pheromone. The pheromone levels are initially assigned the following values:

1. Pheromone for shift (Ph_{sh}) is 0.1.
2. Pheromone for merge (Ph_{m}) is 0.1.
3. Pheromone for split (Ph_{sp}) is 0.1.

To begin, a virtual ant generates an initial random coalition structure CS and conveys this to three other virtual ants. These three ants stand for the three neighbourhood operators shift, merge, and split. The ant that corresponds to shift operator generates a random neighbour (called neighbour 1) of CS using shift. Likewise, the ant that corresponds to merge (split) operator generates a random neighbour of CS using merge (split). The neighbours of CS generated by merge are called neighbour 2 and those generated by split are called neighbour 3.

Let $CurBest$ be the best of neighbour 1, neighbour 2, and neighbour 3. If $CurBest$ is better than CS, then the complete neighbourhood of $CurBest$ is generated using the same neighbourhood operator that generated $CurBest$ from CS. If the best candidate, say X, in this neighbourhood is better than $CurBest$, then CS is assigned X and the next iteration begins.

On the other hand, if CS is better than $CurBest$, then the best neighbourhood operator is calculated based on the strengths of each of the three neighbourhood operators. The calculation of operator strength is described in the following box.

Strength of a neighbourhood operator:

OS_{sh} : Denote the strength of shift operator.

OS_m : Denotes the strength of merge operator.

OS_{sp} : Denotes the strength of split operator.

The strength of an operator is defined in terms of three factors:

1. The difference in the value of CS and the value of the neighbour generated by the operator.
2. The pheromone level of the operator.
3. The cost of the operator.

A cost is assigned to each operator as follows:

1. Cost of shift Operator (C_{sh}) – It is assigned the lowest value among the three.
2. Cost of merge Operator (C_m) – It is assigned a value between the highest and lowest.
3. Cost of split Operator (C_{sp}) – It is assigned a higher value among the three.

Then the strength of shift operator OS_{sh} is calculated as follows:

$$OS_{sh} = \frac{(v(CS) - v(CS')) \times \mathrm{P_{sh}}}{\mathrm{C_{sh}}} \qquad (5)$$

OS_m and OS_{sp} are calculated analogously.

A complete neighbourhood of CS is generated using the highest strength neighbourhood operator. If the best candidate, say X, in this neighbourhood is better than CS, then CS is assigned X and the next iteration begins. Otherwise, although the value of $CurBest$ is lower it will become CS and the next iteration begins (see Algorithm 3).

Pheromone Update

In order to reinforce positive feedback from pheromones and also to avoid getting caught in local optima, the pheromone levels are updated as follows. At the beginning of each iteration, once the neighbours are generated, the pheromone level corresponding to the operator that generated the highest value neighbour is increased by a random value drawn from a uniform distributed between 0 and 1. The more frequently an operator finds the highest quality solution, the higher the pheromone level for that operator.

Then, to avoid getting stuck in a local optima, the pheromone evaporation schedule is set as follows. At every iteration, the value of the pheromones is decreased at the rate of 25%. Thus the pheromone values fluctuate throughout the search. This avoids being stuck in a local optima. Below, we illustrate with an example how the value of pheromones effects the operator strengths when choosing to move to inferior solutions.

Algorithm 3 ACS: Ant Colony search algorithm for finding an optimal coalition structure

$pheromones \leftarrow [\,]$ {initialise pheromones with each operator merge, split, shift}
$cost \leftarrow [\,]$ {initialise cost for each operator merge, split, shift}
$CS \leftarrow$ randomly generated CS; {generate a random start point}
$bestCS \leftarrow CS$
for *iterationCount* ← 1, *maxIterations* ***do***
  Generate the neighbourhood N of CS with shift, merge and split operators
  $CurBest \leftarrow$ best neighbouring structure within N
  if a $v(CurBest)$ in $N > v(CS)$ **then**
    $CS \leftarrow CurBest$ {Update CS}
    update $CurBest$ operator pheromone
    Generate the neighbourhood N' of $CurBest$ using same operator as $CurBest$
    if a $v(\mathrm{X})$ in $N' > v(CurBest)$ **then**
      $CS \leftarrow \mathrm{X}$ {Update CS}
      end if
  else
    if all $v(CurBest)$ in $N < v(CS)$ **then**
      calculate $OS_{operator} = \frac{(v(CS)-v(CS'))\times \mathrm{P}_{\mathrm{operator}}}{\mathrm{C}_{\mathrm{operator}}}$ {evaluate inferior solution}
      $CS \leftarrow CurBest$ with **highest** $OS_{operator}$ value
      update $CurBest$ operator pheromone
      Generate the neighbourhood N' of $CurBest$ using same as $CurBest$
      if a $v(X)$ in $N' > v(CS)$ **then**
        $CS \leftarrow X$ {Update CS}
      else
        a $v(X)$ in $N' < v(CS)$ **then**
          $CS \leftarrow CurBest$ {Update CS}
      end if
    if $v(CS) \geq v(bestCS)$ **then**
      $bestCS \leftarrow CS$ {Update $bestCS$}
end for
Return $bestCS$

Example

For this particular example (see Figure 5.1), at the start of the iteration, the value of CS shown as ant(start) is 6. Using the three neighbourhood operators, three neighbouring solution shown as ant(shift), ant(merge) and ant(split) are generated. Among these three, none has a value

higher than ant(start). Thus to decide which solution to use to continue the search from, the operator strengths are calculated using the formula given in Equation (5).

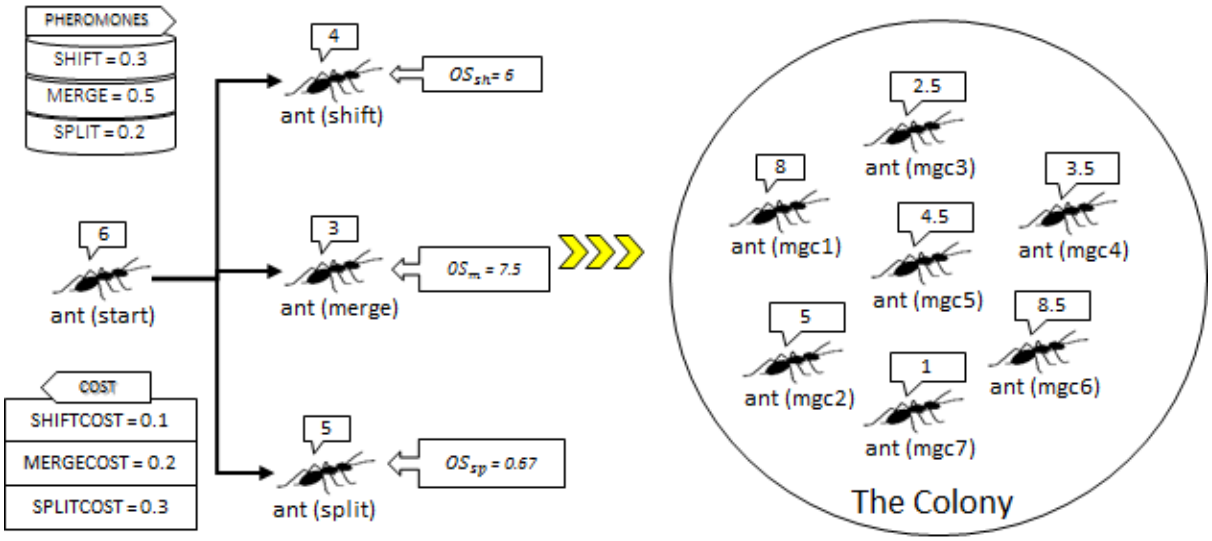

Figure 5.1 – Ant colony movement by operator strength.

The resulting operator strengths for OS_{sh}, OS_m and OS_{sp} are 6, 7.5 and 0.67 respectively. Thus ant(merge) has the highest operator strength. Therefore, the search will continue by generating an entire neighbourhood of ant(merge) as shown in Figure 5.1. Within this neighbourhood, ant(mgc6) has a higher value than ant(merge). Thus the next iteration will start with ant(mgc6) as the starting solution.

If none of the neighbours in the colony have a higher value than ant(merge) then ant(merge) will become the starting solution for the next iteration. This cycle only applies when the neighbouring solutions generated from ant(start) are inferior to ant(start). Otherwise if anyone of the ant(shift), ant(merge), ant(split) solution has a higher value, the next iteration will start from the ant with the higher value.

5.4 Particle Swarm Search

Particle swarm optimisation (PSO), a heuristic method for solving combinatorial optimisation problems, was first introduced by Kennedy & Eberhart (1995) to emulate social behaviour.

This is represented as movements of organisms (particles) for example in a school of fish or flock of birds. Using this behaviour to move a swarm toward good solutions in the search space is known as swarm intelligence (Kennedy & Eberhart, 1995).

PSO has been applied to numerous hard problems such as the travelling salesman problem (Clerc, 2004) and coalition formation problem (Guo & Wang, 2006). The method can, in general, be used to solve any optimisation problem. Application of this method requires correct modelling and adaptation of the problem space to the movement of particles flying through the search space looking for solutions. Many variants of the method have been proposed (Trelea, 2003). A particle swarm based method devised for solving the CSG problem is given in Algorithm 4.

Algorithm 4 PSS: Particle Swarm search algorithm for finding an optimal coalition structure

```
1:  gbest ← [ ] {gbest initially empty}
2:  ac ← set initial acceptance criteria {minimum solution quality for pbest}
3:  for iterationCount ← 1, maxIterations do
4:    pbest ← [ ] {pbest initially empy}
5:    SP ← Initialize start particles
6:      CS ← randomly generated CS; {SP generate a random start point}
7:      Generate the neighbourhood N of CS with selected Neighbourhood Operators
8:      Sort the neighbours in N from highest to lowest {Highest solution on top}
9:          Update the highest value found into pbest
10:     SW ← Initialize swarm particles SWx {x is the particle number e.g. SW1)
11:         SWx ← Generate the neighbourhood Nx {x is the index of the neighbour e.g. N1)
12:         Update the highest value found for each SWx into pbest
13:   if one of the pbest > gbest then
14:     gbest ← pbest
15:     CS ← pbest
16:   else
17:     no pbest > gbest
18:     evaluate highest value pbest
19:     if pbest meets acceptance criteria
20:         CS ← pbest
21:     else
22:         CS ← a randomly generated coalition structure
23:     end if
24:   end if
25: end for
26: Return gbest
```

There are two key variables: a *global best* (gbest) that stores the best solution found so far in the entire search space, and *personal best* (pbest) that is best within a swarm. To begin, gbest is initialised to a random coalition structure. Then, a complete neighbourhood of CS is generated using all the three neighbourhood operators. Suppose we want to create four swarms. Then, the four best coalition structures will be chosen from the generated

neighbourhood. These are used to generate four swarms SW1, SW2, SW3, and SW4. Then the neighbourhood of each swarm is generated. We then find the pbest for each of the four swarm neighbourhoods. If the best of the four pbest solutions (call it *X*) is better than gbest, then gbest is updated to *X*. Otherwise, if the value of gbest is better than *X*, and value of *X* is at least *Y*% of the value of CS, then CS is updated to *X* (this is done to escape local optima). Otherwise, CS takes on a random coalition structure and the next iteration is commenced. The value of *Y* is initially set at 50 but is increased with the number of iterations. (see Algorithm 4).

Table 5.1 – Comparison of the Heuristic Methods.

	Tabu Search	Simulated Annealing	Particle Swarm	Ant Colony
Search Movement	Moves are recorded, tabu list keeps a record of previous moves to guide the search to move to areas of the search space yet to be explored. Moving to a solution already found in the tabu list is prohibited.	Moves are random and accepted on a certain acceptance probability. Acceptance probability depends on several parameters such as temperature. No record is kept of already visited solution.	Moves from one good solution to the next as a swarm (parallel), when a non-improving solution found a fitness function is used to determine fitness of next move. Global best is kept to be used when calculating fitness.	Mimic the movement of ants in a colony. Start by moving to good source of food then bring colony to find good solutions around it. If no better solution found, refer to history on which moves was beneficial and moves in that direction.
Advantages	Guided search and a structured neighbourhood enables systematic search of the solution space. Eliminates explored solution from search, only explores new solution space avoiding repeated moves.	Moves are quicker as it does not need to generate a neighbourhood structure to evaluate its movement. Memory less implementation, requires less resources.	Inherent parallelism. Search moves as a swarm with potential to horizontally cover a wider space. Uses small amounts of long term memory to store global best and short term memory to store own best.	Moves are guided by trails left by other ants in previous search, when a non-improving move encountered, refer to history of moves to guide next move, some move history preserved. Short term memory only to store pheromones.
Disadvantages	Requires generation of a neighbourhood to evaluate moves. Requires keeping in memory a list called tabu list that consumes more resources. May use a lot of long term memory.	Random selection could result in repeated moves. Unguided jumps could mean exploration might be stuck in worse parts of the search space.	Sometimes slower convergence especially when many non-improving solution encountered slowing down movement to new parts of the space.	Implementation is strictly problem specific, care must be given to parameters used for keeping search history. Keeping pheromones for too long will effect search.

5.5 Chapter Summary

This chapter presented four heuristic methods for finding the optimal coalition structure. These methods are Tabu Search, Simulated Annealing, Ant Colony, and Particle Swarm Search. Table 5.1 gives a general comparison between the four heuristics. The implementation details of these methods was given earlier in this chapter.

Chapter 6 Simulation Setup for Performance Evaluation

The four heuristic methods viz., tabu search, simulated annealing, ant colony, and particle swarm were evaluated and compared in terms of two criteria:

1. **Running time**, i.e., time taken to generate a solution.
2. **Quality of solution**, i.e., how close a heuristic solution is to the exact optimum. In those cases, where the search for exact optimum was computationally infeasible, an upper bound on the exact optimum was used.

Performance evaluation and comparison was done for CFGs as well as PFGs. All the simulations were run on an Apple iMac computer equipped with Intel Core i5 4570R Processor running at 2.7 GHz (3.2 GHz Turbo) and 8GB of RAM running Windows 7 Enterprise Edition. All four heuristic methods were implemented in Python (Python Software Foundation, https://www.python.org/).

This chapter is organised as follows. Section 6.1 is a description of how the quality of a heuristic solution was determined. Section 6.2 is a description of how data was generated for evaluating performance. Section 6.3 is a description of how the upper bounds for an exact optimum were calculated.

6.1 Evaluation Method

The quality of a heuristic solution relative to the exact optimum is calculated as follows. Let $CS_{TACOS}, CS_{SA}, CS_{ACS}, CS_{PSS}$ be the solutions (i.e., coalition structures) returned by TACOS, simulated annealing, ant colony and particle swarm methods respectively. Let CS_{opt} be the *exact* optimal solution. The quality of a solution generated by TACOS is measured as follows:

$$\frac{v(CS_{TACOS})}{v(CS_{opt})} \times 100 \qquad (6.1)$$

The quality for simulated annealing, ant colony, and particle swarm is analogous.

Quality of solution relative to an upper bound is measured as follows:

$$\frac{v(CS_{TACOS})}{v(Upper\ Bound)} \times 100 \qquad (6.2)$$

where $v(Upper\ Bound)$ is the upper bound on the exact optimum. Details regarding the calculation of $v(Upper\ Bound)$ can be found in Section 6.3.

Time taken to generate a heuristic solution is set as follows. Each of the four heuristic methods was given the same average running time over a fixed number of iterations (see Table 6.3). This running time was calculated as follows. For 25 agent CFGs, we ran TACOS for 1800 iterations. This took 30000ms.

For CFGs, these times were set as shown in Table 6.1.

Table 6.1 – Running Time (rounded to next decimal) for CFGs.

Input Size	Average Running Time (ms)
25-agents	30000
27-agents	60000

For a CFG comprised of 25 agents, the quality of a heuristic solution was evaluated relative to the exact optimum. For 27 agents, computing the exact optimum was infeasible, so the quality of a heuristic solution was evaluated relative to the upper bound for the exact optimum.

For PFGs with 10 agents, heuristic solution quality was evaluated relative to the exact optimum. For PFGs with 27 agents, heuristic solution quality was evaluated relative to the upper bound for the exact optimum. The running times for PFGs were set as shown in Table 6.2.

Table 6.2 – Running Time (rounded to next decimal) for PFG.

Input Size	Average Running Time (ms)
10-agents	180000
27-agents	300000

It should be noted that, for a heuristic method, the number of iterations possible within the targeted average running time differs depending on the heuristic method. For example, within a 1 minute running time, more iterations may be possible with TACOS compared to ACS or PSS.

The tabu search method TACOS was run in two different modes: single agent mode (this corresponds to a single thread) and multiagent (this corresponds to multiple threads) mode. The former is denoted TACOS(S) and the latter TACOS(M). A multiagent approach (i.e., TACOS(M)) resulted in a lower number of iterations due to the need to initialise multiple threads and synchronise them. For a fixed running time, the number of iterations for different methods are as given in Table 6.3.

Table 6.3 – Number of Iterations and the corresponding run time for each method.

Simulation Type	**Method**	**Number of Iterations**	**Average Running Time**
25-agent CFG	TACOS	1800	30 Seconds
	SA	300	
	ACS	6000	
	PSS	200	
27-agent CFG	TACOS-S	2500	1 Minute
	TACOS-M	1100	
	SA	250	
	ACS	8000	
	PSS	120	
10-agent PFG	TACOS	10	3 Minutes
	SA	200	
	ACS	30	
	PSS	3	
27-agent PFG	TACOS-S	12000	5 Minutes
	TACOS-M	6000	
	SA	2500	
	ACS	1000	
	PSS	100	

6.2 Data Generation for Performance Evaluation

The data needed to evaluate performance is the value of each coalition. Data sets comprised of coalition values were generated randomly using six different probability distributions (details regarding these distributions are given in Section 6.2.2). The probability distributions chosen are commonly used in recent literature (Michalak et al., 2016; Sandholm et al., 1998) with the exception of the Triangular distribution which was chosen to explore a distribution that has yet been studied for the CSG problem.

Since the performance of all four heuristic methods is sensitive to the starting point, which is random, each method was run ten-times for each of the datasets generated. Average solution quality and average running time were then measured across the ten runs for each data set. Ten datasets were generated for each of the six probability distributions. This means that, in total, there are 60 datasets for each scenario. Four different scenarios were considered: (25-agent CFGs, 27-agent CFGs, 10-agent PFGs, and 27-agent PFGs).

6.2.1 Data for Characteristic Function Games

In CFGs, each coalition has a single value that is independent of external coalitions. Thus, the value of each coalition is drawn only once from a probability distribution. The six probability distributions used to generate coalition values are as follows:

1. Uniform: For all $C \in C^{Ag}$, the value $v(C) \sim \mathrm{U}(0, |\mathrm{C}|)$
2. Normal: For all $C \in C^{A}g$, the value $v(C) \sim \mathrm{N}(\mu, \sigma 2)$ where $\mu = |\mathrm{C}|$, $\sigma = 0.1$
3. Gamma: For all $C \in C^{Ag}$, the value $v(C) \sim |\mathrm{C}| \times \mathrm{Gamma}(\mathrm{k}, \theta)$, where $\mathrm{k} = 7.5$, $\theta = 0.5$
4. Beta: For all $C \in C^{Ag}$, the value $v(C) \sim |\mathrm{C}| \times \mathrm{Beta}(\alpha, \beta)$, where $\alpha = \beta = 0.5$.
5. Exponential: For all $C \in C^{Ag}$, the value $v(C) \sim |\mathrm{C}| \times \exp(\lambda)$, where $\lambda = 3$.
6. Triangular: For all $C \in C^{Ag}$, the value $v(C) \sim \Delta(\mathrm{a}, \mathrm{b}, \mathrm{c})$ where a = 0, b = 1, c=none

These distributions were used to generate data for 10-agent, 25-agent and 27-agent games. The parameters chosen are similar to those found in literature (Michalak et al., 2016; Sandholm et al., 1998).

6.2.2 Data for Partition Function Games

In PFGs, the value of a coalition depends on the coalition structure in which the coalition is embedded. Thus, data for PFGs requires much more storage. For example, in a search space with 20-agents, if a coalition is a member of 150 different structures, it means that there are 150 possible values for just this one coalition. Storing all these coalition values for games with more than ten agents is infeasible in terms of memory space. Thus, simulations were conducted for 10-agent games. A random value for a coalition was drawn for each structure it is embedded in. The probability distributions used are the ones listed above.

Two types of externalities can arise: **positive** and **negative** (De Clippel & Serano 2008).

- Positive externality: An increase in the value of a coalition that results from reorganization of external coalitions means the externality on the coalition is positive.
- Negative externality: A decrease in the value of a coalition that results from reorganization of external coalitions means the externality on the coalition is negative.

In order to facilitate the calculation of upper bounds (see Section 6.3 for details regarding the calculation of upper bounds), data for 27-agent games was generated with two different parameters for each probability distribution. Depending on the number of member agents, coalitions were divided into two types: those of size $2 \leq k \leq n-1$, and for a given *k*, those of size i ($i \neq k$ and $1 \leq i \leq n$). The value of a coalition was generated using the following parameters for probability distributions (here n is the number of agents in a game):

1. Uniform Distribution:
   For all $C_i \in C^{Ag}$, the value $v(C_i) \sim \mathrm{U}(a, b)$ where $a = 0$ and $b = n$
   For all $C_k \in C^{Ag}$, the value $v(C_k) \sim \mathrm{U}(a, b)$ where $a = n^2$ and $b = 2(n^2)$
2. Normal Distribution:
   For all $C_i \in C^{Ag}$, the value $v(C_i) \sim \mathrm{N}(\mu, \sigma 2)$ where $\mu = n$, $\sigma = 0.2$
   For all $C_k \in C^{Ag}$, the value $v(C_k) \sim \mathrm{N}(\mu, \sigma 2)$ where $\mu = (n+1)^2$, $\sigma = 0.2$
3. Gamma Distribution:
   For all $C_i \in C^{Ag}$, the value $v(C_i) \sim \mathrm{Gamma}(k, \theta)$, where $k = 7.5$, $\theta = 0.5$
   For all $C_k \in C^{Ag}$, the value $v(C_k) \sim (n+1)^2 \times \mathrm{Gamma}(k, \theta)$, where $k = 7.5$, $\theta = 0.5$

4. Beta Distribution:
   For all $C_i \in C^{Ag}$, the value $v(C_i) \sim \text{Beta}(\alpha, \beta)$, where $\alpha = 5$ and $\beta = 1$
   For all $C_k \in C^{Ag}$, the value $v(C_k) \sim n^2 \times \text{Beta}(\alpha, \beta)$, where $\alpha = 5$ and $\beta = 1$
5. Exponential Distribution:
   For all $C_i \in C^{Ag}$, the value $v(C_i) \sim \exp(\lambda)$, where $\lambda = 3$.
   For all $C_k \in C^{Ag}$, the value $v(C_k) \sim n^2 \times \exp(\lambda)$, where $\lambda = 3$.
6. Triangular Distribution:
   For all $C_i \in C^{Ag}$, the value $v(C_i) \sim \Delta(a, b, c)$ where a = 0, b = n , c=none
   For all $C_k \in C^{Ag}$, the value $v(C_k) \sim \Delta(a, b, c)$ where a = n^2, b = $2(n^2)$, c=none

These parameters only apply to simulations for 27-agent games.

6.3 Calculating Bounds

We used two different methods for calculating the upper bound on the value of a coalition structure. One method is for CFGs and the other for PFGs.

6.3.1 Calculating the Upper Bound for CFGs

In order to understand the calculation of bounds, we need to first understand how coalition structures are represented. The space of all coalition structures, can be represented as a coalition structure graph (T. Sandholm et al., 1998) or as integer partitions (Rahwan & Jennings, 2007). We shall be using the latter representation as it is more suitable for our purposes.

An integer partition of n is a sequence of positive integers whose sum is exactly n. As an example, consider a set of $n = 5$ agents. There are 7 possible partitions: [5], [2, 3], [1, 4], [1, 2, 2], [1, 1, 3], [1, 1, 1, 2] and [1, 1, 1, 1, 1]. The parts in an integer partition represent the sizes of the coalitions in a coalition structure. For example, the coalition structure {{a1},{a2},{a3,a4,a5}} and {{a2},{a3},{a1,a4,a5}} corresponds to the integer partition [1, 1, 3] as there are three coalitions, two of which are of size one and of size three.

The number of integer partitions of n grows exponentially in n. For 27-agents, the integer partition count is 3010, while there are 134,217,728 possible coalitions, and over five hundred forty five quadrillion (≈ 545,717,047,947,902,329) coalition structures.

The integer partition representation divides the search space into subspaces. All partitions with an equal number of parts are put in the same subspace. For example, the two part partitions [2, 3] and [1, 4] belong to the same subspace containing all partitions with two parts. Therefore, when there are 5 agents, the sub-spaces are defined as shown in Figure 6.1.

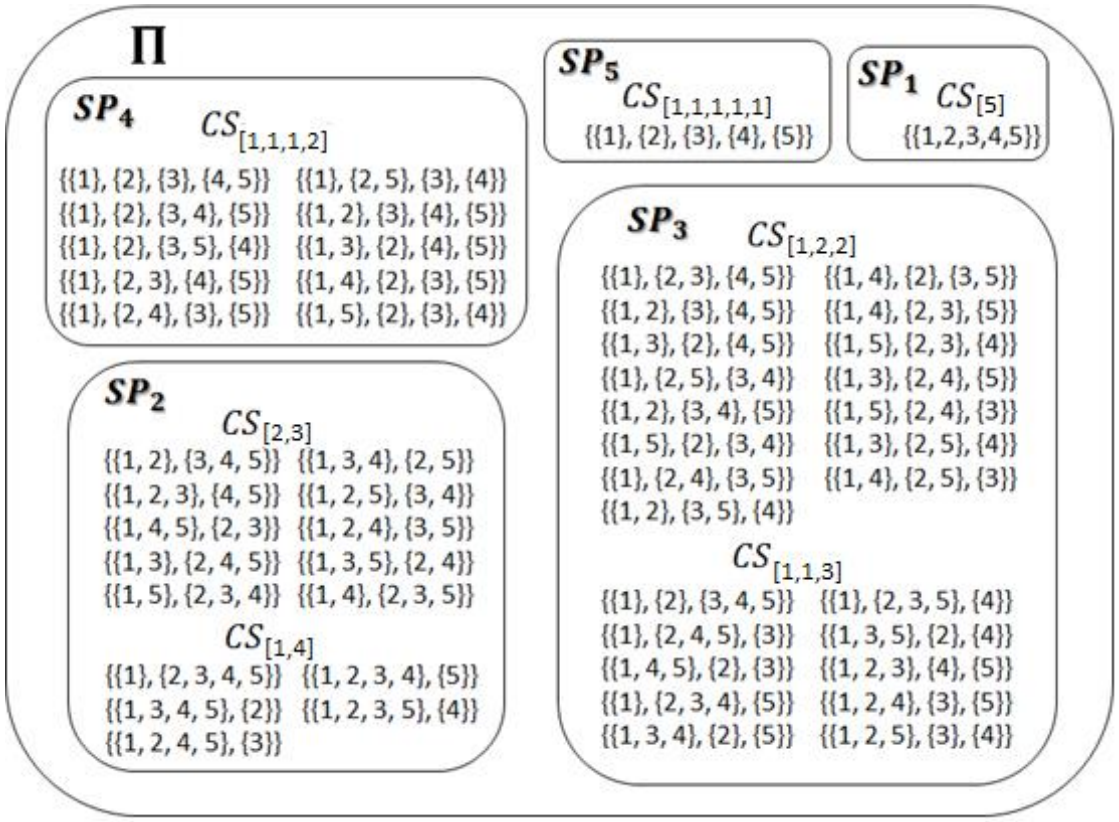

Figure 6.1 – Space of all coalition structures for 5 agents.

Here, Π denotes the space of all coalition structures while SP_i denotes the subspace with integer partitions comprised of i parts (e.g. SP_2 contains all structures with two parts or two coalitions. We let $CS_{|i,j|}$ denote the set of all coalition structures with two parts, the first part of which contains i agents and the second part contains j agents.

The upper bound on the value of structures in the set $CS_{|i,j|}$ is calculated as follows. Recall that the value of a coalition is drawn from a distribution. The maximum value of a coalition thus depends on the parameters of the distribution from which it is drawn. For example, for

the Uniform distribution $U(a, b)$, the values drawn depend on two parameters: a and b. This means that the minimum possible value of a coalition is a and the maximum is b.

The upper bound on the value of structures in the set $CS_{|IP|}$ is the sum of the upper bounds of each part in IP. Likewise, the lower bound on the value of structures in the set $CS_{|IP|}$ is the sum of the lower bounds of each part in IP.

For some probability distributions, the upper bound on the value of a coalition cannot be easily determined in terms of the parameters of the distribution. For example, for the Normal distribution, a randomly drawn value may go up to infinity. For such cases, the upper bound on the value of a coalition of a certain size was the highest of all randomly drawn values (in the data sets) for coalitions of that size. Then, the upper bound for a coalition structure is the sum of the upper bounds for each integer partition of the structure (as in (Rahwan et al., 2007)).

For PFGs, the calculation of upper bound is not as straightforward as it is for CFGs. The following section describes upper bound calculation for PFGs.

6.3.2 Calculating the Upper Bound for PFGs

For PFGs, externalities must to be considered. Recall from Chapter 2, that externalities are modelled by agent type. Recall also that there are two types of agents: Type A and Type B. The default ratio between the number of Type A and Type B agents for *n*-agent games is set as follows:

a) Number of Type A agents is $\frac{n}{3}$

b) *N*umber of Type B agents is $n - \frac{n}{3}$

Now, the upper bound for the value of a coalition of a given size depends not only on its size but also on the structure it is embedded it. In order to incorporate externalities, we introduce an **externality factor**. This factor is then used to calculate the value of a coalition. In PFGs, the value of a coalition depends on

1. the of member agents in it, and
2. the organization of external coalitions.

Thus, we defined externality factor for a coalition as follows. Let $m = |C|$, denote the size of a coalition C and $d = |CS|$, denote the number of coalitions in the structure $CS\ it\ is\ embedded\ in$. Externalities imposed on a coalition $C \in CS$ are given by an externality factor. The externality factor, ef, for coalition *C* in structure *CS* is calculated as follows:

$$ef = \frac{m}{d} \qquad (6.3)$$

As with CFGs, the value of a coalition for PFGs is randomly drawn. However, for PFGs, the randomly drawn value is combined with the externality factor of the coalition. As there are two types of externalities, positive and negative, we have the following two formulae for calculating the value of a coalition.

1. Positive externalities: The value $v(C)$ of any coalition C is calculated as follows. Let RV denote a randomly drawn value from any probability distribution. This value is **increased** by a factor of ef to obtain $v(C)$ as follows:

$$v(C) = RV + (RV \times ef) \qquad (6.4)$$

2. Negative externalities: The value $v(C)$ of any coalition C is calculated as follows. Let RV denote a randomly drawn value from any probability distribution. This value is **decreased** by a factor of ef to obtain $v(C)$ as follows

$$v(C) = RV - (RV \times ef) \qquad (6.5)$$

Suppose $2 \leq k \leq n\text{-}1$ and for a given *k*, let $i \neq k$. The entire search space will consist of the following three types of coalition structures:

i) Type *k* structure: A coalition structure all of whose members are coalitions of size *k*. Such a structure is denoted CS^k .

ii) Type *i* structure: A coalition structure all of whose members are coalitions of size *i*. Such a structure is denoted CS^i .

iii) Type *i,k* structure: A coalition structure some of whose members are coalitions of size *i* and some of size *k*. Such a structure is denoted $CS^{i,k}$.

For simulations, we set $k = 2$ since this provided the largest coverage of the search space for any number n of agents. Table 6.4 shows this for a 27-agent scenario.

Table 6.4 – Number of Integer Partitions for each k for 27-agent games.

k	Number of Integer Partitions with coalitions of size k
2	1958
3	1575
4	1255
5	1002
6	792
7	627
8	490
9	385
10	297
11	231
12	176
13	135
14	101
15	77

We denote the upper and lower bound on the value of a coalition as follows:

i) UB_k denotes the upper bound on the value of any coalition of size *k*.

ii) LB_k denotes the lower bound on the value of any coalition of size *k*.

iii) UB_i denotes the upper bound on the value of any coalition of size *i*.

iv) LB_i denotes the lower bound on the value of any coalition of size *i*.

The parameters of the probability distributions are set such that the following relation is true.

$$LB_k \gg UB_i \qquad (6.6)$$

The data (i.e. the values of coalitions drawn from the six probability distributions mentioned earlier) was verified so that it satisfies the relationship in Equation (6.6) such that the value of any coalition structure with at least one size k coalition is always greater than the value of any coalition structure without a coalition of size k.

The upper bound on the value of any coalition structure of type CS^i are computed as follows:

$$UB(CS^i) = \frac{n}{i} \times UB_i \qquad (6.7)$$

Likewise, the upper bound on the value of any coalition structure of type CS^k are computed as follows:

$$UB(CS^k) = \frac{n}{k} \times UB_k \qquad (6.8)$$

And for type $CS^{i,k}$, some of whose members are coalitions of size i and others of size k, the upper bound on the value is:

$$UB(CS^{i,k}) = \left(\sum_{C \in CS^{i,k}, |C|=i} UB_i + \sum_{C \in CS^{i,k}, |C|=k} UB_k \right) \qquad (6.9)$$

We will illustrate the calculation of bounds in the context of the following example.

Example:

Consider a 4 agent PFG. For 4 agents, there are 5 possible integer partitions: [1,1,1,1], [1,1,2], [1,3], [2,2] and [4]. Assume that $k = 2$.

Recall form Chapter 2 that there are two types of agents: Type A and Type B. From these two agent types, **three** different types of coalitions can be formed:

1. Type AA: Recall that such a coalition contains only Type A agents.
2. Type BB: Recall that such a coalition contains only Type B agents.
3. Type MX coalitions: Recall that such a coalition contains both Type A and Type B agents.

This means that there are **five** distinct types of coalition structures possible:

1. Type AABB: Recall that such a structure contains some coalitons of Type AA and others of Type BB.
2. Type AAMX: Recall that such a structure contains some coalitons of Type AA and others of Type MX.
3. Type BBMX: Recall that such a structure contains some coalitons of Type BB and others of Type MX.
4. Type MXMX: Recall that such a structure only contains coalitons of Type MX.
5. Type AABBMX: Recall that such a structure contains some coalitons of Type AA, some of Type BB and others of Type MX.

In this 4 –agent example, suppose agent types are as follows:

i) There is only one Type A agent.
ii) There are three Type B agents.

Since there is only 1 Type A agent, there can be only 2 (out of the five possible types listed above) types of coalition structures:

i) Type BBMX whose member coalitions are of Type BB and Type MX.
ii) Type AABB whose member coalitions are of Type AA and Type BB.

Although the grand coalition can be formed, externalities on it are not considered as they are no external coalitions in that structure. In this example, because there is only one Type A agent, the coalitions of size k (recall also that $k=2$) can only be of Type BB and that only partitions [1, 1, 2] and [2, 2] have coalitions of size k.

Due to (6.6), any integer partition that does not contain coalitions of size k can be ruled out from having the upper bound. Let RV_1 denote the randomly drawn value for a coalition of size i and RV_2 that for size k. The inequality $LB_{\mathrm{k}} \gg UB_i$ means that the upper bound on RV_1 will always be lower than the lower bound value of RV_2. With this in mind, the upper bound for each integer partition containing a coalition of size k can be calculated using Equations (6.8) and (6.9) as shown in Table 6.5.

Table 6.5 – Upper bound for partitions containing coalition of size k.

Type	Possible Partition	Value of structure	Upper bound
BBMX	[2,2]	$RV_2 - (\frac{2}{2} \times RV_2) + RV_2 + (\frac{2}{2} \times RV_2)$	$2 * UB(RV_2)$
	[1,1,2]	$RV_1 - (\frac{1}{3} \times RV_1) + RV_1 - (\frac{1}{3} \times RV_1) + RV_2 + (\frac{2}{3} \times RV_2)$	$\frac{4}{3}UB(RV_1) + \frac{5}{3}UB(RV_2)$
AABB	[1,1,2]	$RV_1 - (\frac{1}{3} \times RV_1) + RV_1 - (\frac{1}{3} \times RV_1) + RV_2 + (\frac{2}{3} \times RV_2)$	$\frac{4}{3}UB(RV_1) + \frac{5}{3}UB(RV_2)$

In this example, there can be only two types of coalition structures: BBMX and AABB.

First, consider Type BBMX structure shown as Row 1 in Table 7.8. For partition [2,2] there can only be a coalition structure of Type BBMX where the value of Type BB coalition is decreased due to the presence of a Type MX coalition (refer to Chapter 2 for details on determining positive or negative externalities). Thus the upper bound for partition [2,2] is twice the upper bound for RV_2. The upper bound for RV_2 is the highest randomly drawn value (for size 2 coalition) between all data sets.

For the other partition [1,1,2], when the coalition structure is of Type BBMX and coalition 2 is TypeMX, the other two coalitions can only be Type BB as there is only one Type A agent. Thus, the value of coalition 2 is changes into $\frac{5}{3}UB(RV_2)$. Each one of the other two coalitions will be Type BB. Then, due to the presence of a Type MX coalition, their values are decreased. Thus, the total value of this coalition structure is $\frac{4}{3}UB(RV_1)+\frac{5}{3}UB(RV_2)$ which is lower than $2 * UB(RV_2)$ due to the relationship in (6.6).

Next, consider Type AABB structure shown as the third row in Table 6.5. When the coalition structure is of Type AABB and coalition 2 is Type BB while the other two coalitions is of Type AA and Type BB respectively the value of coalition 2 is increased thus the upper bound $\frac{5}{3}RV_2$. For the other two coalitions, their values are decreased with $\frac{4}{3}RV_1$ being the sum of the value of both coalition. Thus, the total value of this coalition structure is $\frac{4}{3}RV_1+\frac{5}{3}RV_2$ that is also lower than $2RV_2$.

Thus, between all 3 inter partitions possible, the upper bound for integer partition [2, 2] is the highest (note that this integer partition contains the maximum number of size k coalitions) and this will be taken as the upper bound for performance evaluation.

6.4 Chapter Summary

This chapter provided information on the simulation setup for evaluating the methods implemented. First we presented our method of evaluating the performance of each algorithm. This is followed by information how the input instances are generated. Lastly we provided details on how the bound for each setting was obtained.

Chapter 7 Performance Analysis for the Individual Probability Distributions

All four heuristic methods, tabu search, simulated annealing, particle swarm, and ant colony were evaluated in the simulation set up detailed in Chapter 6. This chapter is about the results of simulations and a comparative performance analysis of these four methods. More precisely, the objectives of this chapter are as follows:

1. To study the performance of each of the four heuristic methods for each of the six probability distributions for CFGs.
2. To study the performance of each of the four heuristic methods for each of the six probability distributions for PFGs.
3. To study the effect of an increase in the number of agents on the performance of each of the four heuristic methods for each of the six probability distributions for CFGs.
4. To study the effect of an increase in the number of agents on the performance of each of the four heuristic methods for each of the six probability distributions for PFGs.

Recall that data sets comprise values of coalitions. These values were generated randomly using six different probability distributions. For CFGs, the evaluation was done for two settings: 25 agent CFGs and 27 agent CFGs. These settings were chosen because 25-agent was the maximum size where an actual optimal can be obtained for comparison and 27-agents was the maximum coalition value data that can be accepted by Python.

For PFGs, the evaluation was done for two settings: 10 agent PFGs and 27 agent PFGs. All heuristic methods were run for the some duration of time. These two settings was chosen because 10-agent was the maximum size where an actual optimal can be obtained in PFGs, 27-agents was the maximum coalition value data that can be accepted by Python. This was the reasoning behind the choice of the four scenarios considered in this thesis .The run time was fixed so that TACOS achieved a solution that was 80% of optimal.

The rest of this chapter is organized as follows. The results for CFGs are given in Section 7.1 and for PFGs in Section 7.2. Section 7.3 is a study of the effect of the number of agents on heuristic performance for CFGs and for PFGs.

7.1 Performance for Characteristic Function Games

We will first consider 25 agent CFGs in Section 7.1.1 and then 27 agent CFGs in Section 7.1.2. These results are the averages of 100 runs over 10 datasets for each game.

7.1.1 Performance for 25-Agent CFGs

We will examine performance for each of the six probability distributions individually.

The Uniform Distribution

The run time was set at 30 seconds. The performance of a heuristic solution is measured as a percentage of the exact optimum. The performance of each heuristic method is shown in Table 7.1. All the four heuristic methods performed well with each returning an average solution quality that is above 90% of the optimum (see Table 7.1).

TACOS(S) performed the best achieving a solution that is 99.77% of the optimum within 30 seconds. This is followed by ACS at 98.01%, then PSS with 96.97% with SA yielding the lowest results of 93.38% of optimum. Note that all methods are quite close to each other and even the worst performing heuristic achieves over 93% of the optimum.

Let us now examine the number of neighbours explored by each method (see Table 7.1). SA explored the most number of neighbours as it ran for a higher number of iterations, however this did not translate into best performance among all the methods here. The reason for this is that SA is a memory less method. It does not remember the good/bad performing neighbours, only the solution that is best so far. TACOS(S) on the other hand explored just slightly more neighbours than PSS and achieved 2.8% higher solution quality relative to PSS.

These results show that exploring more neighbours does not necessarily give a better quality solution. It is important to remember the good/bad neighbours. This results in:

1. Avoiding repeatedly visiting the same neighbours.
2. Directing search in the promising directions.

Table 7.1 – Average Performance for 25-agent CFGs (Uniform Distribution).

Uniform Distribution		
Method	Average Neighbours Explored (≈30 Seconds Running Time)	Average Performance - % of Optimal (≈30 Seconds Running Time)
TACOS(S)	204901	99.77
SA	1391243	93.38
ACS	306913	98.01
PSS	204259	96.97

The Normal Distribution

For Normal distribution (see Table 7.2), ACS performed the best, with an average solution quality that is better than TACOS by 1.72% (with ACS at 96.92% vs 95.20% for TACOS(S)). Note that ACS explored more coalition structures than TACOS(S) here, and this translated to a real performance gain for ACS. Third place again belongs to PSS followed closely by SA, with PSS having only 0.73% advantage over SA which is again the lowest performing method.

Table 7.2 – Average Performance for 25-agents CFGs (Normal Distribution).

Normal Distribution		
Method	Average Neighbours Explored (≈30 Seconds Running Time)	Average Performance - % of Optimal (≈30 Seconds Running Time)
TACOS(S)	206511	95.20
SA	1486927	92.84
ACS	305823	96.92
PSS	202742	93.57

Looking at the number of neighbours explored, the ordering in Table 7.2 for Normal distribution is the same as the ordering in Table 7.1 for the Uniform distribution. Despite this, the performance ranking in the tables is different, except for SA which got rank four for both distributions.

The Gamma Distribution

We shall now take a look at the Gamma distribution. TACOS(S) again takes the lead with a solution quality of around 78.54% of optimal (see Table 7.3). This time, SA is the second best performing method followed by ACS and PSS. The performance of SA here could be due to the higher average number of neighbours explored compared to ACS and PSS. PSS is the lowest performing method. Here, the average number of coalition structures explored by TACOS(S) is the least, yet TACOS returned the highest average solution quality demonstrating its search efficiency.

Table 7.3 – Average Performance for 25-agents CFGs (Gamma Distribution).

Gamma Distribution		
Method	Average Neighbours Explored (≈30 Seconds Running Time)	Average Performance - % of Optimal (≈30 Seconds Running Time)
TACOS(S)	201653	78.54
SA	1496818	65.38
ACS	305143	63.83
PSS	204885	63.59

All the methods performed poorly for the Gamma distribution compared to Uniform or Normal distributions. Thus, simulation run time was extended to 60 seconds in order to investigate if this can improve performance. The results for the extended run time are shown in Table 7.4. With the exploration of as many as twice the number of neighbours, TACOS(S) was able to return a solution quality of 82.14% while the other methods showed only marginal increases in solution quality.

Table 7.4 – Extended Running Time for 25-agent CFGs (Gamma Distribution).

Gamma Distribution Extended Running Time		
Method	Average Neighbours Explored (≈60 Seconds Running Time)	Average Performance - % of Optimal (≈60 Seconds Running Time)
TACOS(S)	414632	82.14
SA	1843561	67.49
ACS	633682	65.80
PSS	325852	64.80

The Beta Distribution

For the Beta distribution, all four methods performed very well with each returning an average solution quality that is better than 98% of the optimum. TACOS(S) takes the lead again followed closely by PSS, ACS and SA. The differences in performance between TACOS(S), PSS and ACS are negligible as all three method exceeded 99% average solution quality.

With regard to the number of neighbours explored, the ordering remains in the same as that for the Gamma distribution (with 30 sec run time).

Table 7.5 – Average Performance for 25-agents CFGs (Beta Distribution).

Beta Distribution		
Method	Average Neighbours Explored (≈30 Seconds Running Time)	Average Performance - % of Optimal (≈30 Seconds Running Time)
TACOS(S)	201369	99.99
SA	1395754	98.82
ACS	305775	99.29
PSS	202658	99.52

The Exponential Distribution

For the exponential distribution (see Table 7.6), TACOS(S) returned the best solution quality of 73.91% in 30 seconds. SA is second best with a 22.39% gap between the two. In the third place is ACS and the worst performing method is PSS.

With regard to the number of neighbours explored, SA ranks first yet the quality of its solution is the worst.

Table 7.6 – Average Performance for 25-agents CFGs (Exponential Distribution).

Exponential Distribution		
Method	Average Neighbours Explored (≈30 Seconds Running Time)	Average Performance - % of Optimal (≈30 Seconds Running Time)
TACOS(S)	205038	73.91
SA	1391455	51.52
ACS	305694	49.99
PSS	203781	44.20

Since the best heuristic for exponential distribution, i.e., TACOS(S), only achieved 73% of the optimum, each method was given an extended running time of 30 seconds (i.e., a run time of 60 seconds). Once again TACOS(S) exceeded the performance of the others (see Table 7.7). However, SA lost its second place to ACS and came in last after PSS.

Table 7.7 – Extended Running Time for 25-agent CFGs (Exponential Distribution).

Exponential Distribution Extended Running Time		
Method	Average Neighbours Explored (≈60 Seconds Running Time)	Average Performance - % of Optimal (≈60 Seconds Running Time)
TACOS(S)	409321	82.24
SA	19102038	53.46
ACS	621054	59.38
PSS	630462	53.76

The Triangular Distribution

For the triangular distribution, all four heuristic methods performed well (see Table 7.8). The highest performance is attained by TACOS(S) followed by ACS, PSS and then SA.

Table 7.8 – Average Performance for 25-agent CFGs (Triangular Distribution).

Triangular Distribution		
Method	Average Neighbours Explored (≈30 Seconds Running Time)	Average Performance - % of Optimal (≈30 Seconds Running Time)
TACOS(S)	203617	97.91
SA	1477862	87.44
ACS	305208	92.78
PSS	205103	92.08

Figure 7.1 and Table 7.9 show a combined comparative summary of the results.

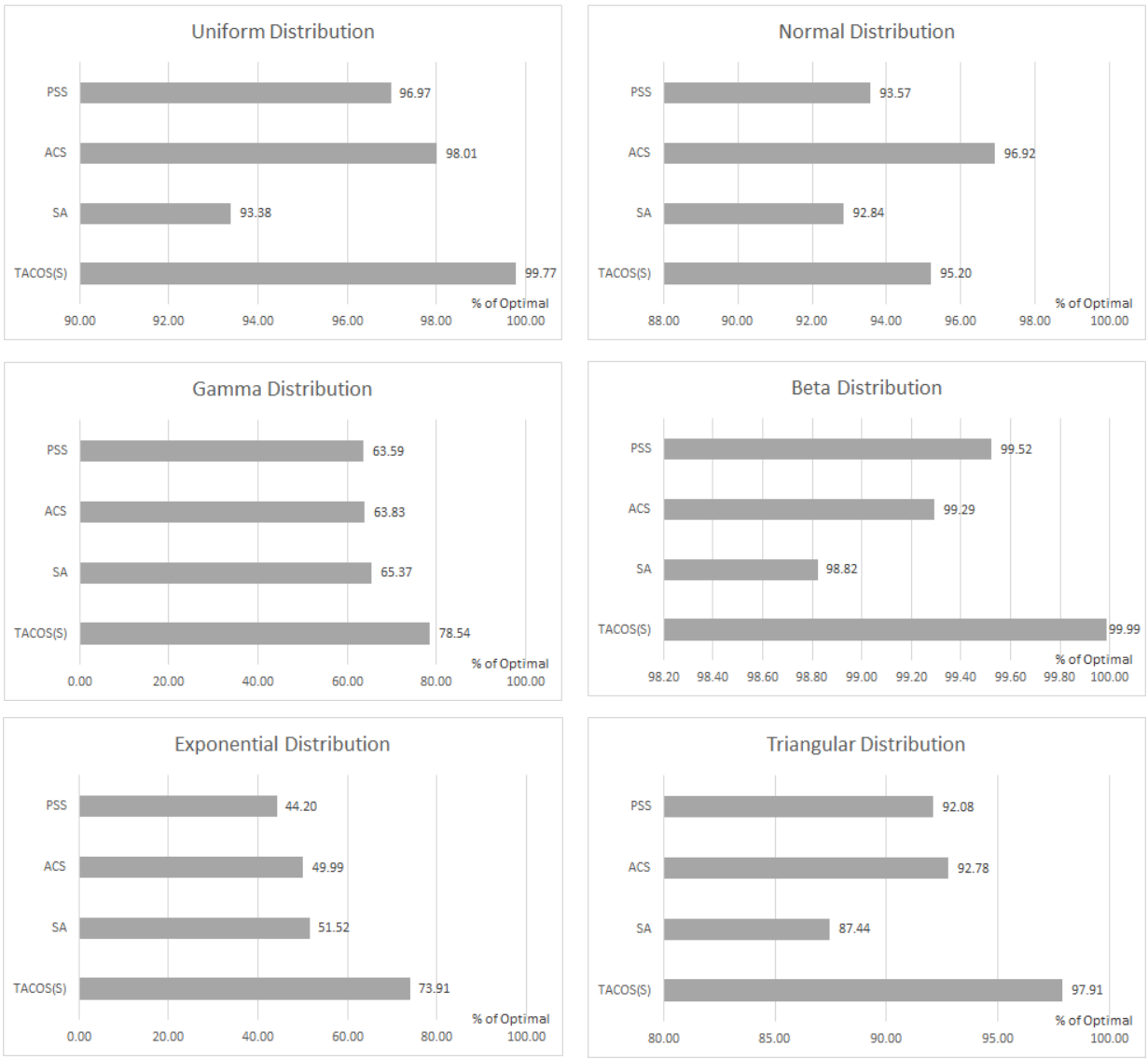

Figure 7.1 – Performance Comparison for 25-agents (CFG) ≈30 seconds running time.

TACOS(S) performed well across all distributions and is the highest performing method for five out the six probability distribution. It was only outperformed slightly by ACS for the Normal distribution where ACS takes the lead with 96.92% compared to 95.20% for TACOS(S).

Table 7.9 – Performance of Each Method for 25-agent CFGs (1 Best – 4 Worst).

Performance	Uniform	Normal	Gamma	Beta	Exponential	Triangular
1	TACOS(S)	ACS	TACOS(S)	TACOS(S)	TACOS(S)	TACOS(S)
2	ACS	TACOS(S)	SA	PSS	SA	ACS
3	PSS	PSS	ACS	ACS	ACS	PSS
4	SA	SA	PSS	SA	PSS	SA

SA and PSS trade places as the lowest performing method for 3 out the total of the six distributions considered.

A comparison of best and worst solutions

To further analyse the overall performance of each method, we will now take a look at the highest and lowest coalition structure value attained by each method for each distribution (see Table 7.10). Highest (Lowest) indicates the highest value attained by a heuristic between all the data sets.

Table 7.10 – Best and Worst Solutions (25-agent CFGs).

25-Agent CFG - Highest and Lowest CS Value found (% of Optimal)												
Distribution	Uniform		Normal		Gamma		Beta		Exponential		Triangular	
Method	Highest	Lowest	Highest	Lowest	Highest	Lowest	Highest	Lowest	Highest	Lowest	Highest	Lowest
TACOS	99.96	99.47	97.02	94.10	91.74	69.12	100	99.99	88.59	57.73	99.70	95.81
SA	99.25	73.85	94.63	91.39	76.85	56.07	99.99	92.42	80.89	41.20	97.40	75.38
ACS	99.61	95.44	99.51	92.83	80.75	51.86	99.99	92.42	82.74	31.66	97.45	84.80
PSS	99.68	91.32	95.61	91.88	77.10	50.97	99.99	97.27	70.97	27.32	97.40	84.03

For the Uniform distribution, the highest quality solution is 99.96% of the optimal value and the lowest is 99.47%. The gap between the highest and lowest value is small demonstrating the ability of TACOS in consistently finding high quality solution. The gap between highest and lowest value found increases with ACS at 4.17% and PSS at 8.36% respectively. The highest gap of 25.40% is shown by SA. Thus SA is the most variational of all the four heuristic methods considered.

ACS returned highest quality solution for the Normal distribution and indeed it is the best performing method for this distribution, slightly outperforming TACOS(S). Although ACS is the highest performing method, the gap between the highest and lowest solution value found is also the highest among all the methods at 6.68%. However, it was still able to outperform all the others possibly due to frequently returning solutions that are closer to the highest value which is already 2.48% higher than the second best method TACOS(S). TACOS(S) still maintains its consistency by having the smallest gap of 2.92%, which is smaller than both SA and PSS at 3.23% and 3.73% respectively. ACS was best for the Normal distribution although the randomness exhibited by the gap between the highest and lowest deserves further attention.

For the Gamma distribution, ACS had the widest gap between the best and worst quality solutions. This was followed by PSS, TACOS and then by SA. The best coalition structure found by SA is just 76.85% of the optimal making it the lowest performing method here although it has the smallest gap. TACOS(S) found the coalition structure with the best value of 91.74% of the optimal.

All the heuristics performed well for the Beta distribution. The best quality solution for all the methods exceeded 99% with TACOS(S) finding the optimal CS in just a few of its runs. TACOS(S) is again the highest performing method for this distribution although the performances of the other algorithms are indistinguishable. ACS again has the largest gap of 7.58% which is slightly more than simulated annealing. TACOS(S) once again attained the smallest gap between the highest and lowest value solution found.

For the Exponential distribution, TACOS(S) returned the best solution at 88.59% of the optimal. The lowest performing method here was PSS. The method with the largest gap of 51.08% was ACS. TACOS(S) once again demonstrating it efficiency by having the smallest

gap of 30.86%. This rather large gap explains why the overall average performance for all these methods (see Table 7.7 and Table 7.8) are lower compared to other distributions.

TACOS(S) again performs best for the Triangular distribution where it found the highest value solution which is 99.70% of optimal. It also has the smallest gap between the highest and lowest value found among all the methods thus demonstrating its consistency in searching through the space and returning high value solution.

7.1.2 Performance for 27-Agent CFGs

For 27-agent CFGs, the average running time was 60 seconds. TACOS was run in a single thread mode denoted TACOS(S) and also in a multi-threaded mode TACOS(M). Each thread simulates an agent running the search. Since searching for the exact optimum was computationally infeasible for 27 agent CFGs, the performance was measured as a percentage of the upper bound.

The Uniform Distribution

For the Uniform distribution (see Table 7.11), both TACOS(S) and TACOS(M) performed better than all the other methods. TACOS(S) and TACOS(M) have almost similar performance.

Table 7.11 – Average Performance for 27-agents CFGs (Uniform Distribution).

Uniform Distribution		
Method	Average Neighbours Explored (≈60 Seconds Running Time)	Average Performance - % of Upper Bound (≈60 Seconds Running Time)
TACOS(S)	310317	99.83
TACOS(M)	286063	99.81
SA	1145346	92.45
ACS	851413	97.48
PSS	331455	96.67

ACS is the second best performing method, followed by PSS while the worst performance comes from SA despite exploring a higher number of neighbouring structures. This is consistent with the findings for the 25-agent CFG.

The Normal Distribution

For the Normal distribution (see Table 7.12), the results for 27 agents are consistent with those for 25-agent CFGs. ACS performed the best, marginally outperforming both TACOS(S) and TACOS(M) by about 2.03% and 1.99% respectively. The two methods at the bottom remain PSS and SA. SA coming in last in terms of overall average solution quality.

Table 7.12 – Average Performance for 27-agents CFGs (Normal Distribution).

Normal Distribution		
Method	Average Neighbours Explored (≈60 Seconds Running Time)	Average Performance - % of Upper Bound (≈60 Seconds Running Time)
TACOS(S)	308268	87.64
TACOS(M)	284456	87.68
SA	1112277	85.65
ACS	818941	89.67
PSS	700302	85.97

The Gamma Distribution

For the Gamma Distribution (see Table 7.13), TACOS(S) and TACOS(M) again performed better than all the other methods. The performance between TACOS(S) and TACOS(M) are within a fraction of each other with TACOS(S) slightly ahead by 0.49%. Third place is ACS followed by PSS in fourth and SA coming in last with an average solution quality that is just 41.75% of the upper bound

Table 7.13 – Average Performance for 27-agents (Gamma Distribution).

Gamma Distribution		
Method	Average Neighbours Explored (≈60 Seconds Running Time)	Average Performance - % of Upper Bound (≈60 Seconds Running Time)
TACOS(S)	309051	63.92
TACOS(M)	283474	63.43
SA	1004789	41.75
ACS	807410	52.27
PSS	367344	47.51

Since none of the methods were able to attain better than 64% solution quality, simulations were conducted with an extended running time of 3 minutes (triple the original running time). The results are shown in Table 7.14.

Table 7.14 – Extended Running Time for 27-agent CFGs (Gamma Distribution).

Gamma Distribution Extended Running Time		
Method	Average Neighbours Explored (≈180 Seconds Running Time)	Average Performance - % of Upper Bound (≈180 Seconds Running Time)
TACOS(S)	921025	67.66
TACOS(M)	861019	65.83
SA	3625934	44.01
ACS	1655885	58.84
PSS	412880	48.72

Although there was an increase in the number of neighbours explored, the increase in average solution quality for each method was small. Thus the performance for the Gamma distribution requires further investigation. Perhaps, a dedicated TACOS-GAMMA algorithm that is optimised only for Gamma can be formulated. In this thesis, the focus is on general-purpose heuristics methods that work well with all distributions.

The Beta Distribution

The results for the Beta distribution are shown in Table 7.15. Consistent with the performance exhibited for the 25-agent CFGs, all heuristics performed well. Four out of five methods returned an excellent average solution quality of over 99%. Even SA returned an overall average solution quality of 97.05% of the upper bound. Both TACOS(S) and TACOS(M) performs exactly the same returning an average solution quality of 99.99% of the upper bound. As with the other distributions, SA was the lowest performing method. However, this time the gap between it and the others is really small, just 2.95% of TACOS(S) and TACOS(M)

Table 7.15 – Average Performance for 27-agents CFGs (Beta Distribution).

Beta Distribution		
Method	Average Neighbours Explored (≈60 Seconds Running Time)	Average Performance - % of Upper Bound (≈60 Seconds Running Time)
TACOS(S)	310065	99.99
TACOS(M)	285016	99.99
SA	1020971	97.05
ACS	836271	99.54
PSS	363258	99.33

The Exponential Distribution

The results for the Exponential distribution are in Table 7.16. With 27-agents, there is a noticeable reduction (relative to the results for 25 agent CFGs) of performance for all methods. TACOS(S) and TACOS(M) remains the highest performing method, however the average quality of solution was 51% of the upper bound. ACS came in second with a huge difference of 37.58%. Both PSS and SA returned a poor sub-30% average solution quality with SA returning the lowest average solution quality of just 22.24% of upper bound.

Table 7.16 – Average Performance for 27-agent CFGs (Exponential Distribution).

Exponential Distribution		
Method	Average Neighbours Explored (≈60 Seconds Running Time)	Average Performance - % of Upper Bound (≈60 Seconds Running Time)
TACOS(S)	313318	51.59
TACOS(M)	285125	51.82
SA	1080725	22.24
ACS	827223	37.58
PSS	314245	29.25

Due to the lower performance, further simulations were conducted with an extended time period three times the original running time. The results are shown in Table 7.17.

Running the algorithms three times longer did not translate into tangible performance increases. All the methods recorded a less than encouraging increase given the extra amount of time to search the space. TACOS(S) and TACOS(M) only recording a 1% to 2% increase despite exploring about three time more neighbours. This performance is similar to the one encountered with the Gamma distribution and deserves attention in future extensions of this research.

Table 7.17 – Extended Running Time for 27-agent CFGs (Exponential Distribution).

Exponential Distribution Extended Running Time		
Method	Average Neighbours Explored (≈180 Seconds Running Time)	Average Performance - % of Upper Bound (≈180 Seconds Running Time)
TACOS(S)	917710	53.01
TACOS(M)	863159	52.73
SA	3084480	28.18
ACS	1660544	38.91
PSS	1011361	35.86

The Triangular Distribution

For the Triangular distribution (see Table 7.18), like the Uniform, Normal and Beta distributions, all methods performed reasonably well exceeding 90% on average. TACOS(S) and TACOS(M) are the best performing methods with both returning an average solution quality of around 97%. ACS was consistently just behind TACOS while SA and PSS both returned an average solution quality of around 91.9%.

Table 7.18 – Average Performance for 27-agent CFGs (Triangular Distribution).

Triangular Distribution		
Method	Average Neighbours Explored (≈60 Seconds Running Time)	Average Performance - % of Upper Bound (≈60 Seconds Running Time)
TACOS(S)	309005	97.82
TACOS(M)	285701	97.75
SA	1048315	91.92
ACS	854167	93.56
PSS	265965	91.99

As with any parallelised implementation, more resources are needed for initialising the search. As such TACOS(M) runs at a lower number of iterations for the same amount of time (see previous chapter for details). Consistent with the simulation results for 25-agents, the for 27-agents exhibit a similar pattern. Both TACOS(S) and TACOS(M) are consistently the best performing method for all other distributions except the Normal distribution (see Table 7.12). For the Normal distribution, ACS very slightly outperformed TACOS(M) by returning an average solution quality that is 1.99% higher (89.67% for ACS vs . 87.68% for TACOS(M)).

Figure 7.2 and Table 7.19 show a combined comparative summary of the results.

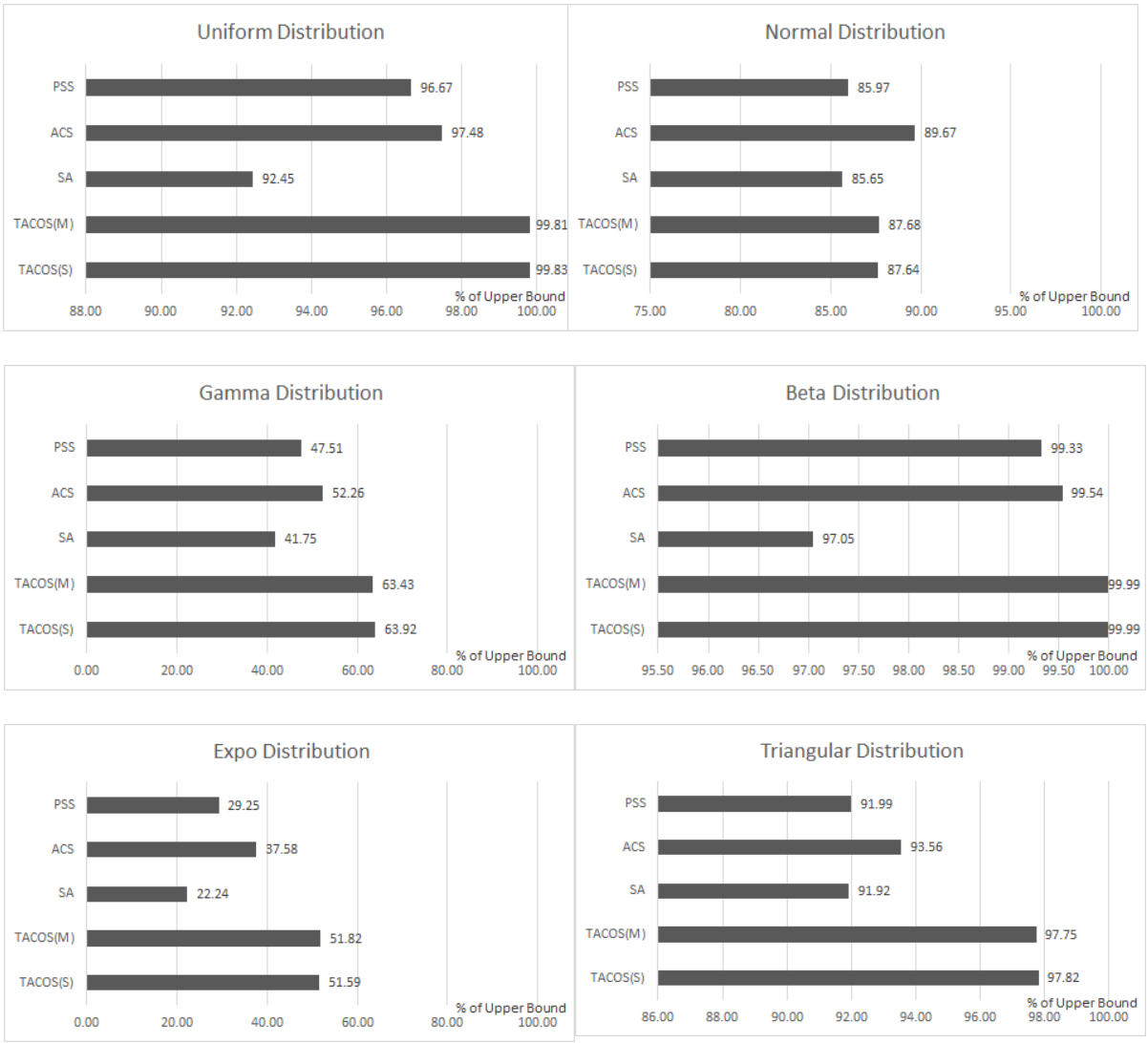

Figure 7.2 – Performance Comparison for 27-agents (CFG) ≈ 60 seconds running time.

Table 7.19 – Performance of Each Method for 27-agent CFGs (1 Best – 5 Worst)

Performance	Uniform	Normal	Gamma	Beta	Exponential	Triangular
1	TACOS(S)	ACS	TACOS(S)	TACOS(S)	TACOS(M)	TACOS(S)
2	TACOS(M)	TACOS(M)	TACOS(M)	TACOS(M)	TACOS(S)	TACOS(M)
3	ACS	TACOS(S)	ACS	ACS	ACS	ACS
4	PSS	PSS	PSS	PSS	PSS	PSS
5	SA	SA	SA	SA	SA	SA

The lower performing methods here are PSS and SA for all distributions with SA consistently returning the lowest performance across all distributions. We shall now take a look at the performance of each method on individual distributions starting with the Uniform distribution.

27-Agent CFG - Highest and Lowest CS Value found (% of Upper Bound)												
Distribution	Uniform		Normal		Gamma		Beta		Exponential		Triangular	
Method	Highest	Lowest	Highest	Lowest	Highest	Lowest	Highest	Lowest	Highest	Lowest	Highest	Lowest
TACOS(S)	99.98	99.64	92.25	82.40	79.47	56.76	100.00	99.99	76.84	41.15	99.25	96.41
TACOS(M)	99.95	99.56	92.42	82.36	79.47	55.43	100.00	99.99	82.45	40.22	99.25	96.18
SA	99.63	81.47	89.54	80.41	56.05	30.43	99.98	81.88	37.32	13.29	94.76	88.83
ACS	99.65	89.79	94.18	84.11	62.15	38.68	100.00	95.67	59.00	22.17	98.65	88.30
PSS	99.63	86.48	90.92	81.41	59.09	36.37	100.00	96.24	46.66	17.08	97.13	84.79

Table 7.20 – Best and Worst Solutions (27-agent CFGs).

A comparison of best and worst solutions

We will now take a look at the performance of each method in terms of the highest and lowest quality solutions found (see Table 7.20).

For the Uniform distribution, TACOS(S) and TACOS(M) showed the smallest gap. This re-affirms the findings from the 25-agent simulations that TACOS consistently returns high quality solutions with less randomness on their quality. SA had the highest gap which explains why it is the lowest performing method in terms of average solution quality. ACS has the third smallest gap while PSS has the fourth. This is consistent with the performance of these methods in terms of average solution quality.

The situation is different for the Normal distribution. All four methods showed the same 10% gap. Recall that only for this distribution ACS is the highest performing method in terms of average solution quality, and indeed as demonstrated here ACS returns the highest solution quality that is 1.76% higher than the nearest competitor TACOS(M). The frequency with which ACS was able to return such high solution may be a contributing factor to its marginally superior 1.99% lead over TACOS(M). PSS and SA remained in the second last and last places respectively. This is consistent with their performance for 27-agent simulations.

For the Gamma distribution, the gap was wider for all the heuristic methods. When compared to the average solution quality, the best is significantly higher than the average. For example, while the highest quality for TACOS(S) for the Gamma distribution is 79.47% of the upper bound, the overall average quality was a mere 67.66%. This shows that while TACOS was on occasions able to discover high quality solutions, the frequency of such high quality solutions being found is much less resulting in a lower overall average of 11.81% shy of the highest quality found. This implies that, for the Gamma distribution, heuristic methods were all effected by hits and misses where the gap is high, resulting in a wider range of solution quality which hampers the overall average performance.

For the Beta distribution, the gaps is smallest for TACOS(S) and TACOS(M), and this is followed by PSS and ACS. For some simulations, TACOS(S) and TACOS(M) returned solutions that is 100% the value of the upper bound. The highest quality found by ACS is slightly higher than the quality of PSS, although the gap is smaller for PSS. ACS was able to perform better than PSS despite its larger gap. As it was with other distributions, ACS was in third place right behind TACOS(S) and TACOS(M).

The results for the Exponential distribution showed a similar pattern to the Gamma distribution. Big gaps exist which hamper the performance of all the methods. Although a single search may return a solution as high as 82.45% such as that returned TACOS(M), the overall average is much. Thus, for this distribution, all the methods return solutions of varying qualities; they sometimes give good results but are mostly bad.

The Triangular distribution is one of the distributions where all the heuristic methods gave good overall performance. TACOS(S) and TACOS(M) has the smallest gap of less than 3%. This puts both of them ahead of all the other methods. ACS comes in just after these two, although as was found in the other distributions, the higher gap for ACS has less influence on

its average performance. It has a higher gap compared to SA but outperforms SA on average. This is also true for PSS. Although SA has a smaller gap, it performed poorly compared to ACS and PSS.

7.2 Performance for Partition Function Games

For Partition Function Games, simulations were run for 10-agents with heuristic solutions compared to the exact optimum. For 27-agents, the heuristic solutions were compared to the upper bound because it is computationally infeasible to find an exact optimum.

The simulations involved 100 runs for each 10 dataset for each distribution. For 10-agents, each method was run for an average of 3 minutes and for 27-agents each method was run for an average of 5 minutes. We will discuss the performance of each method for each individual distribution and then give a combined summary of the main results.

7.2.1 Performance for 10-Agent PFGs

TACOS(S) gave good results and so TACOS(M) was not evaluated.

For the Uniform distribution (see Table 7.21), the average solution quality was above 90% of the optimum. TACOS(S) gave the best average performance with the other methods following closely behind with a 1-2% gap in between them.

Table 7.21 – Average Performance for 10-agent PFGs (Uniform Distribution).

Uniform Distribution		
Method	Average Neighbours Explored (≈180 Seconds Running Time)	Average Performance - % of Optimal (≈180 Seconds Running Time)
TACOS(S)	416	97.35
SA	53666	93.64
ACS	281	95.70
PSS	535	94.42

The Normal Distribution

ACS performed better than other heuristic methods (see Table 7.22). ACS led by a very slim margin of 0.12%. PSS and SA took the third and fourth places respectively.

Table 7.22 – Average Performance for 10-agent PFGs (Normal Distribution).

Normal Distribution		
Method	Average Neighbours Explored (≈180 Seconds Running Time)	Average Performance - % of Optimal (≈180 Seconds Running Time)
TACOS(S)	466	96.96
SA	53168	96.34
ACS	269	97.07
PSS	587	96.63

The Gamma Distribution

The results are as shown in Table 7.23. TACOS(S) and ACS performed reasonably well despite exploring fewer neighbours compared to the other methods. Both PSS and SA, despite exploring more neighbours, performed rather poorly.

Table 7.23 – Average Performance for 10-agent PFGs (Gamma Distribution).

Gamma Distribution		
Method	Average Neighbours Explored (≈180 Seconds Running Time)	Average Performance - % of Optimal (≈180 Seconds Running Time)
TACOS(S)	435	84.77
SA	53206	74.58
ACS	299	81.70
PSS	573	78.70

The Beta Distribution

All methods performed well for 10-agent PFGs. All methods were able to provide an average solution quality that is at least 95% of the optimal (see Table 7.24).

Table 7.24 – Average Performance for 10-agent PFGs (Beta Distribution).

Beta Distribution		
Method	Average Neighbours Explored (≈180 Seconds Running Time)	Average Performance - % of Optimal (≈180 Seconds Running Time)
TACOS(S)	429	99.09
SA	53202	95.50
ACS	282	98.76
PSS	553	97.60

The Exponential Distribution

Exponential distribution continues to be a difficult distribution (see Table 7.25). The best performing method, TACOS(S), was only able to provide an average solution quality of 67.29%. The other methods gave an even lower average solution quality with SA managing just 54.41%. This consistently poor performance for CFGs and PFGs deserves further attention. This is part of future work.

Table 7.25 – Average Performance for 10-agent PFGs (Exponential Distribution).

Exponential Distribution		
Method	Average Neighbours Explored (≈180 Seconds Running Time)	Average Performance - % of Optimal (≈180 Seconds Running Time)
TACOS(S)	440	67.29
SA	53469	54.41
ACS	254	64.28
PSS	516	59.19

The Triangular Distribution

For the Triangular distribution (see Table 7.26), all the methods except SA exceeded 90% average solution quality. TACOS(S) was the best performing method while SA was the poorest but returning an overall average of 88.96%.

Table 7.26 – Average Performance 10-agent PFGs (Triangular Distribution).

Triangular Distribution		
Method	Average Neighbours Explored (≈180 Seconds Running Time)	Average Performance - % of Optimal (≈180 Seconds Running Time)
TACOS(S)	431	93.55
SA	53962	88.96
ACS	261	92.66
PSS	581	91.14

A combined summary of results

Figure 7.3 and Table 7.27 provide a combined summary of the results for 10 agent PFGs.

Table 7.27 – Performance of Each Method for 10-agents PFGs (1 Best – 4 Worst).

Performance	Uniform	Normal	Gamma	Beta	Exponential	Triangular
1	TACOS(S)	ACS	TACOS(S)	TACOS(S)	TACOS(S)	TACOS(S)
2	ACS	TACOS(S)	ACS	ACS	ACS	ACS
3	PSS	PSS	PSS	PSS	PSS	PSS
4	SA	SA	SA	SA	SA	SA

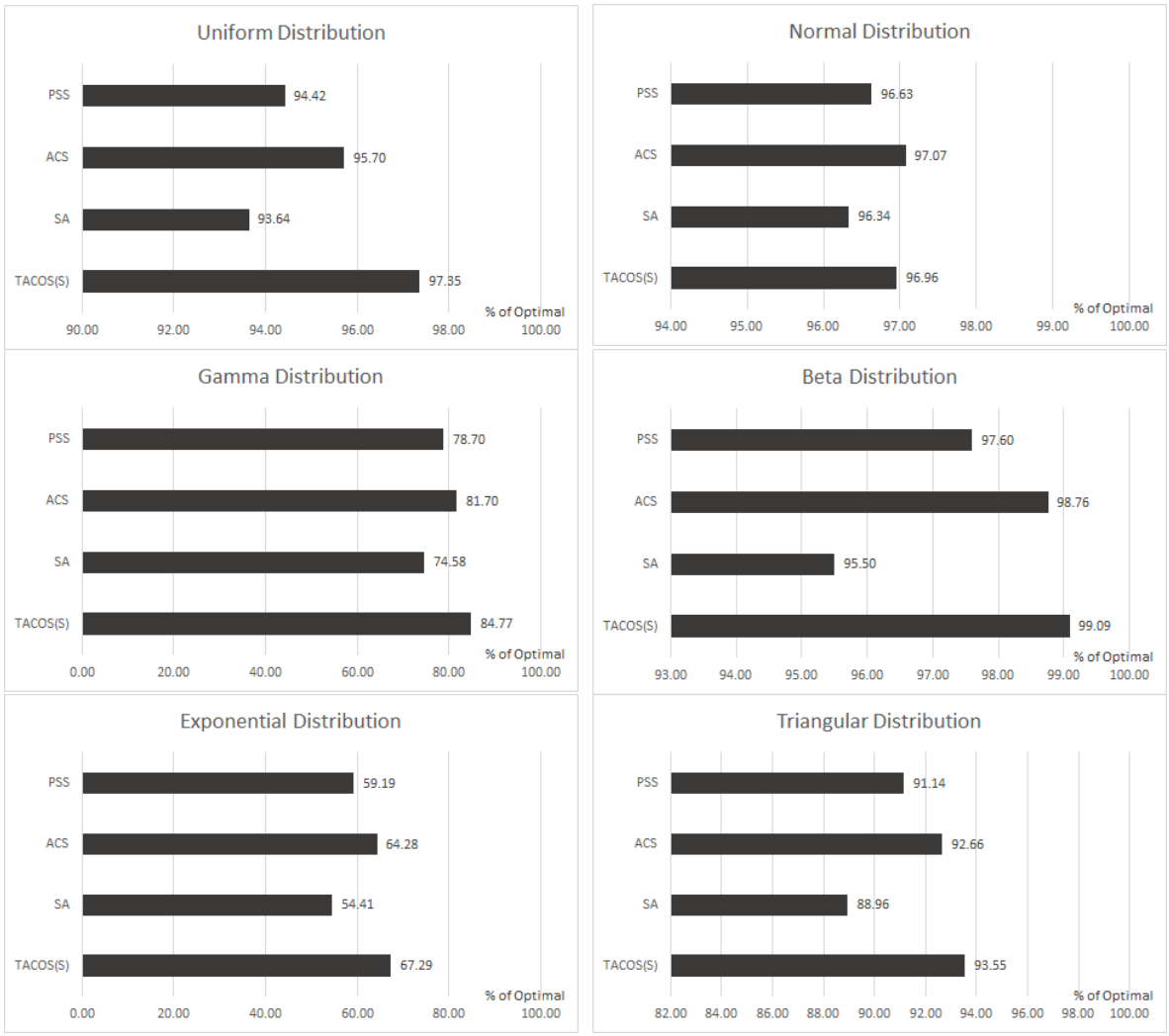

Figure 7.3 – Performance Comparison for 10-agents (PFG) ≈ 180 seconds running time.

TACOS(S) outperformed all other methods for Uniform, Gamma, Beta, Exponential and Triangular distributions. Consistent with other results shown so far, ACS again slightly outperformed TACOS(S) for the Normal distribution (see Figure 7.3). For all other distributions, ACS was the second best method for all distributions followed by PSS and SA (see Table 7.27).

A comparison of best and worst solutions

Next we will take a look at the best and the worst solutions found for each heuristic method by distribution. These results are shown in Table 7.28. For the Uniform distribution, all the methods were able to at one point in their search locate the actual optimum. Although, the gaps between the best and worst meant that the method with the smallest gap, i.e., TACOS(S) outperformed all the others in terms of average solution quality.

Table 7.28 – Best and Worst Solutions (10-agent PFGs).

10-Agent PFG - Highest and Lowest CS Value found (% of Optimal)												
Distribution	Uniform		Normal		Gamma		Beta		Exponential		Triangular	
Method	Highest	Lowest	Highest	Lowest	Highest	Lowest	Highest	Lowest	Highest	Lowest	Highest	Lowest
TACOS(S)	100.00	91.38	99.64	94.77	100.00	69.13	100.00	97.13	100.00	37.78	100.00	86.22
SA	100.00	84.17	99.14	93.74	100.00	56.54	100.00	84.41	81.50	32.35	100.00	75.63
ACS	100.00	84.80	99.14	93.08	100.00	57.12	100.00	95.56	100.00	35.53	100.00	77.40
PSS	100.00	80.95	99.14	93.74	99.51	60.00	100.00	88.63	100.00	30.86	100.00	83.14

For the Normal distribution, the gap was around 5% to 6% for all the methods which is why their performances was indistinguible from each other, with ACS taking a very narrow lead over TACOS(S) (see Table 7.23).

For the Gamma distribution, TACOS(S), SA and ACS were in some instances able to reach the actual optimal solution during the search. However with the gaps of 30% or more, the overall average of each method was much lower. SA which has the highest gap of 43.46% is the worst performing method for the Gamma distribution.

Conversely for Beta, all the methods at some point in the search was able to locate the exact optimal solution and the gaps between the best and the worst solutions were narrower. This contributes to the better average performance of each method for this distribution. Unsurprisingly, SA, the method with the highest gap is the poorest performing method for the Beta distribution.

For the Exponential distribution, TACOS(S), ACS and PSS were able to locate the optimal solution at least once during the course of simulations. However, the gap was between 60-70% for each method. This meant a lack of consistency. With solutions ranging from the highest (100% of the optimal) to the lowest (≈ 30% of the optimal), the average overall

performance suffers as a result. This is true for all the methods for the Exponential distribution.

The Triangular distribution on the other hand was always a heuristic friendly distribution. All the methods including SA were at some point able to find the optimal solution. The small gap between the best and worst solutions meant that all the heuristics performed well for this distribution type (see Table 7.28).

7.2.2 Performance for 27-Agent PFGs

For 27-agents PFGs, performance of the heuristic methods was evaluated with respect to the upper bounds (details regarding the calculation of upper bounds can be found in Chapter 6). The externalities inherent in PFGs make the task of finding a solution that is close to optimum much harder relative to CFGs. Thus each heuristic method was given a longer run time of 5 minutes.

The Uniform Distribution

TACOS(S) and TACOS(M) performed ahead of all the other methods (see Table 7.29 although the average solution quality returned by ACS is just 0.35% lower than TACOS(M). The third best was SA followed by PSS.

Table 7.29 – Average Performance for 27-agent PFGs (Uniform Distribution).

Uniform Distribution		
Method	Average Neighbours Explored (≈300 Seconds Running Time)	Average Performance - % of Upper Bound (≈300 Seconds Running Time)
TACOS(S)	1343787	90.73
TACOS(M)	760977	88.81
SA	959743	67.19
ACS	675131	88.46
PSS	646751	60.48

The Normal Distribution

For the Normal distribution (see Table 7.30), TACOS(S) and TACOS(M) outperformed ACS which had consistently been the best performing method for this distribution for 25 and 27-agent CFGs as well as 10-agent PFGs. For this particular case, this could be due to the larger number of neighbours explored. Both TACOS(S) and TACOS(M) explored more neighbours than ACS.

Table 7.30 – Average Performance for 27-agent PFGs (Normal Distribution).

Normal Distribution		
Method	Average Neighbours Explored (≈300 Seconds Running Time)	Average Performance - % of Upper Bound (≈300 Seconds Running Time)
TACOS(S)	1348962	98.69
TACOS(M)	768815	98.61
SA	955210	82.99
ACS	672218	93.59
PSS	642057	74.19

The Gamma Distribution

As the number of agents increase, the performance of all the heuristics on the Gamma distribution decreases due in part to the higher complexity of the problem due to a larger space and externalities considered (see Table 7.31).

Table 7.31 – Average Performance for 27-agents PFGs (Gamma Distribution).

Gamma Distribution		
Method	Average Neighbours Explored (≈300 Seconds Running Time)	Average Performance - % of Upper Bound (≈300 Seconds Running Time)
TACOS(S)	1341988	56.15
TACOS(M)	765587	54.18
SA	955437	38.14
ACS	680327	55.92
PSS	644015	33.31

Running the search for an increased duration of time did not help increase the overall average by a significant amount. Marginal increases was recorded which re-affirms the need to extend investigation into the peculiarity of the Gamma distribution and its relationship to heuristic performance in future work. One consistent trend is that the order in which each method performed remained the same with TACOS(S) giving a dismal best average of just 58.138% and the lowest performing method remaining to be PSS and returning an average of just 34.876% (see Table 7.32).

Table 7.32 – Extended Running Time for 27-agent PFGs (Exponential Distribution).

Gamma Distribution Extended Running Time		
Method	Average Neighbours Explored (≈600 Seconds Running Time)	Average Performance - % of Upper Bound (≈600 Seconds Running Time)
TACOS(S)	2637194	58.14
TACOS(M)	1498678	55.07
SA	1902333	40.96
ACS	1356438	58.67
PSS	1340154	34.88

The Beta Distribution

For the Beta distribution (see Table 7.33), all four heuristics methods performed well. TACOS(S), TACOS(M) and ACS returned an average solution quality above 90% of the upper bound. SA and PSS trailed behind at 77% and 68% respectively.

Table 7.33 – Average Performance for 27-agent PFGs (Beta Distribution).

Beta Distribution		
Method	Average Neighbours Explored (≈300 Seconds Running Time)	Average Performance - % of Upper Bound (≈300 Seconds Running Time)
TACOS(S)	1344200	98.79
TACOS(M)	764556	97.47
SA	959117	77.11
ACS	683536	94.28
PSS	647730	68.21

The Exponential Distribution

Like the Gamma distribution, the Exponential distribution was difficult for all four heuristic methods to handle. TACOS(S) and ACS lead the table. SA and PSS performs the worst returning a less than 20% average solution quality (see Table 7.34).

Table 7.34 – Average Performance for 27-agens PFGs (Exponential Distribution).

Exponential Distribution		
Method	Average Neighbours Explored (≈300 Seconds Running Time)	Average Performance - % of Upper Bound (≈300 Seconds Running Time)
TACOS(S)	1346263	33.89
TACOS(M)	764497	31.62
SA	952678	16.72
ACS	675094	33.88
PSS	643249	14.06

Extending the running time by twice, enabled all the methods to explore a larger portion of the space (see Table 7.35). However, this still failed to increase the average solution quality by any significant amount. With the extended time, ACS was the only method with a good increase in average solution quality.

Table 7.35 – Extended Running Time 27-agent PFGs (Exponential Distribution)

Exponential Distribution Extended Running Time		
Method	Average Neighbours Explored (≈600 Seconds Running Time)	Average Performance - % of Upper Bound (≈600 Seconds Running Time)
TACOS(S)	2682541	34.40
TACOS(M)	1496641	37.78
SA	1891400	18.82
ACS	1332405	46.71
PSS	1291644	16.29

The Triangular Distribution

With Triangular distribution, TACOS(S), TACOS(M) and ACS were able to exceed 80% average solution quality. However, no method exceeded 90% while both SA and PSS again falling below 70% (see Table 7.36).

Table 7.36 – Average Performance for 27-agent PFGs (Triangular Distribution).

Triangular Distribution		
Method	Average Neighbours Explored (≈300 Seconds Running Time)	Average Performance - % of Upper Bound (≈300 Seconds Running Time)
TACOS(S)	1346947	89.07
TACOS(M)	757745	87.77
SA	953529	67.68
ACS	676111	85.94
PSS	643064	61.21

A combined summary of results

A combined summary of results is given in Figure 7.4 and Table 7.37. TACOS returned a high quality solution for 4 out of the 6 distribution (see Figure 7.4). TACOS(S) was the best performing method for all six distributions.

Figure 7.4 – Performance Comparison for 27-agent (PFG) ≈ 300 seconds running time.

TACOS(M) was the second best method for the Uniform, Normal, Gamma, Beta and Triangular distributions, while ACS took the second best spot for Exponential distribution. The poorest performing method for all distributions was PSS while SA was slightly better than PSS across all distributions (see Table 7.37).

Table 7.37 – Performance of Each Method for 27-agents PFG (1 Best – 5 Worst).

Performance	Uniform	Normal	Gamma	Beta	Exponential	Triangular
1	TACOS(S)	TACOS(S)	TACOS(S)	TACOS(S)	TACOS(S)	TACOS(S)
2	TACOS(M)	TACOS(M)	ACS	TACOS(M)	ACS	TACOS(M)
3	ACS	ACS	TACOS(M)	ACS	TACOS(M)	ACS
4	SA	SA	SA	SA	SA	SA
5	PSS	PSS	PSS	PSS	PSS	PSS

A comparison of best and worst solutions

We shall now evaluate the performance of each method with regard to the best and the worst quality solutions found (see Table 7.38). For the Uniform distribution, TACOS(S) and TACOS(M) returned the best average solution quality and indeed this is consistent with the lowest gap. Although ACS was able to find a better solution, the gap was bigger so that when it comes to the average performance, it lost to TACOS(S) and TACOS(M). Both SA and PSS have huge gaps relegating them to the bottom of the performance spectrum.

For the Normal distribution, TACOS again had the smallest gap proving the consistency of the method in always returning high quality solution. The gap was highest for PSS which is no surprise that it is the poorest performing method for this distribution.

For the Gamma distribution, ACS comes in second after TACOS(S) overtaking TACOS(M) and all the other methods.

For the Beta distribution, TACOS(S), TACOS(M) and ACS were able to find solutions that are 100% of the upper bound. The gap between the best and the worst solutions was lower for all methods except PSS which came in last in terms of performance.

Table 7.38 – Best and Worst Solutions (27-agent PFGs).

27-Agent PFG – Highest and Lowest CS Value found (% of Upper Bound)												
Distribution	Uniform		Normal		Gamma		Beta		Exponential		Triangular	
Method	Highest	Lowest	Highest	Lowest	Highest	Lowest	Highest	Lowest	Highest	Lowest	Highest	Lowest
TACOS(S)	96.33	84.49	98.95	98.14	66.43	42.30	100.00	95.13	47.79	29.13	94.01	84.76
TACOS(M)	94.91	83.13	98.95	98.14	64.92	41.25	100.00	92.46	44.29	18.52	96.48	79.29
SA	80.61	55.18	90.79	82.07	46.54	27.55	88.25	72.32	27.82	7.53	77.71	63.26
ACS	98.48	76.30	99.21	86.91	67.98	40.74	100.00	88.47	53.82	7.53	94.83	76.95
PSS	80.64	43.23	90.74	58.93	48.80	19.65	89.84	51.17	26.12	8.07	78.66	48.52

All the methods performed poorly for the Exponential distribution. Again, investigating this forms part of future work.

For the Triangular distribution, the gap was high for all the methods. However, the good quality solution found by TACOS(S), TACOS(M) and ACS are attributed to these methods more frequently returning higher quality solutions.

7.3 The Effect of Number of Agents on Performance

We will now look at the effect on each method as the number of agent increases. For CFGs, we will compare the performance for 25-agent games with that for 27-agent games. For PFGs, we will compare the performance for 10-agent games with 27-agent games.

7.3.1 Performance for CFGs

TACOS(M) was run only for 27-agent games while TACOS(S) was run for both 25 and 27 agent games, thus comparison will only be made for TACOS(S). We will examine the change in the performance of a heuristic method for each of the six probability distributions.

The Uniform Distribution

For the Uniform distribution, all the methods experience very little degradation in performance. Their performances are almost identical for both 25-agent and 27-agent games taking almost no impact from the increase in the number of agents (see Figure 7.5).

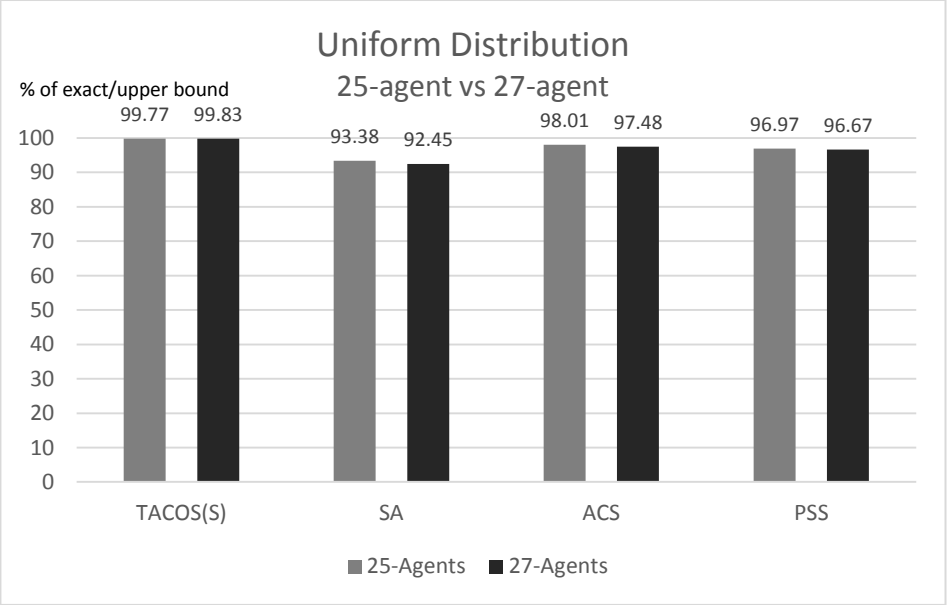

Figure 7.5 – Performance for 25-agent and 27-agent CFGs (Uniform Distribution).

The Normal Distribution

Next we will take a look at Normal distribution. Increasing the number of agents has a slight impact on all the methods (see Figure 7.6). The drop in performance is roughly the same for all four methods.

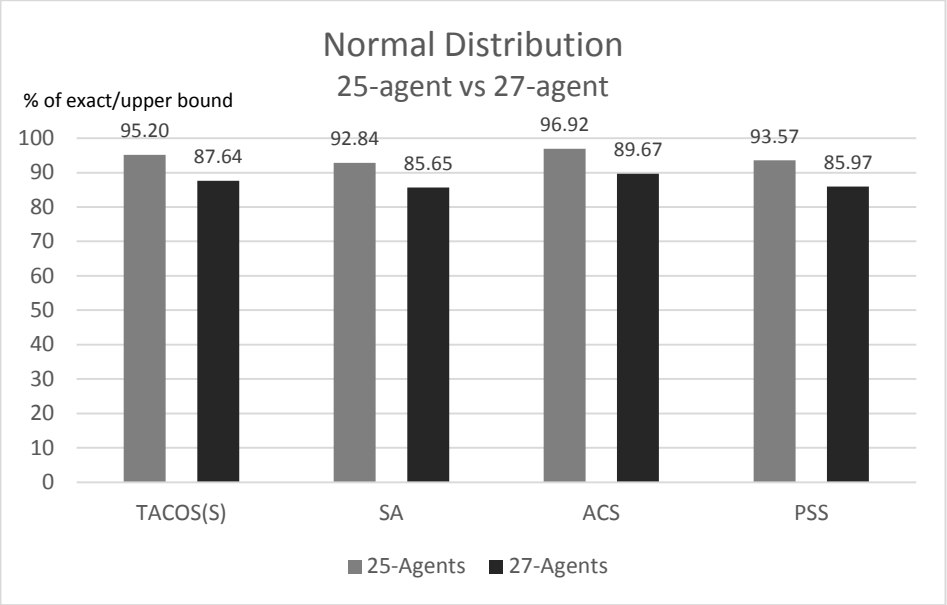

Figure 7.6 – Performance for 25-agent and 27-agent CFGs (Normal Distribution).

The Gamma Distribution

For the Gamma distribution, all the methods experienced a large reduction in performance (see Figure 7.7). SA is the most effected with the largest gap in its performance as the number of agents increases while ACS shows the smallest decline.

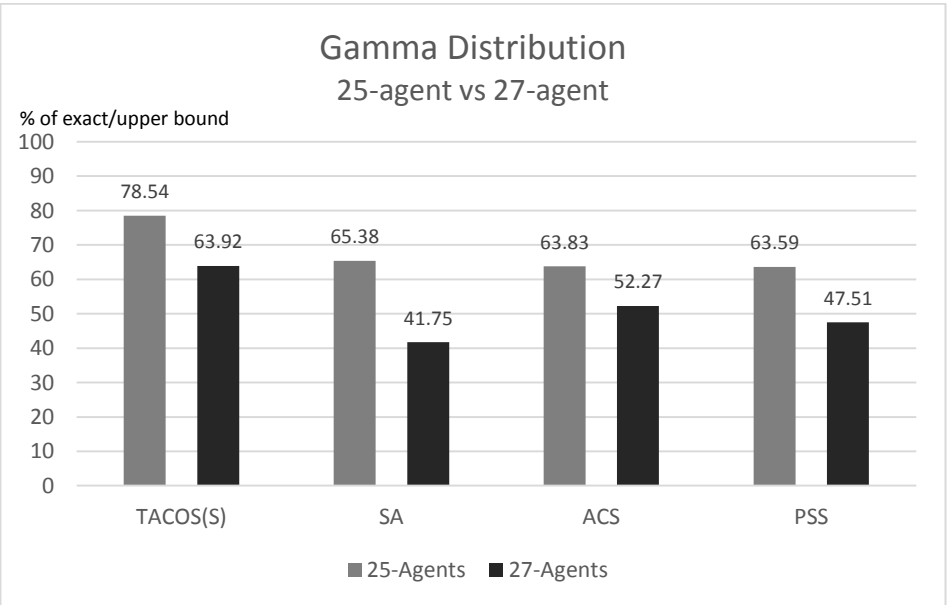

Figure 7.7 – Performance for 25-agent and 27-agent CFGs (Gamma Distribution).

The Beta Distribution

We will now look at the Beta distribution. Similar to the Uniform distribution, the performance of all four methods for Beta distribution is excellent (shown in Figure 7.8). There is barely noticeable degradation in performance from the increase in the number of agents.

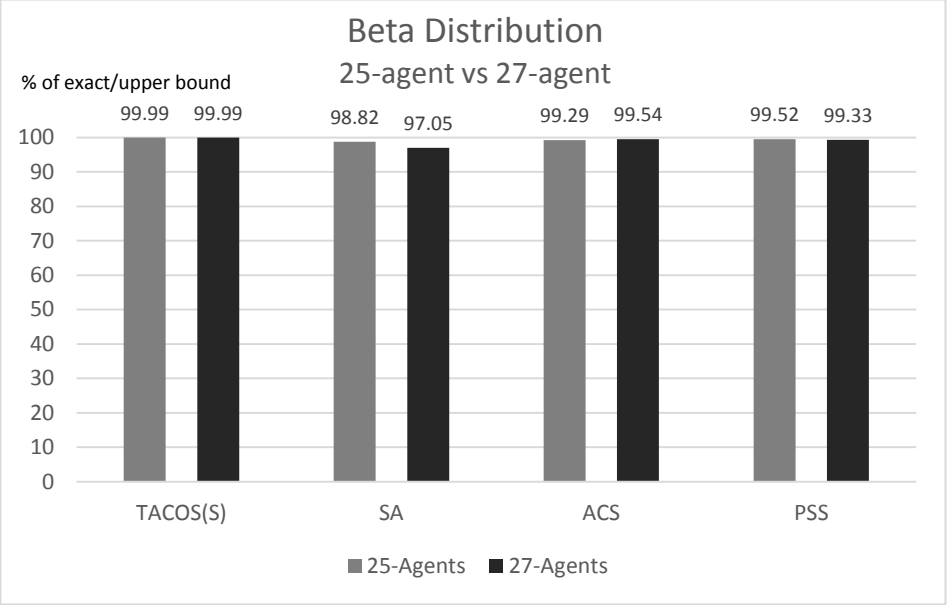

Figure 7.8 – Performance for 25-agent and 27-agent CFGs (Beta Distribution).

The Exponential Distribution

Now, let's take a look at the performance for Exponential distribution. Huge reductions in performance can be seen across all the methods as the number of agents increase (see Figure 7.9). The largest impact again is on SA which see its performance drop by more than 50%.

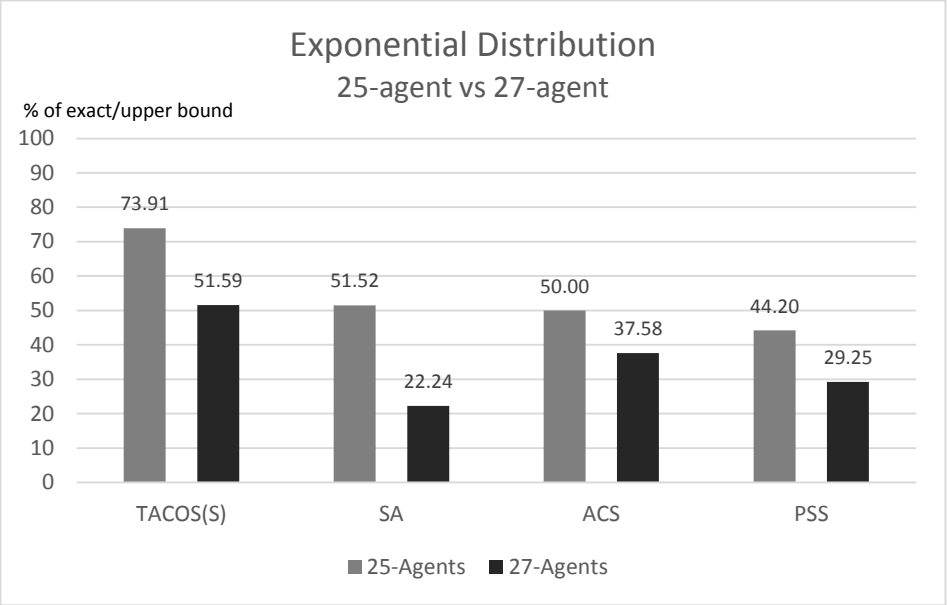

Figure 7.9 – Performance for 25-agent and 27-agent CFGs (Exponential Distribution).

The Triangular Distribution

Figure 7.10 shows the performance for the Triangular distribution. For this distribution, all the method performed very well. TACOS(S) and PSS showing little or no reduction in performance while SA and ACS actually performed better for 27-agent games.

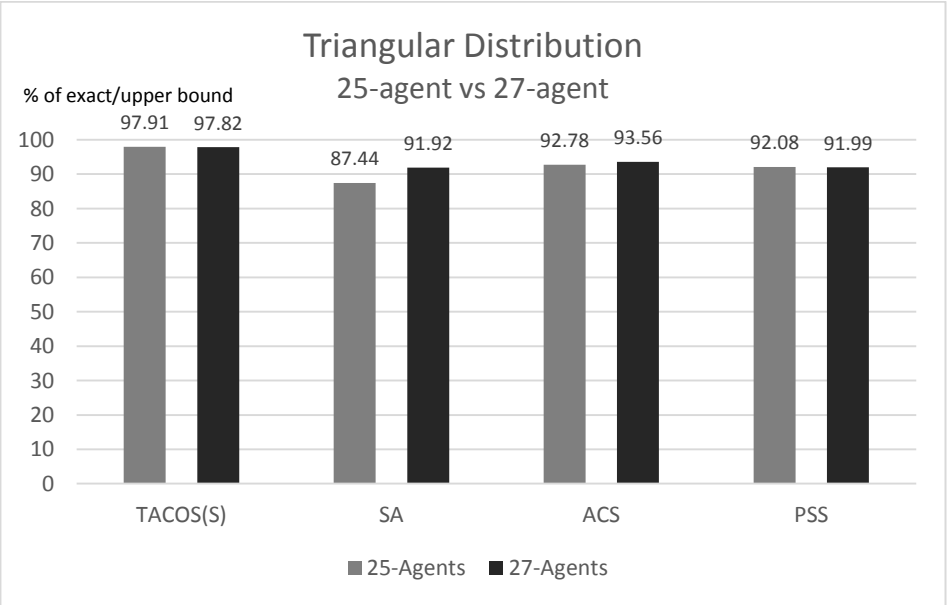

Figure 7.10 – Performance for 25-agent and 27-agent CFGs (Triangular Distribution).

7.3.2 Performance for PFGs

For PFGs we will examine the effect on the performance of a method for 10-agent games and for 27-agent games. Here, TACOS(M) was run for only 27-agent games while TACOS(S) was run for both 10 and 27 agent games. We will examine the change in performance of each method for each of the six probability distributions.

The Uniform Distribution

For the Uniform distribution (see Figure 7.11), the drop in performance with the increase in the number of agents was more for some methods than for others. TACOS(S) showed the lowest impact compared to the other methods. PSS is most effected losing more than 30% of its performance.

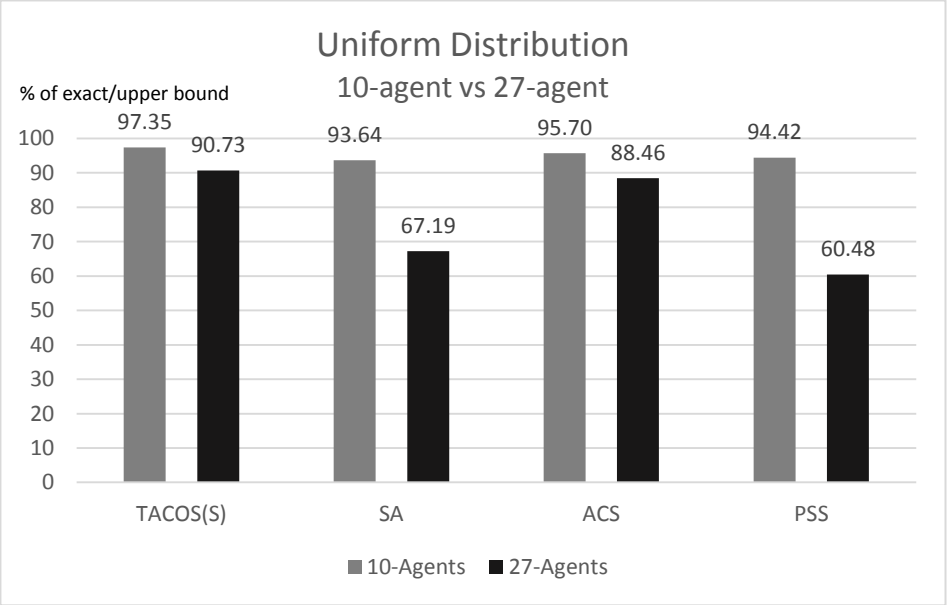

Figure 7.11 – Performance for 10-agent and 27-agent PFGs (Uniform Distribution).

The Normal Distribution

Next we will look at the Normal distribution. For this, all methods except TACOS(S) showed a reduction in performance (shown in Figure 7.12). Surprisingly, TACOS(S) actually performs better for 27-agent games. PSS is the most impacted again, losing its performance in by more than 20%.

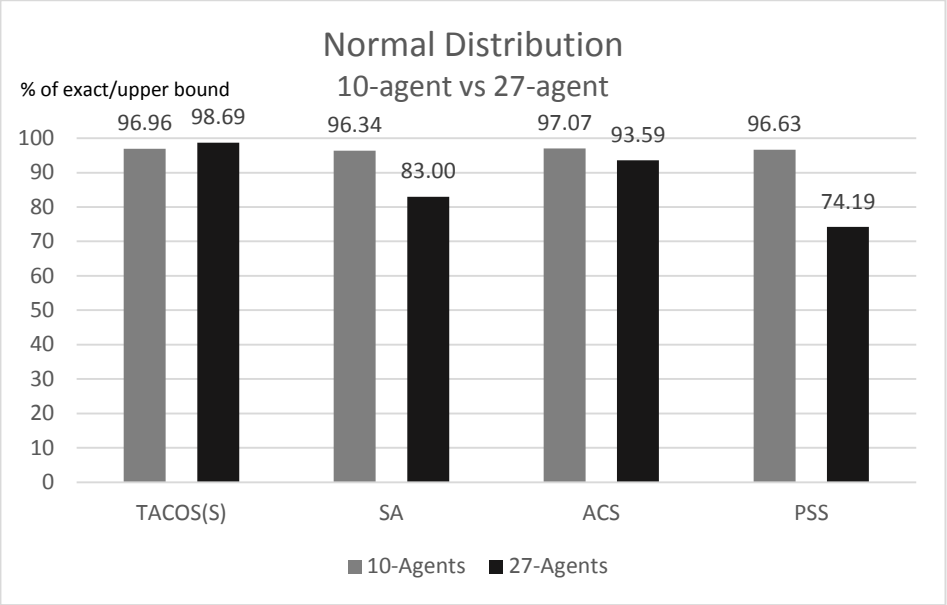

Figure 7.12 – Performance for 10-agent and 27-agent PFGs (Normal Distribution).

The Gamma Distribution

For the Gamma distribution, the impact is quite bad for all the methods as shown in Figure 7.13. Each method lost more than 25% of its performance as the number of agents increased. This means that the Gamma distribution is not a heuristic friendly distribution.

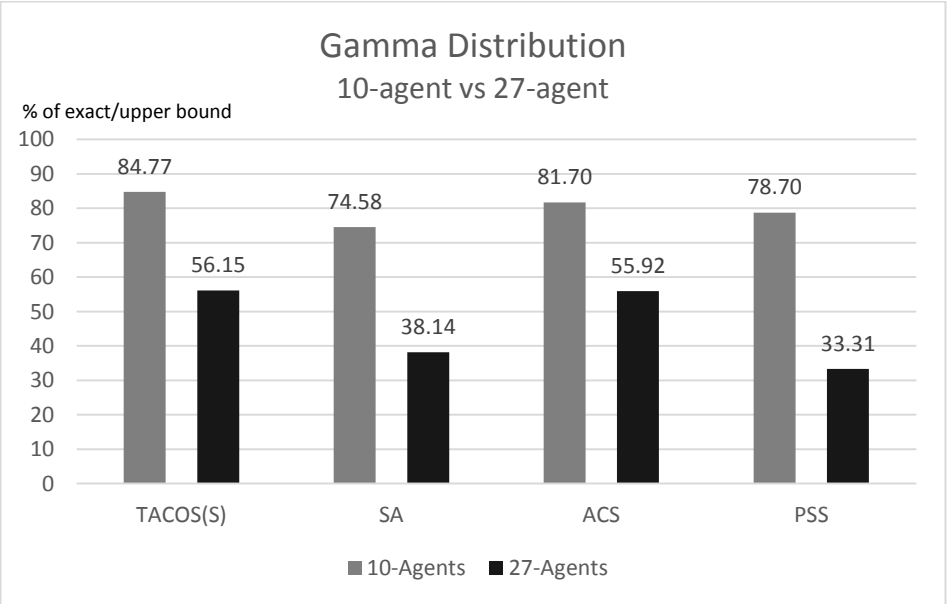

Figure 7.13 – Performance for 10-agent and 27-agent PFGs (Gamma Distribution).

The Beta Distribution

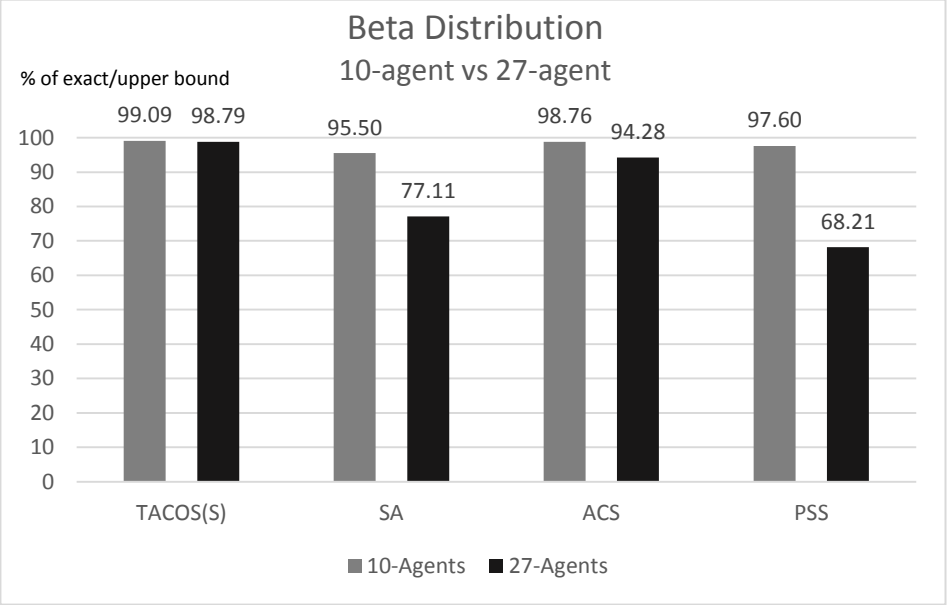

Figure 7.14 – Performance for 10-agent and 27-agent PFGs (Beta Distribution).

Next is the Beta distribution (see Figure 7.14). Here, the performance of TACOS(S) for 10-agent and 27-agent games is consistent with no real reduction seen. ACS showed a very small decrease while SA and PSS again showed the highest reduction in performance.

The Exponential Distribution

None of the methods performed well for 10-agent games for this distribution. Increasing the number of agents to 27 resulted in all the methods performing worse and losing at least 50% of their performance (see Figure 7.15). This pattern echoes the performance on CFGs.

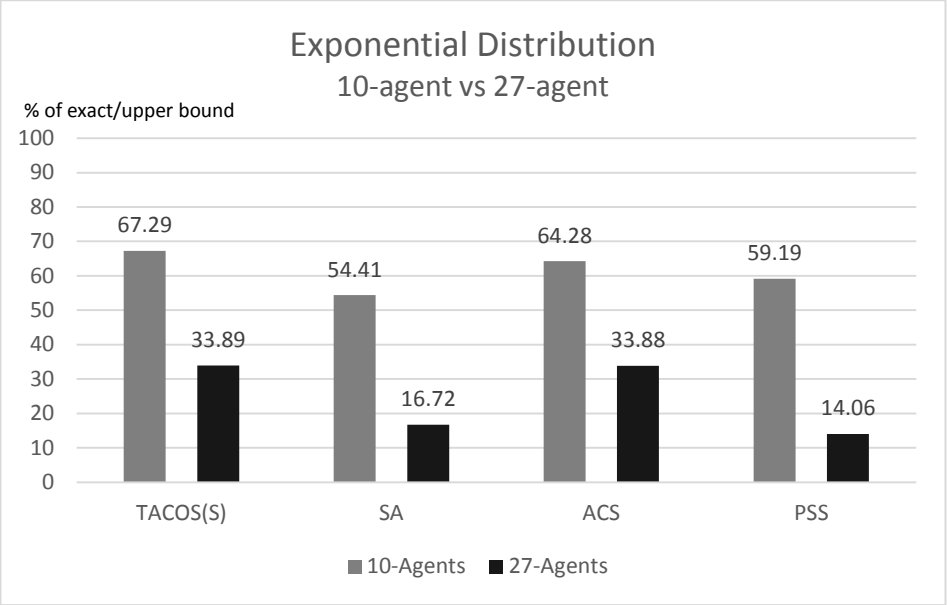

Figure 7.15 – Performance for 10-agent and 27-agent PFGs (Exponential Distribution).

The Triangular Distribution

For the Triangular distribution, the impact is not so bad for TACOS(S) and ACS. Both methods scaled well with smaller degradation in performance as the number of agents increased. As was the case with the other distributions, SA and PSS are the most impacted (see Figure 7.16).

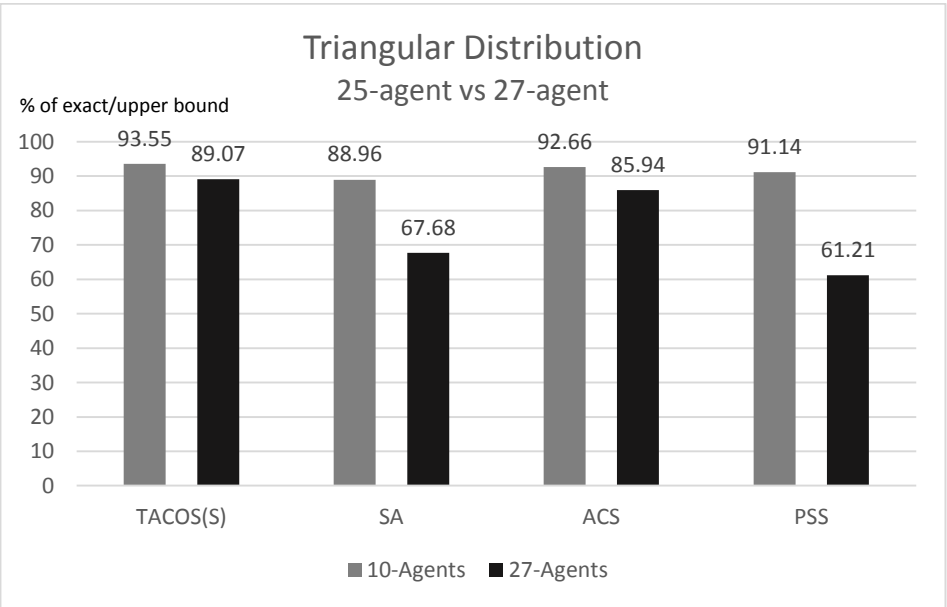

Figure 7.16 – Performance for 10-agent and 27-agent PFGs (Triangular Distribution).

7.4 Statistical Test on Results

To statistically analyse the performance of each method, a statistical test using Confidence Intervals (CI) was done. A confidence interval (also referred to as an interval estimate) is a range (or an interval) of values used to estimate the true value of a population parameter (Triola, 2006). Here we are using CI to compare with the actual average (mean) presented in the results. The purpose is to measure consistency by taking a number samples from the results, calculate the CI then draw some conclusion between the relationships of the values obtained from the CI and the actual mean.

Samples were randomly taken from the results data and CI was calculated for each individual distribution for the four different scenarios (25-agent CFGs, 27-agent CFGs, 10-agent PFGs and 27-agent PFGs). In each scenario the number of samples is n = 33, thus the critical value found using 32 degrees of freedom (where $t_{\alpha/2}$ has $n-1$ degrees of freedom) is $t_{\alpha/2} = 2.037$ such that the margin of error E is calculated as $E = 2.037 \cdot {}^{s}/_{\sqrt{33}}$ where s is the sample standard deviation. The CI is then computed as:

$$\bar{x} - E < \mu < \bar{x} + E$$

Therefore, the confidence interval limit is be between $\bar{x} - E$ and $\bar{x} + E$ where μ denotes the population mean. This limit is being calculated based on a 95% confidence level as this is most commonly used in the majority of studies found in literature (Triola, 2006). We will now take a look at each of the scenarios considered starting with 25-agent CFGs.

7.4.1 Tests on 25-agent CFG

We start with 25-Agent CFGs. The CI for each of the six probability distributions were individually computed.

The Uniform Distribution

For the Uniform distribution we can see that CI limit for TACOS and ACS are the narrowest while SA has the widest limit (see Table 7.39). This is to say that for say, we are 95% confident that the average performance is between 89.65% and 94.64%. The results shown in section 7.1.1 shows that the average solution quality found for SA was 93.38% of the optimal which is within the CI calculated using the randomly chosen samples.

This is true for all the other methods as well, however their CI limit is much smaller compared to SA. A smaller CI limit indicates that these other methods are more consistent as the solutions they found were less varied which led to their higher performance. The order between the widest and narrowest CI limits also follows the pattern of the actual μ found in the results with the method with the smallest interval limit TACOS(S) performing best, followed by ACS, PSS and finally SA. The CI limit for TACOS(S) is very small that as can be seen in Figure 7.17.

Table 7.39 – Confidence Interval 25-Agent CFG (Uniform Distribution).

Method	Descriptive Statistics	95% Confidence Interval
TACOS(S)	$n = 33, \bar{x} = 99.77, s = 0.12$	$99.73 < \mu < 99.81$
SA	$n = 33, \bar{x} = 92.14, s = 7.04$	$89.65 < \mu < 94.64$
ACS	$n = 33, \bar{x} = 97.15, s = 0.87$	$97.79 < \mu < 98.41$
PSS	$n = 33, \bar{x} = 98.10, s = 1.60$	$96.58 < \mu < 97.72$

The Normal Distribution

For the Normal distribution, the ACS has the widest CI limit between 96.10 and 97.10 (see Table 7.40). However because the CI of ACS does not overlap with the CI of any other method, it could suggest that ACS performed differently compared to the others where it found solutions with higher quality that were at least 96% of the optimal. This means that in this particular distribution it was able to outperform TACOS(S) with a CI that has smaller limit but finds solutions whose quality is no higher than 95.33% of the optimal. None of the CI overlap each other and the actual average performance falls within the CI for all the methods.

Table 7.40 – Confidence Interval 25-Agent CFG (Normal Distribution).

Method	Descriptive Statistics	95% Confidence Interval
TACOS(S)	$n = 33, \bar{x} = 95.10, s = 0.66$	$94.86 < \mu < 95.33$
SA	$n = 33, \bar{x} = 92.72, s = 0.76$	$92.45 < \mu < 92.99$
ACS	$n = 33, \bar{x} = 96.60, s = 1.42$	$96.10 < \mu < 97.10$
PSS	$n = 33, \bar{x} = 93.41, s = 1.03$	$93.04 < \mu < 93.78$

The Gamma Distribution

The Gamma distribution was one of the worst performing distributions for all the methods. When calculating the CI, it was found that the CI was wider for all the methods (see Table 7.41). This gap between the lower and upper limit of the CI shows that none of the methods could generate solutions with consistent solution quality. The CI of SA, ACS and PSS overlap each other which explains the similar performance for these methods. TACOS(S) had the smaller CI limit and conversely had the best performance for the gamma distribution here with an average solution quality of 78.54% which is within the CI computed.

Table 7.41 – Confidence Interval 25-Agent CFG (Gamma Distribution).

Method	Descriptive Statistics	95% Confidence Interval
TACOS(S)	$n = 33, \bar{x} = 78.05, s = 4.10$	$76.60 < \mu < 79.51$
SA	$n = 33, \bar{x} = 64.86, s = 6.46$	$62.56 < \mu < 67.15$
ACS	$n = 33, \bar{x} = 64.63, s = 6.33$	$62.38 < \mu < 66.87$
PSS	$n = 33, \bar{x} = 62.95, s = 6.57$	$60.62 < \mu < 65.28$

The Beta Distribution

For the Beta distribution, again the CI for SA, ACS and PSS overlap each other although the CI limit for PSS was narrowest among the three (see Table 7.42). This also corresponds to its overall average performance as found in section 7.1.1 where it outperformed both SA and ACS. TACOS(S) leads the pack with the narrowest CI limit which also corresponds to its overall performance for the Beta distribution in 25-agent CFGs. The CI for TACOS(S) also does not overlap with any other methods indicating a different consistency level compared to the others.

Table 7.42 – Confidence Interval 25-Agent CFG (Beta Distribution).

Method	Descriptive Statistics	95% Confidence Interval
TACOS(S)	$n = 33, \bar{x} = 99.99, s =$0.002	$99.99 < \mu < 100.00$
SA	$n = 33, \bar{x} = 98.72, s = 2.42$	$97.86 < \mu < 99.58$
ACS	$n = 33, \bar{x} = 99.07, s = 0.40$	$98.31 < \mu < 99.84$
PSS	$n = 33, \bar{x} = 99.50, s = 2.16$	$99.36 < \mu < 99.64$

The Exponential Distribution

As it was with the Gamma distribution, the CI limit for the Exponential distribution for each method is also wider (see Table 7.43). This indicates that the samples taken, the estimated μ is within a wider range of values which could be due to the variations of the solution quality found. Unfortunately, a wider CI also meant that the performance for each method was lower for this distribution. TACOS(S) which was the best performing method here managed an average solution quality of 73.91% which is within the CI limits computed.

Table 7.43 – Confidence Interval 25-Agent CFG (Exponential Distribution).

Method	Descriptive Statistics	95% Confidence Interval
TACOS(S)	$n = 33, \bar{x} = 74.75, s = 8.49$	$71.74 < \mu < 77.76$
SA	$n = 33, \bar{x} = 52.76, s = 8.20$	$49.85 < \mu < 55.67$
ACS	$n = 33, \bar{x} = 51.37, s = 9.05$	$48.16 < \mu < 54.57$
PSS	$n = 33, \bar{x} = 45.38, s = 8.69$	$42.30 < \mu < 48.46$

The Triangular Distribution

The Triangular distribution exhibits the same pattern as for SA, ACS and PSS where their CI limit was wider compared to TACOS(S) (see Table 7.44). The CI for ACS and PSS overlap each other which explains their similar performance. Similarly the upper limit of CI for SA was less than 90% which means it is the lowest performing method for this distribution while TACOS(S) has the narrowest CI which means it is also the top performing method on average.

Table 7.44 – Confidence Interval 25-Agent CFG (Triangular Distribution).

Method	Descriptive Statistics	95% Confidence Interval
TACOS(S)	$n = 33, \bar{x} = 97.94, s = 0.71$	$97.69 < \mu < 98.20$
SA	$n = 33, \bar{x} = 87.88, s = 5.74$	$85.84 < \mu < 89.91$
ACS	$n = 33, \bar{x} = 92.98, s = 2.71$	$92.02 < \mu < 93.94$
PSS	$n = 33, \bar{x} = 92.28, s = 3.47$	$91.04 < \mu < 93.51$

Figure 7.17 – Confidence Intervals 25-agent CFGs (% of Optimal).

Figure 7.17 shows the CI for each method in each individual distribution for the 25-agent CFG setting. The next section will take look at 27-agent CFGs.

7.4.2 Tests on 27-agent CFG

For the 27-agent CFG setting, simulations were conducted with both the single-threaded TACOS(S) as well as the multi-threaded TACOS(M). Now let us look at the CI computed for the 27-agent CFGs starting with the Uniform Distribution.

The Uniform Distribution

For the Uniform distribution, the CI limit for SA was the widest (see Table 7.45). Unlike the pattern found with the 25-agent CFG for the Uniform distribution, the CI for ACS and PSS overlap each other indicating similar performance although the actual results found that ACS

slightly outperformed PSS on the overall average. TACOS(S) and TACOS(M) whose CI are narrower compared to the others had the best performance and their respective averages fall within the CI computed here.

Table 7.45 – Confidence Interval 27-Agent CFG (Uniform Distribution).

Method	Descriptive Statistics	95% Confidence Interval
TACOS(S)	$n = 33, \bar{x} = 99.85, s = 0.09$	$99.82 < \mu < 99.88$
TACOS(M)	$n = 33, \bar{x} = 99.83, s = 0.09$	$99.80 < \mu < 99.86$
SA	$n = 33, \bar{x} = 93.47, s = 5.36$	$91.57 < \mu < 95.37$
ACS	$n = 33, \bar{x} = 97.89, s = 2.27$	$96.27 < \mu < 98.69$
PSS	$n = 33, \bar{x} = 97.04, s = 2.18$	$97.09 < \mu < 97.81$

The Normal Distribution

ACS was the best performing method for the Normal distribution (see section 7.1.2) and this average falls within the computed CI. When computing the CI for this distribution it was found that the CI for ACS was the only one that did not overlap any other method. TACOS(S) and TACOS(M) has overlapping CIs, so do SA and PSS. The methods with overlapping CI performs similarly to each other and average for each method still falls within the CI computed here.

Table 7.46 – Confidence Interval 27-Agent CFG (Normal Distribution).

Method	Descriptive Statistics	95% Confidence Interval
TACOS(S)	$n = 33, \bar{x} = 87.56, s = 2.60$	$86.63 < \mu < 88.48$
TACOS(M)	$n = 33, \bar{x} = 87.60, s = 2.59$	$86.68 < \mu < 88.51$
SA	$n = 33, \bar{x} = 85.53, s = 2.47$	$84.66 < \mu < 86.41$
ACS	$n = 33, \bar{x} = 89.56, s = 2.33$	$88.73 < \mu < 90.40$
PSS	$n = 33, \bar{x} = 85.86, s = 2.36$	$85.03 < \mu < 86.68$

The Gamma Distribution

For the 27-agent CFGs, the performance for each method for the Gamma distribution was so good. The CI for each method was wider for this distribution which can be directly contributed by the variations in the solution quality of the samples chosen. The average performance does fall within the computed CI showing the consistency of sampling. However this also means that the wider CI indicates that randomness of the results which means none of the methods was able to consistently find high quality solutions when the input is generated with the Gamma distribution. Table 7.47 shows the CI for each method for the Gamma distribution.

Table 7.47 – Confidence Interval 27-Agent CFG (Gamma Distribution).

Method	Descriptive Statistics	95% Confidence Interval
TACOS(S)	$n = 33, \bar{x} = 63.41, s = 4.58$	$61.78 < \mu < 65.03$
TACOS(M)	$n = 33, \bar{x} = 62.96, s = 5.28$	$61.09 < \mu < 64.83$
SA	$n = 33, \bar{x} = 41.17, s = 5.81$	$39.11 < \mu < 43.23$
ACS	$n = 33, \bar{x} = 52.03\ , s = 4.52$	$50.42 < \mu < 53.63$
PSS	$n = 33, \bar{x} = 47.30, s = 4.53$	$45.70 < \mu < 48.91$

The Beta Distribution

The Beta distribution found the narrowest CI for TACOS(S) and TACOS(M) between 99.98 and 99.99 (see Table 7.48). The rest had a wider CI but overall, this coincides with the actual results and the ranks between the best and worst performing method follows the pattern shown by the CI where the method with a narrower CI generally performing better than those with a wider CI.

Table 7.48 – Confidence Interval 27-Agent CFG (Beta Distribution).

Method	Descriptive Statistics	95% Confidence Interval
TACOS(S)	$n = 33, \bar{x} = 99.99, s$ =0.001	$99.98 < \mu < 99.99$
TACOS(M)	$n = 33, \bar{x} = 99.99, s$ =0.001	$99.98 < \mu < 99.99$
SA	$n = 33, \bar{x} = 97.78, s = 2.34$	$96.95 < \mu < 98.61$
ACS	$n = 33, \bar{x} = 99.72, s = 0.74$	$99.46 < \mu < 99.98$
PSS	$n = 33, \bar{x} = 99.49, s = 0.50$	$99.31 < \mu < 99.67$

The Exponential Distribution

Each method continues to experience a drop in performance for the Exponential distribution. As it was with the Gamma distribution, the CI for each method here was wider. This pattern corresponds to the actual results where the wider the CI, the lower the performance of the method. The average performance for each method fall within the CI shown in Table 7.49.

Table 7.49 – Confidence Interval 27-Agent CFG (Exponential Distribution).

Method	Descriptive Statistics	95% Confidence Interval
TACOS(S)	$n = 33, \bar{x} = 51.63, s = 6.10$	$49.47 < \mu < 53.79$
TACOS(M)	$n = 33, \bar{x} = 51.87, s = 6.77$	$49.47 < \mu < 54.27$
SA	$n = 33, \bar{x} = 22.36, s = 6.16$	$20.18 < \mu < 24.54$
ACS	$n = 33, \bar{x} = 37.96, s = 6.82$	$35.54 < \mu < 40.38$
PSS	$n = 33, \bar{x} = 29.26, s = 5.37$	$27.36 < \mu < 31.17$

The Triangular Distribution

The CI for ACS and PSS overlap each other for the Triangular distribution (see Table 7.50). Although the actual average falls within the CI limit, the results indicates that the two methods did not have similar performance as ACS outperformed PSS. The CI for SA and PSS also overlap, however the CI computed is more indicative of the results as these two methods performed similarly. TACOS(S) and TACOS(M) had the narrowest CI limits which also translates into the actual average performance where they were the top performing methods for this distribution.

Table 7.50 – Confidence Interval 27-Agent CFG (Triangular Distribution).

Method	Descriptive Statistics	95% Confidence Interval
TACOS(S)	$n = 33, \bar{x} = 97.76, s = 0.68$	$97.51 < \mu < 98.00$
TACOS(M)	$n = 33, \bar{x} = 97.66, s = 0.75$	$97.39 < \mu < 97.93$
SA	$n = 33, \bar{x} = 91.21, s = 2.45$	$90.34 < \mu < 92.08$
ACS	$n = 33, \bar{x} = 93.23, s = 2.19$	$92.46 < \mu < 94.01$
PSS	$n = 33, \bar{x} = 91.64, s = 2.83$	$90.64 < \mu < 92.65$

Figure 7.18 shows the CI for each method in each individual distribution for the 27-agent CFG setting. From this figure we can clearly see how the CI of some methods are wider than the others and which methods have their CI overlapping each other.

Figure 7.18 – Confidence Intervals 27-agent CFGs (% of Upper Bound).

The next section will take look at PFGs in which 10-agent and 27-agent PFGs were considered for each distribution.

7.4.3 Tests on 10-agent PFG

Next, we will look at 10-agent PFGs. Only the single-threaded TACOS(S) was ran for the 10-agent PFG setting. CI was computed for each of the six distributions for all four methods. We will look at each of these beginning with the Uniform distribution.

The Uniform Distribution

For the Uniform distribution, SA has the widest CI, this corresponds to its lower performance compared to the other methods (see Table 7.51). TACOS(S) which has the narrowest CI coincidentally performs well. ACS performs better than PSS despite having a wider CI. This is due to the worst solution quality returned by ACS being higher than PSS (see Table 7.28) despite the wider range of results returned by ACS.

Table 7.51 – Confidence Interval 10-Agent PFG (Uniform Distribution).

Method	Descriptive Statistics	95% Confidence Interval
TACOS(S)	$n = 33, \bar{x} = 97.34, s = 1.70$	$96.74 < \mu < 97.94$
SA	$n = 33, \bar{x} = 93.57, s = 4.46$	$91.99 < \mu < 95.15$
ACS	$n = 33, \bar{x} = 96.15, s = 2.83$	$95.15 < \mu < 97.15$
PSS	$n = 33, \bar{x} = 94.61, s = 3.57$	$93.34 < \mu < 95.88$

The Normal Distribution

ACS is the best performing method for the Normal distribution in 10-agent PFG. However the CI computed is found to be wider than TACOS(S) which means the quality of solutions returned by TACOS(S) is more consistent. The CI does give some indication of why ACS performed better than the other. If we look at the overall average performance (the real mean of the results), for ACS the average is towards the upper limit of the CI while for TACOS(S) it leans towards the lower limit of its CI. This means than it returns higher quality solutions than times than TACOS(S) which explains it higher overall average performance.

Table 7.52 – Confidence Interval 10-Agent PFG (Normal Distribution).

Method	Descriptive Statistics	95% Confidence Interval
TACOS(S)	$n = 33, \bar{x} = 96.67, s = 0.98$	$96.32 < \mu < 97.02$
SA	$n = 33, \bar{x} = 96.02, s = 1.08$	$95.64 < \mu < 96.40$
ACS	$n = 33, \bar{x} = 96.86, s = 1.36$	$96.38 < \mu < 97.34$
PSS	$n = 33, \bar{x} = 96.32, s = 1.33$	$95.85 < \mu < 96.79$

The Gamma Distribution

For the Gamma distribution, the CI computed for all the methods are wide (see Table 7.53). This means that none of the methods were consistent in the quality of solutions returned. However, TACOS(S) is the best among the four with a narrower CI. This is consistent with its performance as the best performing method on average for the Gamma distribution. For the rest of the methods, their CI also corresponds their performance and the method with the widest CI (SA) performed worst for the Gamma distribution.

Table 7.53 – Confidence Interval 10-Agent PFG (Gamma Distribution).

Method	Descriptive Statistics	95% Confidence Interval
TACOS(S)	$n = 33, \bar{x} = 83.13, s = 7.82$	$80.36 < \mu < 85.90$
SA	$n = 33, \bar{x} = 72.19, s = 9.66$	$68.76 < \mu < 75.62$
ACS	$n = 33, \bar{x} = 80.20, s = 9.04$	$76.99 < \mu < 83.41$
PSS	$n = 33, \bar{x} = 76.74, s = 9.12$	$73.51 < \mu < 79.97$

The Beta Distribution

All the methods performed well for the Beta distribution (see Figure 7.3), among these TACOS which has the narrower CI (see Table 7.54) is the best performing method. The rest of the method ranks in performance from best to worst according to their interval size with the method that has a wider CI performing worst, in this case it is SA.

Table 7.54 – Confidence Interval 10-Agent PFG (Beta Distribution).

Method	Descriptive Statistics	95% Confidence Interval
TACOS(S)	$n = 33, \bar{x} = 98.82, s = 0.91$	$98.50 < \mu < 99.14$
SA	$n = 33, \bar{x} = 93.87, s = 5.17$	$92.04 < \mu < 95.70$
ACS	$n = 33, \bar{x} = 98.28, s = 1.45$	$97.77 < \mu < 98.79$
PSS	$n = 33, \bar{x} = 96.66, s = 2.93$	$95.62 < \mu < 97.70$

The Exponential Distribution

For the Exponential distribution even for a smaller number of agents, all the heuristics struggled to consistently return higher quality solutions. The only anomaly found from the computed CI is that SA has the narrowest CI (see Table 7.56) despite being the worst performing method among all the methods. This could be due to the random sample taken favouring SA, however the overall average for each method still falls within their computer CI.

Table 7.55 – Confidence Interval 10-Agent PFG (Exponential Distribution).

Method	Descriptive Statistics	95% Confidence Interval
TACOS(S)	$n = 33, \bar{x} = 67.22, s = 12.86$	$62.66 < \mu < 71.78$
SA	$n = 33, \bar{x} = 54.13, s = 12.49$	$49.70 < \mu < 58.56$
ACS	$n = 33, \bar{x} = 64.16, s = 12.89$	$59.59 < \mu < 68.73$
PSS	$n = 33, \bar{x} = 59.70, s = 14.64$	$54.51 < \mu < 64.89$

The Triangular Distribution

TACOS(S) is the best performing method for the Triangular distribution, correspondingly the computed CI for TACOS(S) is the narrowest. Consequently here the method with the widest CI, SA is the worst performing method. ACS and PSS comes in second and third position with their computed CI reflecting the overall average performance.

Table 7.56 – Confidence Interval 10-Agent PFG (Triangular Distribution).

Method	Descriptive Statistics	95% Confidence Interval
TACOS(S)	$n = 33, \bar{x} = 93.73, s = 3.18$	$92.60 < \mu < 94.86$
SA	$n = 33, \bar{x} = 88.86, s = 4.52$	$87.26 < \mu < 90.46$
ACS	$n = 33, \bar{x} = 92.41, s = 3.64$	$91.12 < \mu < 93.70$
PSS	$n = 33, \bar{x} = 90.79, s = 4.16$	$89.31 < \mu < 92.27$

Figure 7.19 shows the CI for each method in each individual distribution for the 10-agent PFG setting, it help to depict how wide the CI for each method is compared to another. We can also see that the CI of some methods overlap each other indicating a similar performance envelope.

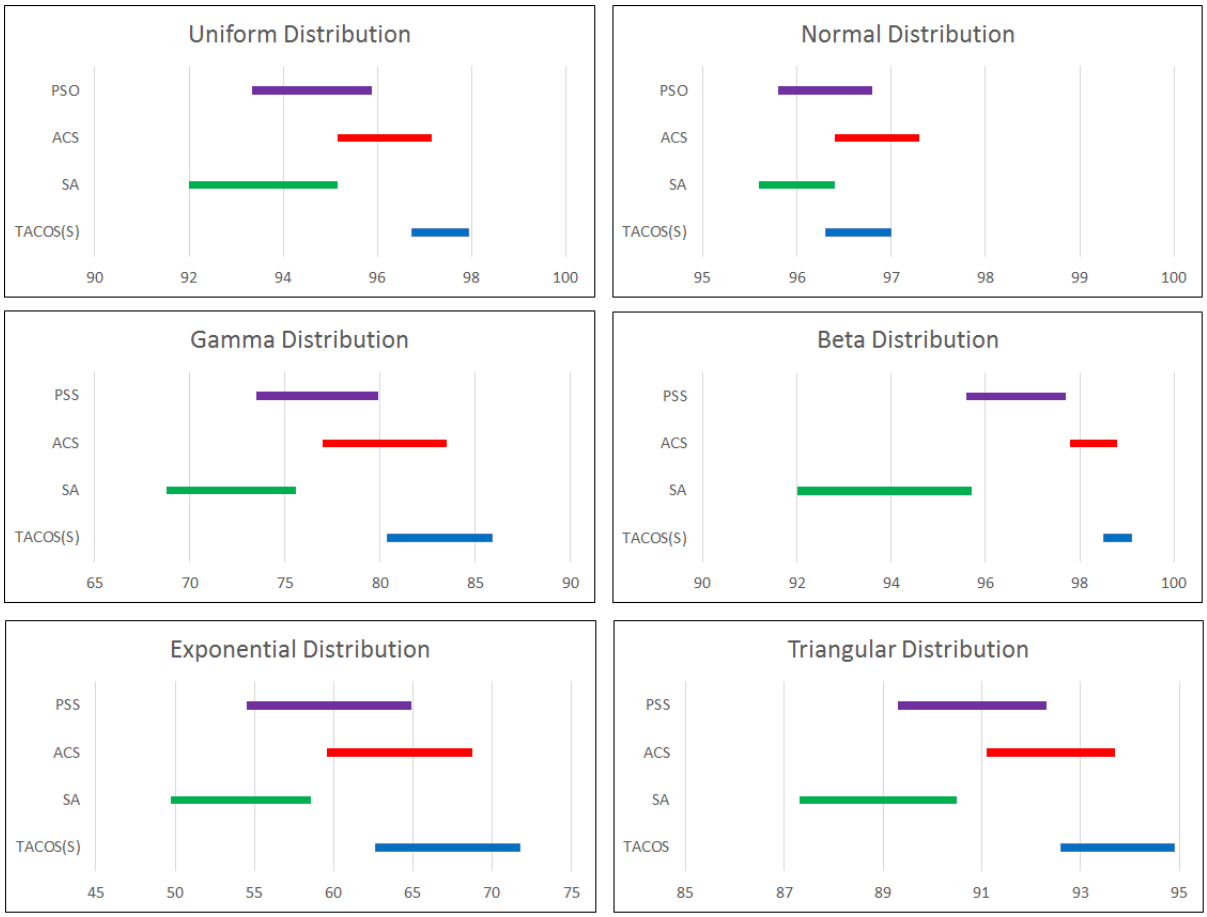

Figure 7.19 – Confidence Intervals 10-agent PFGs (% of Optimal).

7.4.4 Tests on 27-agent PFG

Finally, we calculate the CI for 27-agent PFGs. As this size of the problem is larger for this setting, simulations were conducted for both TACOS(S) and TACOS(M). We will now take a look at the CI computed for each input distribution starting with the Uniform distribution.

The Uniform Distribution

The CI for both TACOS(S) and TACOS(M) is narrower than the other methods for the Uniform distribution. This is followed by ACS. In consequent to this TACOS(S), TACOS(M) and ACS are the top performing methods here (see Figure 7.4). PSS which has the widest CI (see Table 7.57) is the worst performing method.

Table 7.57 – Confidence Interval 27-Agent PFG (Uniform Distribution).

Method	Descriptive Statistics	95% Confidence Interval
TACOS(S)	$n = 33, \bar{x} = 91.03, s = 2.88$	$90.01 < \mu < 92.05$
TACOS(M)	$n = 33, \bar{x} = 89.21, s = 2.91$	$88.18 < \mu < 90.24$
SA	$n = 33, \bar{x} = 68.16, s = 7.10$	$65.65 < \mu < 70.68$
ACS	$n = 33, \bar{x} = 88.79, s = 3.71$	$87.48 < \mu < 90.11$
PSS	$n = 33, \bar{x} = 61.33, s = 8.11$	$58.45 < \mu < 64.20$

The Normal Distribution

For the Normal distribution, TACOS(S) and TACOS(M) has a very narrow CI which can hardly be shown (see Figure 7.20). PSS has the widest CI (see Table 7.59) which corresponds to it performance where it is the worst performing method for the Normal distribution.

Table 7.58 – Confidence Interval 27-Agent PFG (Normal Distribution).

Method	Descriptive Statistics	95% Confidence Interval
TACOS(S)	$n = 33, \bar{x} = 98.62, s = 0.17$	$98.56 < \mu < 98.68$
TACOS(M)	$n = 33, \bar{x} = 98.61, s = 0.24$	$98.52 < \mu < 98.69$
SA	$n = 33, \bar{x} = 82.91, s = 1.40$	$82.41 < \mu < 83.40$
ACS	$n = 33, \bar{x} = 92.76, s = 4.53$	$91.15 < \mu < 94.36$
PSS	$n = 33, \bar{x} = 75.76, s = 8.39$	$72.79 < \mu < 78.74$

The Gamma Distribution

ACS has the widest CI for the Gamma distribution. However it the average performance is better than PSS and SA. From the CI shown in Figure 7.20 we can see that the CI of ACS overlaps with TACOS(S) and TACOS(M). This indicates that it returns higher quality solution similar to TACOS(S) and TACOS(M) which are higher than those returned by PSS and SA which also has wide CIs however they do not overlap with the CI of ACS (see Table 7.59).

Table 7.59 – Confidence Interval 27-Agent PFG (Gamma Distribution).

Method	Descriptive Statistics	95% Confidence Interval
TACOS(S)	$n = 33, \bar{x} = 57.42, s = 3.70$	$56.11 < \mu < 58.73$
TACOS(M)	$n = 33, \bar{x} = 55.51, s = 4.02$	$54.08 < \mu < 56.93$
SA	$n = 33, \bar{x} = 39.46, s = 3.99$	$38.04 < \mu < 40.87$
ACS	$n = 33, \bar{x} = 57.43, s = 4.75$	$55.74 < \mu < 59.11$
PSS	$n = 33, \bar{x} = 34.75, s = 4.57$	$33.13 < \mu < 36.38$

The Beta Distribution

For the Beta distribution, PSS has the widest CI compared to the other methods. Moreover, the solutions returned are considerably worse compared to the other method. This wider CI also means that the range of solution quality returned is erratic and non-consistent. ACS which has a narrower CI compared to SA, performed just behind TACOS(S) and TACOS(M).

Table 7.60 – Confidence Interval 27-Agent PFG (Beta Distribution).

Method	Descriptive Statistics	95% Confidence Interval
TACOS(S)	$n = 33, \bar{x} = 98.46, s = 1.29$	$98.00 < \mu < 98.92$
TACOS(M)	$n = 33, \bar{x} = 97.06, s = 1.44$	$96.55 < \mu < 97.57$
SA	$n = 33, \bar{x} = 76.21, s = 3.03$	$75.14 < \mu < 77.28$
ACS	$n = 33, \bar{x} = 93.63, s = 2.29$	$92.82 < \mu < 94.44$
PSS	$n = 33, \bar{x} = 66.53, s = 5.53$	$64.57 < \mu < 68.49$

The Exponential Distribution

The widest CI for the Exponential distribution was found to be for ACS. The random sample taken returned a wide CI for ACS although the average performance falls in the middle of the upper and lower CI limit. This indicates that the quality of solution given by ACS is highly inconsistent and involves extremes of high and low values. TACOS(S) and TACOS(M) still have the narrower CIs compared to the others (see Table 7.61) and are the best performing methods here. However the wide CI for ACS overlaps with TACOS(S) and TACOS(M) which shows that its performance falls with the performance of these two.

Table 7.61 – Confidence Interval 27-Agent PFG (Exponential Distribution).

Method	Descriptive Statistics	95% Confidence Interval
TACOS(S)	$n = 33, \bar{x} = 33.29, s = 5.90$	$31.20 < \mu < 35.38$
TACOS(M)	$n = 33, \bar{x} = 31.46, s = 5.76$	$29.42 < \mu < 33.50$
SA	$n = 33, \bar{x} = 16.77, s = 4.82$	$15.06 < \mu < 18.48$
ACS	$n = 33, \bar{x} = 34.00, s = 11.09$	$30.07 < \mu < 37.93$
PSS	$n = 33, \bar{x} = 14.06, s = 3.70$	$12.75 < \mu < 15.37$

The Triangular Distribution

PSS has the widest CI for the Triangular distribution and it does not overlap with any of the other methods. Its upper limit is also much lower than the second worst performing method, SA. The average performance of each method falls within the computed CI shown in Table 7.62. TACOS(M) has a rather wide CI however it overlaps with that of TACOS(S) indicating their average performance should be similar of which they are.

Table 7.62 – Confidence Interval 27-Agent PFG (Triangular Distribution).

Method	Descriptive Statistics	95% Confidence Interval
TACOS(S)	$n = 33, \bar{x} = 88.94, s = 2.07$	$88.21 < \mu < 89.67$
TACOS(M)	$n = 33, \bar{x} = 87.65, s = 3.45$	$86.43 < \mu < 88.87$
SA	$n = 33, \bar{x} = 67.55, s = 3.19$	$66.42 < \mu < 68.68$
ACS	$n = 33, \bar{x} = 85.84, s = 3.04$	$84.76 < \mu < 86.92$
PSS	$n = 33, \bar{x} = 60.98, s = 6.82$	$58.56 < \mu < 63.39$

Figure 7.20 shows the CI for each method in each distribution for the 27-agent PFG setting. It can be seen from the figure below that PSS consistently have a wide CI. Surprisingly SA exhibits a consistently narrow CI across the distributions. However, its perform within its own performance envelope where the upper limit in each distribution is much lower than the lower limits of TACOS(S), TACOS(M) and ACS.

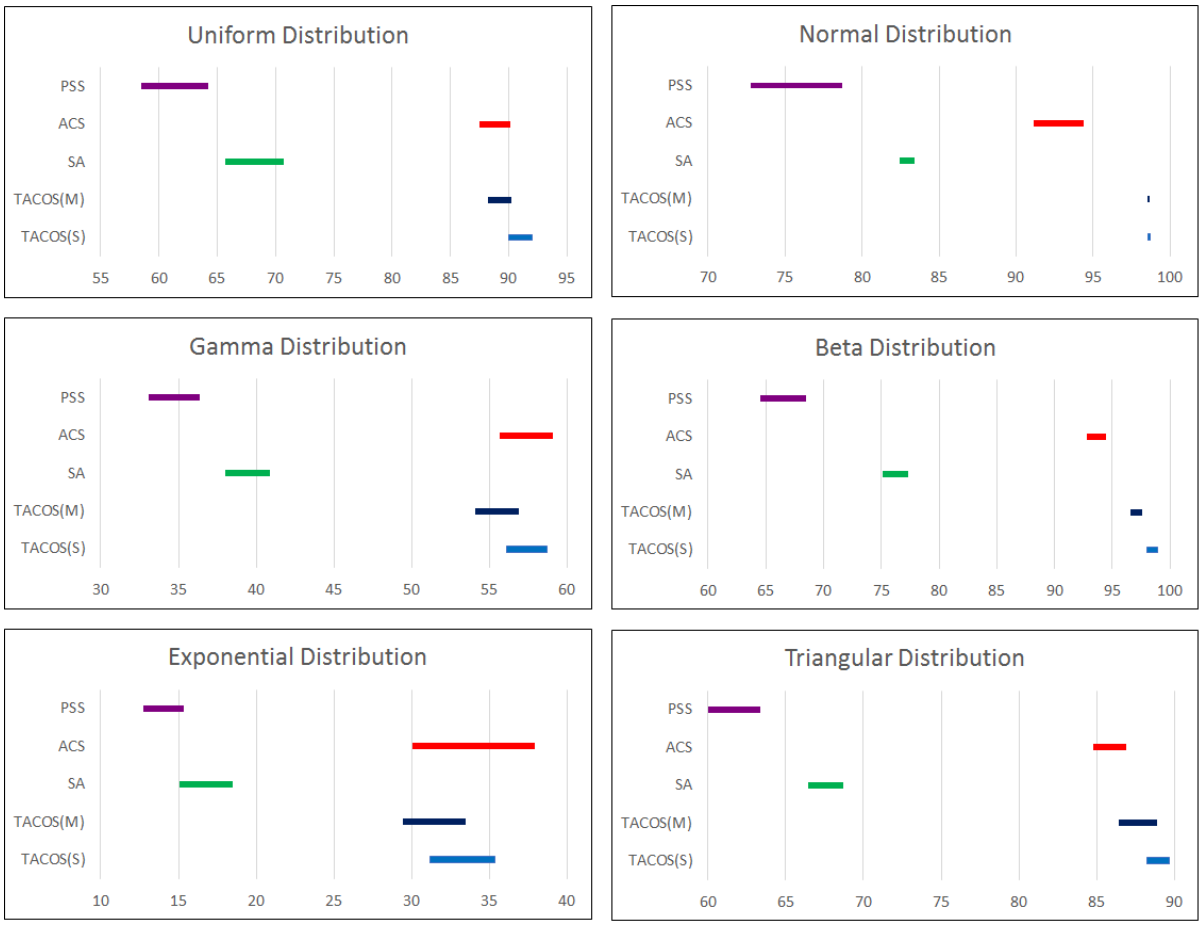

Figure 7.20 – Confidence Intervals 27-agent PFGs (% of Upper Bound).

It must be noted however, while we were able to draw certain conclusions from the comparison between the various CIs calculated, these should only be taken as tentative indications. To make definitive conclusions would require more involved statistical testing which is outside the scope of this research but is worthy of consideration in future work.

7.5 Chapter Summary

This chapter presented the results of the simulations and discussed the performance of each of the four heuristic methods for each of the six probability distributions. For CFGs, the solution quality was measured relative to the exact optimum for 25 agent games and to the upper bound on the exact optimum for 27 agent games. For PFGs, solution quality was measured relative to the exact optimum for 10 agent games and to the upper bound on the exact optimum for 27 agent games.

In terms of the ranking between the methods, In CFGs, TACOS is the best ranking method in the majority of distributions tested with the exception of the Normal Distribution where ACS was best. In PFGs for the smaller problem size of 10-agents, the same pattern follows the CFG ranking, TACOS is the best ranking method in the majority of distributions while ACS was best performing for the Normal distribution. TACOS(S) was the overall best performing methods for the larger PFG games in all distributions. In summary, for CFG, TACOS was the best method for over 83% of the time while for PFG it was 91% of the time the best performing method. This makes TACOS the highest ranking method among the 4 in both CFG and PFGs.

When comparing the impact of an increase in the problem size, i.e. the number of agents, we noticed stark differences between CFGs and PFGs. For CFGs, increasing the number of agents had a much lower impact on the reduction in the performance of each method. This is true except for SA whose performance was severely impacted for the Gamma and Exponential distributions. Remarkably, its performance increased for the Triangular distribution as the number of agents increased from 25 to 27 agents.

For PFGs, as the complexity of the problem increases (a larger problem space resulting from the more agents), noticeable decrease in performance can be see for all the methods. When comparing the performance for 10-agent and 27-agent PFGs, TACOS and ACS suffered the least impact for each individual distribution. The performances of SA and PSS were severely reduced for each distribution type with PSS being the method most effected by the increase in the number of agents. In the next chapter, we attempt to analyse the average performance of each method across all the distributions.

Chapter 8 Performance Analysis Across Distributions

As is now clear from Chapter 7, the performance of heuristic methods depends on the probability distribution from which the values of coalitions are drawn. Chapter 7 gave a detailed analysis of the performance of each heuristic method for each of the six probability distributions. However, it is also important to know how the performance of these methods compares, on average, across all the six probability distribution. Further, in order to fully understand the performance of heuristics, we need to know how well they can scale to larger games. Another consideration is the impact of externalities on the performance of heuristic methods. To this end, the objectives of this chapter are as follows:

1. To examine the performance of the heuristic methods averaged over all the six probability distributions. This analysis is important because it is conceivable that the values of coalitions are drawn from various different distributions and not just one single distribution. Note that all the four heuristic methods (tabu search, simulated annealing, ant colony, and particle swarm) are oblivious to the probability distribution from which the values of coalitions are drawn.
2. To examine the scalability of the four heuristic methods. The practical value of any heuristic method depends on its scalability. It is therefore important to know which heuristics scale well and which ones do not.
3. To analyse how externalities affect the performance of heuristics. The introduction of externalities makes the search problem considerably harder. It is therefore important to understand the extent to which externalities impact the performance of heuristic methods.

This chapter is organised as follows. Section 8.1 is a comparative analysis of the average performance of the four heuristic methods for CFGs. This analysis is conducted first for 25 agent games and then for 27 agent games. Section 8.2 is a comparative analysis of the average performance of the four heuristic methods for PFGs. This analysis is conducted first for 10

agent games and then for 27 agent games. Section 8.3 shows the effect of the number of agents and the effect of externalities on the performance of a heuristic. Section 8.4 describes the memory usage.

8.1 Average Performance for CFGs

We will first compare the average performance of the heuristics for 25 agent CFGs. The performance of each of the four heuristics, averaged across all the six distributions, is shown in Figure 8.1.

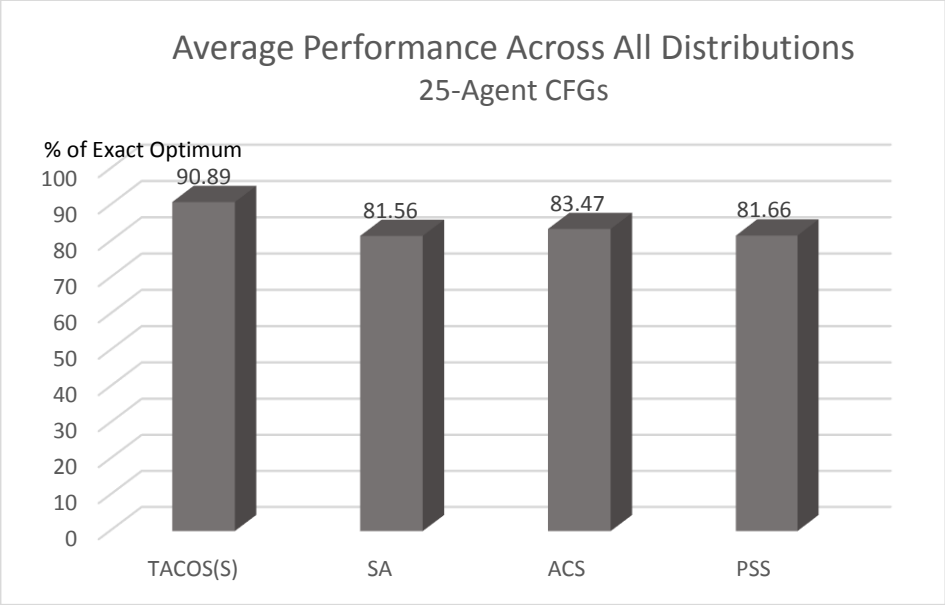

Figure 8.1 – Average performance (25-agent CFGs).

TACOS(S) is the best performing method with an average solution quality was about 91% of the exact optimum. There are two reasons for best performance of TACOS(S). First, although TACOS(S) performed best for the Beta distribution (just like the other three heuristic methods), the gap between the best and worst solutions (see Table 8.1), is smallest for TACOS(S). This gap is around 26% (99.9 – 73.9 as shown in row 1 of Table 8.1). After TACOS(S), ACS is second followed by PSS and then SA. PSS and SA have nearly identical performance.

Table 8.1 –Best and worst performance across distributions (25-agent CFGs).

Method	Best Performance	Value – % of Optimal	Worst Performance	Value – % of Optimal
TACOS(S)	Beta	99.99	Exponential	73.91
SA	Beta	98.82	Exponential	51.52
ACS	Beta	99.29	Exponential	49.99
PSS	Beta	99.52	Exponential	44.20

In terms of the numbers of neighbours explored (see Table 8.2), SA explored the most number of neighbours. PSS, although exploring a lot less neighbours was able to match SA in performance. On the other hand, while ACS explores more neighbours than TACOS(S), the latter still performed better than ACS. This superior performance of TACOS is attributed to its memory; it uses memory to avoid repeatedly visiting the same points in the search space.

Table 8.2 – Average number of neighbours explored (25-agent CFGs).

Method	Average neighbours explored
TACOS(S)	203848
SA	1440010
ACS	305759
PSS	203905

Next we will take a look at the performance for 27-agent games for which TACOS was run in single-threaded TACOS(S) mode as well as the multi-threaded TACOS(M) mode. Figure 8.2 shows the average performance for all the four heuristics. Consistent with the results for the 25-agent games, both TACOS(S) and TACOS(M) were ahead of the other methods. This is followed by ACS, then PSS and then SA. Unlike the 25-agent games, the difference between the performance of SA and PSS is noticeable (it was almost identical for 25-agents). TACOS(S) and TACOS(M) gave identical performance with very negligible differences in overall average.

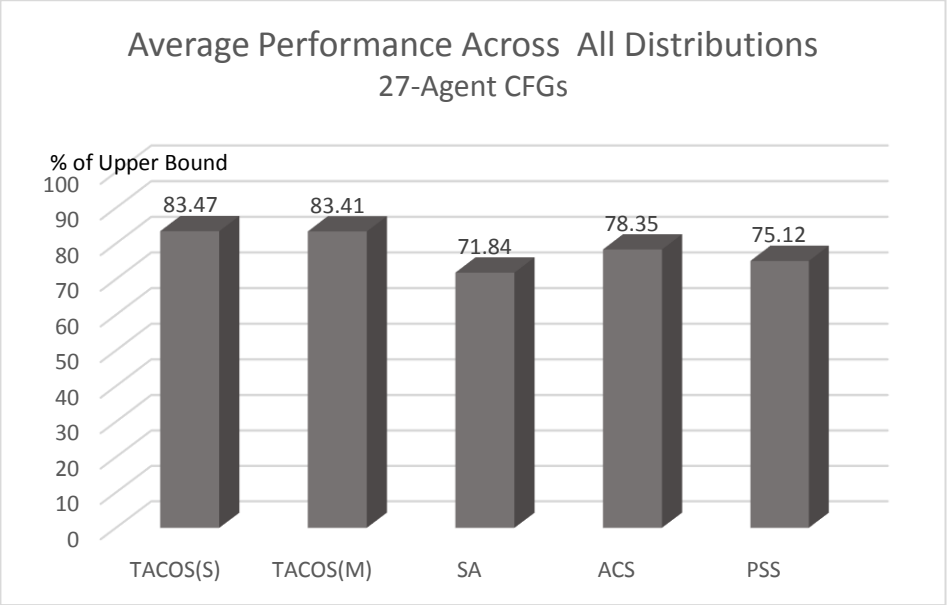

Figure 8.2 – Average performance (27-agent CFGs).

As before, there is a link between average performance and the the gap between the best and worst solutions found by a method. Table 8.3 shows the distribution for which each method performed best and also the distribution for which each method performed the worst. As with 25 agent games, each of the four heuristic methods performed best for the Beta distribution and worst for the Exponential distribution. TACOS(S) and TACOS(M) have the smallest gap. The gap is widest for the worst performing method, i.e., SA.

Table 8.3 – Best and worst performance across distributions (27-agent CFGs).

Method	Best Performance	Value – % of Upper Bound	Worst Performance	Value – % of Upper Bound
TACOS(S)	Beta	99.99	Exponential	51.59
TACOS(M)	Beta	99.99	Exponential	51.82
SA	Beta	97.05	Exponential	22.24
ACS	Beta	99.54	Exponential	37.58
PSS	Beta	99.33	Exponential	29.25

As was the case with 25-agent games, the method that explored the maximum number of neighbours was SA (see Table 8.4). Although PSS explored the smallest number of neighbours but it performed slightly ahead of SA. On the other hand ACS explorer twice as

many neighbours compared to TACOS(S) and TACOS(M), however this did not translate into an advantage for the method as its performance across all distributions is lower than them.

Table 8.4 – Average number of neighbours explored (27-agent CFGs).

Method	Average number of neighbours explored
TACOS(S)	310004
TACOS(M)	284973
SA	1068737
ACS	832571
PSS	390428

8.2 Average Performance for PFGs

The average performance for PFGs was measured for 10-agent and 25-agent games. We will start by looking at the 10-agent games. The performance of each method averaged across the six distributions is shown in Figure 8.3.

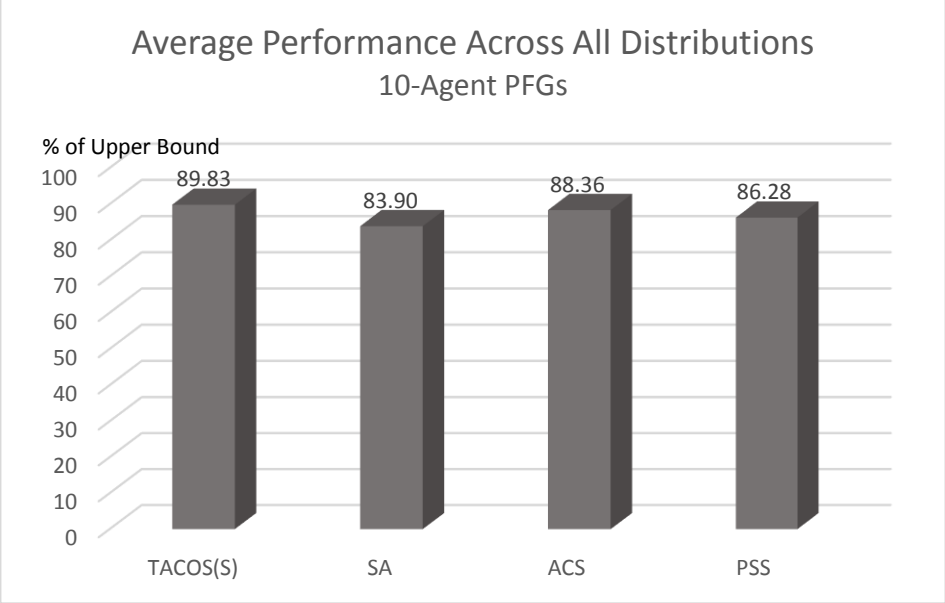

Figure 8.3 – Average performance (10-agent PFGs).

TACOS(S) gave the best average performance, followed by ACS, then PSS and then SA. TACOS(S), ACS and PSS formed best for the Beta distribution (see Table 8.5) while SA performed best for the Normal distributions. All heuristics performed worst for the Exponential distribution.

Table 8.5 – Best and worst Performance across distributions (10-agent PFGs).

Method	Best Performance	Value – % of Optimal	Worst Performance	Value – % of Optimal
TACOS(S)	Beta	99.09	Exponential	67.29
SA	Normal	96.34	Exponential	54.41
ACS	Beta	98.76	Exponential	64.28
PSS	Beta	97.60	Exponential	59.19

Here too, best performance corresponds to smallest gap between the best and worst solutions.

With regard to the number of neighbours, TACOS(S) explored the fewest neighbours and achieved the best performance. ACS explored fewer neighbours compared to PSS and SA, but performed better than both these methods (see Table 8.6).

Table 8.6 – Average number of neighbours explored (10-agent PFGs).

10-Agent	Average neighbours explored
TACOS(S)	203848
SA	1440010
ACO	305759
PSO	203905

The average performance for 27-agent games is shown in Figure 8.4. Surprisingly, it was TACOS(S) that took the lead. In this case, more iterations allowed TACOS(S) to take the lead although TACOS(M) was very close.

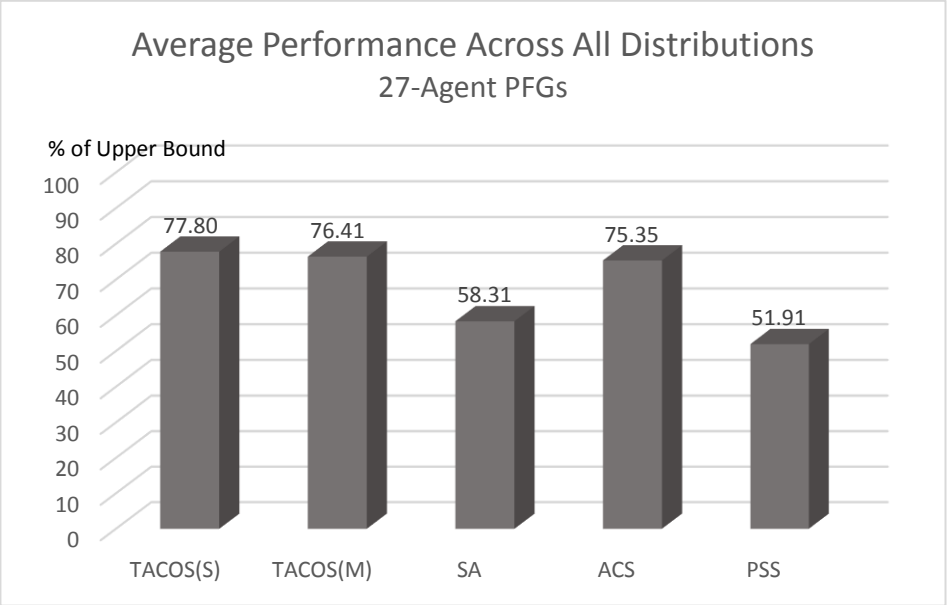

Figure 8.4 – Average performance (27-agent PFGs).

The best performing distribution was different for different methods (see Table 8.7). TACOS(S) and ACS performed best for the Beta Distribution. TACOS(M), SA and PSS performed best for the Normal distribution. All four methods performed worst for the Exponential distribution.

Table 8.7 – Best and worst Performance across distributions (27-agent PFGs).

Method	Best Performance	Value – % of Upper Bound	Worst Performance	Value – % of Upper Bound
TACOS(S)	Beta	98.79	Exponential	33.39
TACOS(M)	Normal	98.61	Exponential	31.62
SA	Normal	83.00	Exponential	16.72
ACS	Beta	94.28	Exponential	33.88
PSS	Normal	74.19	Exponential	14.06

Let us now examine the number of neighbours explored (see Table 8.8). TACOS(S) explored the maximum number of neighbours. This suggests why it performed the best. This is also true when comparing the performance of SA against PSS, SA explored more neighbours than

PSS which reflected in its performance being better than PSS. Similarly, ACS explored more neighbours compared to PSS and it too performed better than PSS.

In summary, as the size of search space grew, the relation between the number of neighbours explored and the average performance became more evident.

Table 8.8 – Average number of neighbours explored (27-agent PFGs).

27-Agent	Average neighbours explored
TACOS(S)	1345358
TACOS(M)	763696
SA	955952
ACS	677070
PSS	644478

8.3 Performance Comparison for Each Method

We shall now take a look at how the number of agents effects the performance for each method for both CFGs and PFGs.

8.3.1 CFGs: The Effect of Number of Agents on Performance

Figure 8.5 shows the effects of the number of agents on performance. Still looking at the overall average across distributions, we can see that when increasing the number of agents from 25 to 27, the overall average slightly decreases. Note that for 25 agents the comparison was made to the optimal while for 27 agents it was made to the upper bound. It would be ideal to be find the exact optimum for 27-agent games but this was not practical in terms of computation time and memory space.

Each one of the four methods deteriorated in performance with the increase in the number of agents. However, the degree to which performance dropped varied from method to method. The drop was least for ACS and most for SA.

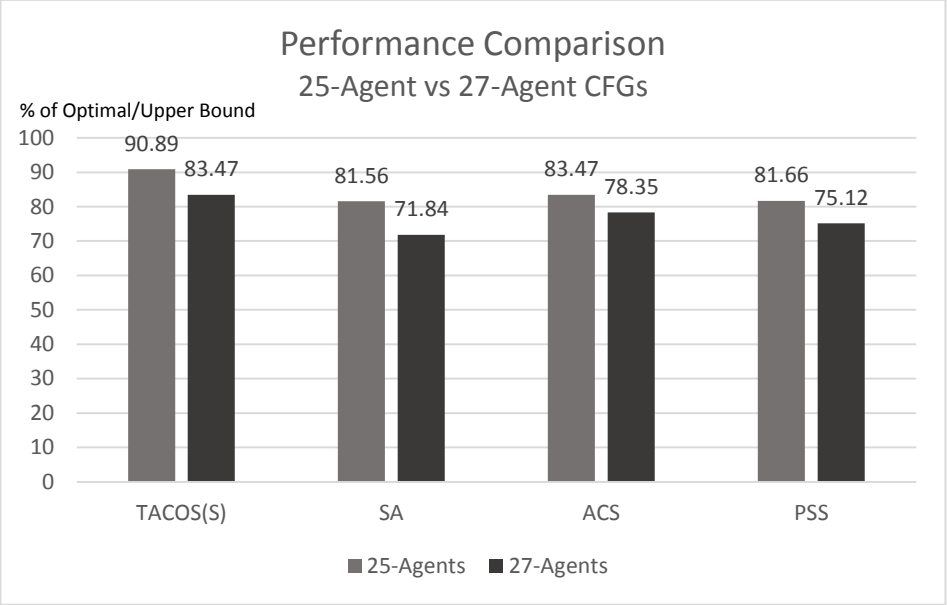

Figure 8.5 – Effects of the number of agents on performance (CFGs).

8.3.2 PFGs: The Effect of Number of Agents on Performance

For PFGs, the effects of the number of agents on performance is shown in Figure 8.6. From the results it can be seen that when the number of agent is small i.e. 10 agents, the average performance of all the heuristics across distributions is mostly encouraging with very small differences between them. Increasing the number of agents affected the performance of different methods to different degrees. For some methods, an increase in the number of agents resulted in a bigger drop in performance relative to others. Between the four heuristics, the drop was least for TACOS(S). This suggests that TACOS(S) is the most scalable between all the four methods.

Once again, it must be noted that performance is compared to the upper bound and not to the exact optimum.

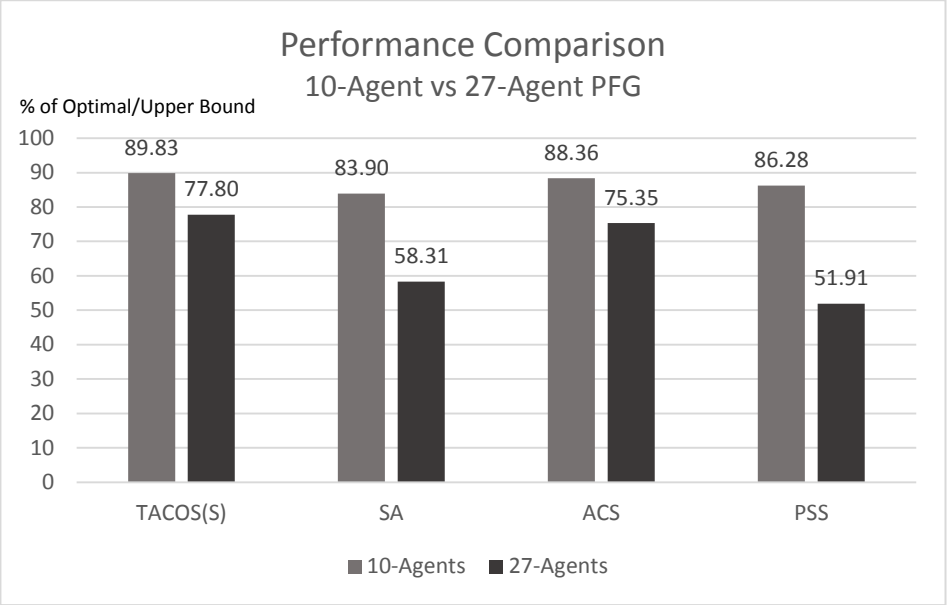

Figure 8.6 – Effects of the number of agents on performance (PFGs).

8.3.3 The Effect of Externalities on Performance

We shall now take a look at how externalities impact the performance of the four methods. This comparison is done in terms of the average performances across all six distributions for 27-agent CFGs and PFGs.

Figure 8.7 shows the difference in performances for each method. This depicts the effects of externalities on performance. The performances were measured against an upper bound. TACOS(S) and TACOS(M) performed best for both CFGs and PFGs for 27-agents. All the methods experienced a reduced performance when considering a PFG compared to CFG. TACOS(S), TACOS(M) and ACS showed a smaller reduction in performance while SA and PSS showed a sharper decline.

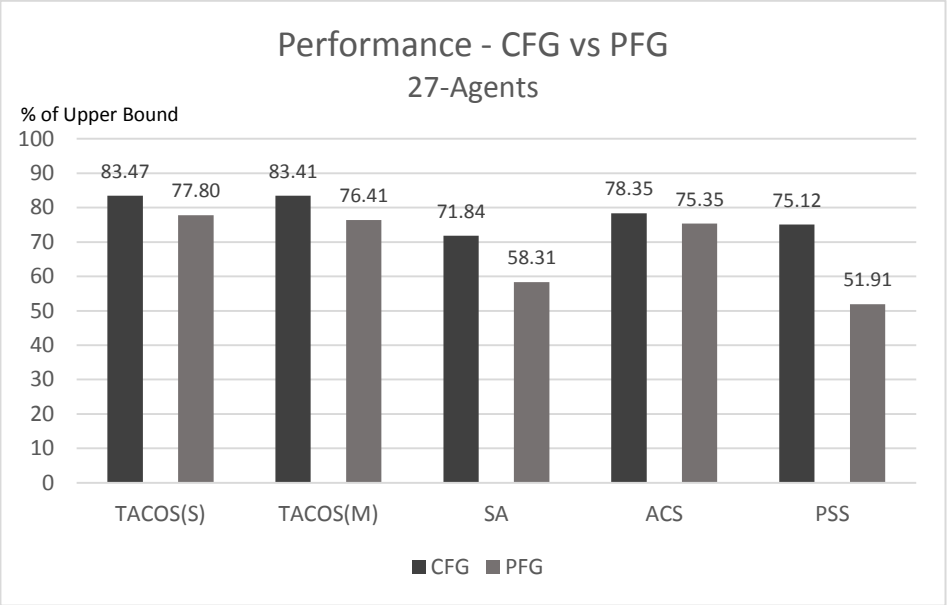

Figure 8.7 – The effects of externalities on performance (27-agent games)

8.4 Memory Usage

The results of simulation showed that TACOS is the best performing between all four heuristic methods. The main difference between TACOS and the other methods is that it uses long term memory while the other methods are memory-less. Thus, we will now examine the memory usage of TACOS for each of the four settings: 25 and 27 agent CFGs, and 10 and 27 agent PFGs. The size memory used for storing the tabu list depends on the number of iterations in TACOS. For TACOS(S) the number of iterations was 2500. In each iteration, at most two coalition structure can get added to the list. So the memory needed for the tabu list will be 2500 times the memory space needed to store two coalition structure. This is additional memory needed by TACOS but not by the other 3 heuristics.

8.5 Chapter Summary

This chapter presents the results of the simulations and discussed the performance of each method across all the distributions. For 25-agent and 27-agent CFGs, TACOS(S) is the best performing method. Even though SA explored on average the most number of neighbours, it

is the worst performing method across all distributions. When increasing the number of agents from 25 to 27, each method experienced a reduction in performance. ACS took the least performance hit while SA's performance degrade the most and was almost 10% lower. In terms of scalability, ACS is the most scalable as it showed the lowest gap in performance when moving to a bigger search space. However, TACOS(S) still managed to be best method by being able to return an overall average solution quality of more than 80% across all distributions.

For PFGs, it was still TACOS(S) that performed best on average over all distributions for both 10-agent and 27-agent games. The performance for all four methods was very similar for 10-agent games. However, increasing the number of agents to 27 saw SA and PSS taking the biggest impact with a performance drop of over 20%. For 27-agent games SA explored more neighbours and was slightly better than PSS. The average performance across all distributions was below 80% for 27-agent PFGs with TACOS(S) having the best average performance compared to the other methods.

When comparing 27-agent CFG and PFG games, we were able to see that externalities caused an impact on the performance of all the methods. TACOS(S), TACOS(M) and ACS were the least impacted when comparing CFGs against PFGs. The introduction of externalities have a greater impact on PSS which was the worst performing method. While it performed better than SA for a smaller number of agents, increasing the number of agents reduced its performance dramatically. The gap between the best and the worst solutions found is higher for PSS compared to SA. This means it is the least scalable method when it comes to PFGs which explains its poor performance.

The analysis provided an observation on the relationship between the overall performance and the scalability of the methods. While ACS is more scalable, its performance is not the best. While the gap is slightly bigger for TACOS, its performance is consistently the best.

Chapter 9 Conclusion and Future Work

This chapter presents conclusions and avenues for future work.

9.1 Conclusions

The problem of determining the optimal coalition structures is something that arises in many multiagent systems. This problem is computationally hard. Existing solutions to this problem are mostly design-to-time or anytime especially for PFGs. These solutions have exponential time complexity which means they are suitable only for small multiagent systems. For large systems, there is a need for methods that can quickly generate a solution that is not necessarily optimal but is close to the optimum. The aim of this research is to address this need.

More precisely, this research was aimed at exploring four different heuristic approaches, viz., tabu search, simulated annealing, ant colony search, and particle swarm search.

These four heuristic algorithms were devised so that they can be applied to different types of coalition games. In particular, there are two types of coalition games, viz., those in characteristic function form and those in partition function form. Two of these methods, viz., tabu search and simulated annealing are single agent approaches. The other two, viz., ant colony search and particle swarm search are population based.

The main contributions of this research are as follows.

Contributions

- A primary contribution of this research is devising the four heuristic algorithms for finding an optimal coalition structure. Although the heuristic methods are not entirely novel, they only provide a general framework that must be tailored to the problem at hand. Our contribution is in tailoring these methods for solving the CSG problem for CSGs and also for PFGs

- The second contribution of this thesis is the development of a compact representation of the values of coalitions. A compact representation is crucial in finding optimal coalition structures for PFGs in reasonable time. The representation we developed allows for the feasible representation of values of coalitions for PFGs. Without this representation, it would be impractical to run or test the heuristic methods.
- The third contribution was development of a method for calculating the upper bound on the value of an optimal coalition structure. The performance of the heuristics was evaluated relative to these bounds. Without this method for calculating the bounds, it would be impractical to evaluate performance since it is impractical to compute the exact optimum for large games.
- The fourth contribution of this thesis is the testing of our implementations of the four heuristic methods for **CFGs** using six different probability distributions. It must be noted that all the implemented heuristics are oblivious to the probability distributions from which the input is drawn. The performance of each heuristic method was analysed for each individual distribution. Specifically, performance was evaluated in terms of *running time*, the *quality of the solution*, and the *scalability* to larger games.
- The fifth contribution of this thesis is the testing of our implementations of the four heuristic methods for **PFGs** using six different probability distributions. Again, it must be noted that all the implemented heuristics are oblivious to the probability distributions from which the input is drawn. The performance of each heuristic method was analysed for each individual distribution. Again, performance was evaluated in terms of *running time*, the *quality of the solution*, and the *scalability* to larger games.
- The sixth contribution of this thesis is average performance evaluation. The average refers to average taken across all the six probability distributions. Again, the evaluation was in terms of *running time*, the *quality of the solution*, and the *scalability* to larger games. Further, the impact of externalities on heuristic performance was also analysed.

A summary of the main results:

- Empirical results suggest that the performance of each algorithm is influenced by the probability distribution from which the input is drawn, as well as on the size of the problem.

- All the heuristic methods performed reasonably well for four of the six distributions. These four distributions are Uniform, Normal, Beta and Triangular.
- For the other two distribution types, Gamma and Exponential, the performance was not so good with some methods performing much worse than others. We investigated the results and found that this is due to the gaps between the best and worst solution found. Thus it can be concluded that none of the heuristics was able to consistently find high value solutions in the problem instances with these input probabilities. We also come to the conclusion that the lower performance in these two distribution was significantly lower than other distributions and deserves further attention.
- When it comes to scalability, we found that each method takes a hit in overall average performance when moving from a smaller to larger problem games. The difference is that some heuristics take a larger hit than others. This varies across distributions but the pattern observed is that for PFG games, the decrease in performance is more obvious.
- We would like to note that, although we used (Rahwan & Jennings, 2007) integer partition graph approach to compute the bound, we did not use these bound to do any pruning of the search space i.e. removing non-promising integer partitions. Rather, the heuristics implemented were allowed to search the entire space of coalition structures. This was done to measure the real viability of heuristics as a general purpose method which theoretically can perform well even when the input is not known a priori, i.e. when it is impossible to compute bounds as the values of the coalitions are not known. Despite no pruning of the space and concentrating the methods to search only promising subspaces, the heuristics performed well in 4 out of the 6 probability distribution types. This stand to shows that the heuristics are quite promising method for finding an optimal coalition structure in CFGs and PFGs.

Summary of results for CFGs

In terms of overall performance, each method has its own advantages and disadvantages in performance. Table 9.1 shows a comparative summary of the performance for each method in the CFG as well as the advantages and disadvantages of each method.

- For CFGs, TACOS returns the best quality solution on average across all the distributions. This is despite it being the method that explores the least number of

neighbours. It is also consistent as evident by the lowest gap between the best and worst solution found. The drawback is that it needs more memory to store the tabu list. Also, it was found that parallelising the search with TACOS(M) lowers the performances slightly as the cost of initialising parallel process resulted in fewer iterations and consequently fewer neighbours being explored compared to TACOS(S).

Table 9.1 – Comparative summary of the performance of all methods (CFGs).

Method	Advantages	Disadvantages
TACOS	• Returns the best quality solution across all six distributions despite the lowest number of neighbours explored. • Lowest gap between best and worst solution	• Requires more memory to store tabu list. • Parallelisation lowers performance as TACOS (M) explores fewer neighbours than TACOS(S).
SA	• No additional memory analogous to tabu list	• Explores the most neighbours but performs worst due to redundant explorations. • Highest gap between best and worst solutions. • Least scalable
ACS	• No additional memory analogous to tabu list • Less memory than TACOS • Quality of solution almost as good as TACOS • Most Scalable	• Average overall performance lower than TACOS.
PSS	• No additional memory analogous to tabu list	• Parallelisation does not justify quality of solution

- For SA, it had the advantage of not needing additional memory as it does not store any information that is analogous to the tabu list. However, it would seem that although it explores the most neighbours, it is the worst performing method. This could be attributed to redundant exploration as it does not keep track of previously explored solutions. It was also shown to be the least scalable, taking the most performance hit when moving to a

bigger problem size. Moreover, it can be said that it is the most random with the highest gap between the best and worst solution.

- Like SA, ACS also does not maintain a large list like the tabu list used by TACOS, it too has the advantage of requiring less memory. One of its strengths is that, despite this, it returns an overall average quality of solution that is comparable to TACOS. It is also the most scalable, taking the least hit on performance as problem size increases. However, its only shortcoming is that overall average performance is slightly lower than TACOS.
- PSS is the average performer for CFGs, its advantage like SA and ACS is low memory usage. Its performance is comparable with SA overall. However, being a parallelised method, the performance does not justify the extra cost incurred to by parallelisation i.e. the extra time needed to initialise parallel threads.

Summary of results for PFGs

We shall now summarise the results for the performance of each method in the PFG setting. Table 9.2 provides a comparative summary of the performance for each method, listing their advantage and disadvantages.

- The result is consistent for TACOS in PFGs as well. Again it is the best performing method across all the distribution types. It exhibited the lowest gap between the best and worst solution and for PFGs it is the most scalable method with a smaller performance hit with the increase in problem size. The same disadvantages are inherited however, it does require more memory (to store the tabu list) and parallelisation has the same effect as it was for CFGs, i.e., lower performance.
- SA is the least scalable method for PFGs, having the biggest drop in performance when moving to larger games. It still suffers from redundant exploration as evident by the fact that the number of neighbours explored did not gave it an advantage as other methods that explored less gave better results. It still however, has the advantage of using less memory than TACOS.
- ACSs performance in PFGs is analogous to its performance in CFGs. It requires no additional memory as there is no tabu list. Also, it is the least impacted by externalities when considering the same problem size. The impact for ACS in settings where there are externalities (PFGs) and where there are no externalities (CFGs) is smaller compared to

the other methods. Its only disadvantage is that its overall average performance is slightly lower than that of TACOS.

Table 9.2 – Comparative summary of the performance of all methods (PFGs).

	Advantages	Disadvantages
TACOS	• Returns the best quality solution across all six distributions • Lowest gap between best and worst solution • Most Scalable	• Requires more memory to store tabu list • Parallelisation lowers performance as TACOS (M) explores fewer neighbours than TACOS(S).
SA	• No additional memory analogous to tabu list	• Explores the most neighbours but performs worst due to redundant explorations. • Redundant exploration • Least scalable
ACS	• No additional memory analogous to tabu list • Quality of solution almost as good as TACOS • Least impacted by externalities	• Average overall performance lower than TACOS
PSS	• No additional memory analogous to tabu list	• Parallelisation does not justify quality of solution • Highest gap between best and worst solutions. • Impacted the most by externalities • Least scalable

- Finally, we take a look at the performance of PSS in PFGs. Its only advantage is that it requires no additional memory for storing a tabu list unlike TACOS. However it is the worst performing method for PFGs. It has the highest gap between the best and worst solution found. It is the most impacted by externalities, losing a lot of performance when

considering PFGs with the same problem size. Lastly, it is also the least scalable in the PFG setting, taking the most performance hit as the problem becomes larger i.e. moving from 10 to 27-agents.

With the main contributions and summary of main results given, we shall now look at some future work that could be done to further the work done in this research.

9.2 Future Work

From the extensive experiments and the conclusions drawn from this research, we found that more can be done to further this research and continue to contribute to the field of knowledge. In the future we would like to extend this research as follows:

- Investigate the possibility of implementing short term memory for tabu search – This can be done by introducing a tabu tenure where the size of the tabu list is limited. This will help to reduce the memory usage of TACOS. The effect of having a smaller tabu list has to be investigated to determine its effect compared to the current implementation where the size of the increases with the number of iterations.
- Improve performance for Gamma and Exponential – Although overall the heuristics performed well in the majority of the distributions considered, they did not perform quite as good for the Gamma and Exponential distributions. This deserves further investigation with the possibility of creating versions of the algorithms with neighbourhood operators that are tailored to such distributions.
- Building combined heuristic algorithms – Looking at best and the worst solutions found for Gamma and Exponential, the gap for these two distributions suggests that the value of the input coalitions sits between two extremes that is so random that no heuristic can perform well. Thus as future work, it might be interesting devise a method which combines two or more heuristics such as combining greedy with tabu search or ant colony with tabu search which may yield better results. As greedy works by constructing solutions based on high value components, it may help to start the algorithm with a greedy solution rather than the random solution used currently as the starting point. Combination of ant colony with tabu search allows the algorithm to remove forbidding moves i.e. coalition structures in the tabu list and keeps track of promising neighbourhood operators. Searching the space with historically good

operators while avoiding already explored solutions should increase the algorithms performance.

- Improvements on neighbourhood operators – Although the neighbourhood operators performed rather well in most case, their performance for Gamma and Exponential is still left to be desired. For future work, improved operators with better exploration guarantees should be devised. These operators must be of better quality to be able to boost the performance of the heuristics. More analysis will have to be conducted on operator performance alone in order to achieve this.
- Conduct more thorough statistical testing and analysis to further investigate the difference in performance from one distribution to the other. Advance, more detailed statistical analysis techniques such as ANOVA to be considered.
- Devise ways for larger games – Another limitation that we would like to address is the limiting size of the problem. Due the limit imposed by storing the value of all coalitions which only allow us to run simulations of up to 27-agents, perhaps a way can be devised to restrict the problem instance that allow generating input for integer partitions where the optimal coalition may be. This way instead of generating the value of all the coalitions, we just generate the value of the coalitions in the promising partitions as contemplated by (Changder, Dutta, & Ghose, 2016b). We then have to modify the algorithm and their neighbourhood operators to generate only neighbourhoods that are of the size of the promising integer partitions. This to deserve further research and if successful will allow the CSG problem to be scaled to a larger number of agents.

References

Aziz, H., & De Keijzer, B. (2011). Complexity of coalition structure generation. *Proceedings of the 10th International Conference on Autonomous Agents and Multiagent Systems – Volume 1*, pp. 191-198.

Banerjee, B., & Kraemer, L. (2010). Coalition structure generation in multi-agent systems with mixed externalities. *Proceedings of the 9th International Conference on Autonomous Agents and Multiagent Systems - Volume 1*, pp. 175-182.

Bellman, R. (1952). On the theory of dynamic programming. *Proceedings of the National Academy of Sciences, United States of America, 38*(8), pp. 716-719.

Blumenfeld, D. (2009). *Operations research calculations handbook*, CRC Press.

Catilina, E. P., & Feinberg, R. M. (2006). Market power and incentives to form research consortia. *Review of Industrial Organization, 28*(2), pp. 129-144.

Chalkiadakis, G., Elkind, E., & Wooldridge, M. J., (2012). *Computational aspects of cooperative game theory*. San Rafael, Calif: Morgan & Claypool.

Chang, C. S., Lu, L. R., & Wen, F. S. (1999). Power system network partitioning using tabu search. *Electric Power Systems Research, 49*(1), pp. 55-61.

Changder, N., Dutta, A., & Ghose, A. K. (2016a). Coalition structure formation using anytime dynamic programming. *Proceedings of the International Conference on Principles and Practice of Multi-Agent Systems (PRIMA2016),* Phuket, Thailand.

Changder, N., Dutta, A., & Ghose, A. K. (2016b). A kernelization approach for anytime coalition structure generation using Knuth Algorithm X. *Proceedings of the International Conference on Principles and Practice of Multi-Agent Systems (PRIMA2016),* Phuket, Thailand.

Clerc, M. (2004). Discrete particle swarm optimization, illustrated by the traveling salesman problem. *New optimization techniques in engineering* (pp. 219-239) Springer.

Crainic, T. G., Perboli, G., & Tadei, R. (2009). TS2PACK: A two-level tabu search for the three-dimensional bin packing problem. *European Journal of Operational Research, 195*(3), pp. 744-760.

Dang, V. D., Dash, R. K., Rogers, A., & Jennings, N. R. (2006). Overlapping coalition formation for efficient data fusion in multi-sensor networks. *Proceedings of the 21st National Conference on Artificial Intelligence - Volume 1*, Boston, Massachusetts, pp. 635-640.

Dang, V. D., & Jennings, N. R. (2004). Generating coalition structures with finite bound from the optimal guarantees. *Proceedings of the Third International Joint Conference on Autonomous Agents and Multiagent Systems - Volume 2*, New York, New York, pp. 564-571.

Dasgupta, D., & Michalewicz, Z. (1997). *Evolutionary algorithms in engineering applications*. United States: Springer Science & Business Media.

Deb, K. (2001). *Multi-objective optimization using evolutionary algorithms*. United States of America: Wiley.

De Clippel, G., & Serrano, R. (2008). *Marginal contributions and externalities in the value. Econometrica, 76*(6), pp. 1413-1436.

Di Mauro, N., Basile, T. M., Ferilli, S., & Esposito, F. (2010). Coalition structure generation with GRASP. *Artificial intelligence: Methodology, systems, and applications* (pp. 111-120) Springer.

Di Mauro, N., Basile, T., Ferilli, S., & Esposito, F. (2014). GRASP and path-relinking for coalition structure generation. *Fundamenta Informaticae, 129*(3), pp. 251-277.

Dorigo, M., Maniezzo, V., & Colorni, A. (1996). Ant system: Optimization by a colony of cooperating agents. *IEEE Transactions on Systems, Man, and Cybernetics, Part B (Cybernetics), 26*(1), pp. 29-41.

Dos Santos, D. S., & Bazzan, A. L. C. (2012). Distributed clustering for group formation and task allocation in multiagent systems: A swarm intelligence approach. *Applied Soft Computing, 12*(8), pp. 2123-2131.

Dunne, P. E. (2005). Multiagent resource allocation in the presence of externalities. In Pechoucek, Michal, Petta, Paolo, Varga, Laszlo Zsolt (Eds.) (Ed.), *Multi-agent systems and applications IV: 4th international Central and Eastern European Conference on Multi-agent Systems, CEEMAS 2005, budapest, hungary, september 15-17, 2005, proceedings* (pp. 408-417). Berlin, Heidelberg: Springer.

Edwards, C., Burrows, P., Grover, R., Lowry, T., & Hall, K. (2005). Daggers drawn over DVDs. *Business Week*, pp. 32-35.

Epstein, D., & Bazzan, A. C. (2013). Distributed coalition structure generation with positive and negative externalities. *Lecture Notes in Computer Science, 8154*, pp. 408-419. doi:10.1007/978-3-642-40669-0_35

Feo, T. A., & Resende, M. G. C. (1995). Greedy randomized adaptive search procedures. *Journal of Global Optimization, 6*, pp. 109-133.

Fiechter, C. (1994). A parallel tabu search algorithm for large traveling salesman problems. *Discrete Applied Mathematics, 51*(3), pp. 243-267.

Finus, M. (2003). Environmental policy in an international perspective. In L. Marsiliani, M. Rauscher & C. Withagen (Eds.), (pp. 19-49). Dordrecht: Springer Netherlands.

Fogel, D. B. (1998). *Evolutionary computation: The Fossil Record*, Wiley-IEEE Press.

Foster, I., & Kesselman, C. (2003). *The Grid 2: Blueprint for a new computing infrastructure* Elsevier.

Galinier, P., Hertz, A., & Zufferey, N. (2008). An adaptive memory algorithm for the k-coloring problem. *Discrete Applied Mathematics, 156*(2), pp. 267-279.

Glinton, R., Scerri, P., & Sycara, K. (2008). Agent-based sensor coalition formation. *2008 11th International Conference on Information Fusion*, pp. 1-7.

Glover, F., & Laguna, M. (1997). *Tabu search*. Norwell, MA, USA: Kluwer Academic Publishers.

Goldberg, D. E. (1989). *Genetic algorithms in search, optimization, and machine learning* Addison-Wesley.

Graham, R. L., Knuth, D. E., & Patashnik, O. (1994). In Patashnik O. (Ed.), *Concrete Mathematics: A Foundation for Computer Science* (2nd Edition). Reading, Massachusetts, Addison-Wesley.

Guo, B., & Wang, D. (2006). Optimal coalition structure based on particle swarm optimization algorithm in multi-agent system. *The Sixth World Congress on Intelligent Control and Automation 2006 (WCICA2006) (Volume: 1)*, pp. 2494-2497.

Hussin, A., & Fatima, S. (2016). Heuristic methods for optimal coalition structure generation. *Multi-agent Systems and Agreement Technologies* (pp. 124-139) Springer.

Hyodo, M., Matsuo, T., & Ito, T. (2003). An optimal coalition formation among buyer agents based on a genetic algorithm. *Proceedings of the 16th International Conference on Developments in Applied Artificial Intelligence,* Loughborough, UK, pp. 759-767.

Jones, G. (2002). Genetic and evolutionary algorithms. *Encyclopaedia of computational chemistry,* John Wiley & Sons, Ltd.

Keinänen, H., & Keinänen, M. (2008). Simulated annealing for coalition formation. *Proceedings of the 18th European Conference on Artificial Intelligence,* pp. 857-858.

Kennedy, J., & Eberhart, R. (1995). Particle swarm optimization (PSO). *Proc. IEEE International Conference on Neural Networks, Perth, Australia,* pp. 1942-1948.

Kirkpatrick, S., Gelatt, C. D.,Jr, & Vecchi, M. P. (1983). Optimization by simulated annealing. *Science (New York, N.Y.), 220*(4598), pp. 671-680.

Klusch, M., & Shehory, O. (1996). A polynomial kernel-oriented coalition algorithm for rational information agents. *Proceeding of International Conference on Multi-Agent Systems ICMAS-96, AAAI,* Kyoto (Japan), pp 157-164.

Knuth, D. E. (1992). Two notes on notation. *American Mathematical Monthly,* pp. 403-422.

Lin, G., Zhu, W., & Ali, M. M. (2014). A tabu search-based memetic algorithm for hardware/software partitioning. *Mathematical Problems in Engineering, 2014*, p 15.

Liu, J., Shen, S., Yue, G., Han, R., & Li, H. (2015). A stochastic evolutionary coalition game model of secure and dependable virtual service in sensor-cloud. *Applied Soft Computing, 30*, pp. 123-135.

Lodi, A., Martello, S., & Vigo, D. (2004). TSpack: A unified tabu search code for multi-dimensional bin packing problems. *Annals of Operations Research, 131*(1-4), pp. 203-213. doi:10.1023/B:ANOR.0000039519.03572.08

Metropolis, N., Rosenbluth, A. W., Rosenbluth, M. N., Teller, A. H., & Teller, E. (1953). Equation of state calculations by fast computing machines. *The Journal of Chemical Physics, 21*(6), pp. 1087-1092.

Michalak, T. P., Dowell, A., McBurney, P., & Wooldridge, M. (2008). Optimal coalition structure generation in partition function games. *Proceedings of the 18th European Conference on Artificial Intelligence*, pp. 388-392.

Michalak, T., Sroka, J., Rahwan, T., Wooldridge, M., McBurney, P., & Jennings, N. R. (2010). A distributed algorithm for anytime coalition structure generation. *Proceedings of the 9th International Conference on Autonomous Agents and Multiagent Systems – Volume 1*, pp. 1007-1014.

Michalak, T., Rahwan, T., Elkind, E., Wooldridge, M., & Jennings, N. R. (2016). A hybrid exact algorithm for complete set partitioning. *Artificial Intelligence, 230*, pp. 14-50. doi:http://dx.doi.org/10.1016/j.artint.2015.09.006

Neumann, L. J., & Morgenstern, O. (1947). *Theory of games and economic behavior*, Princeton: Princeton University Press.

Nonobe, K., & Ibaraki, T. (2002). Formulation and tabu search algorithm for the resource constrained project scheduling problem. *Essays and Surveys in Metaheuristics*, pp. 557-588 Springer, doi:10.1007/978-1-4615-1507-4_25

Norman, T. J., Preece, A., Chalmers, S., Jennings, N. R., Luck, M., Dang, V. D., Gray, W. A. (2004). Agent-based formation of virtual organisations. *Knowledge-Based Systems, 17*(2), pp. 103-111.

Plasmans, J. E., Engwerda, J., Van Aarle, B., Di Bartolomeo, G., & Michalak, T. (2006). *Dynamic modeling of monetary and fiscal cooperation among nations*, Springer Science & Business Media.

Rahwan, T., & Jennings, N. R. (2007). An algorithm for distributing coalitional value calculations among cooperating agents. *Artificial Intelligence, 171*(8), pp. 535-567.

Rahwan, T., & Jennings, N. R. (2008a). An improved dynamic programming algorithm for coalition structure generation. *Proceedings of the 7th International Joint Conference on Autonomous Agents and Multiagent Systems – Volume 3*, pp. 1417-1420.

Rahwan, T., Michalak, T. P., & Jennings, N. R. (2012). A hybrid algorithm for coalition structure generation. *Proceedings of the 26th Conference on Artificial Intelligence (AAAI-12)*, Toronto, Canada, pp. 1443-1449.

Rahwan, T., Michalak, T. P., Jennings, N. R., Wooldridge, M., & McBurney, P. (2009). Coalition structure generation in multi-agent systems with positive and negative externalities. *Proceedings of the 21st International Joint Conference on Artificial Intelligence (IJCAI-09), (9),* pp. 257-263.

Rahwan, T., Ramchurn, S. D., Dang, V. D., Giovannucci, A., & Jennings, N. R. (2007). Anytime optimal coalition structure generation. *Proceedings of the 22nd Conference on Artificial Intelligence, (7*), pp. 1184-1190.

Rahwan, T., Ramchurn, S. D., Dang, V. D., & Jennings, N. R. (2007). Near-optimal anytime coalition structure generation. *Proceedings of the 20th International Joint Conference on Artificial Intelligence (IJCAI),* pp. 2365-2371.

Rahwan, T., Ramchurn, S. D., Jennings, N. R., & Giovannucci, A. (2009). An anytime algorithm for optimal coalition structure generation. *Journal of Artificial Intelligence Research, 34*(2), 521.

Rahwan, T., & Jennings, N. R. (2008b). Coalition structure generation: Dynamic programming meets anytime optimization. *Proceedings of the 23rd National Conference on Artificial Intelligence - Volume 1,* Chicago, Illinois, pp. 156-161.

Rahwan, T., Michalak, T. P., Elkind, E., Wooldridge, M., & Jennings, N. R. (2013). *An exact algorithm for coalition structure generation and complete set partitioning,* DCS.

Rahwan, T., Michalak, T. P., Wooldridge, M., & Jennings, N. R. (2015). Coalition structure generation: A survey. *Artificial Intelligence, (229*), pp. 139-174.

Rahwan, T., Michalak, T., Wooldridge, M., & Jennings, N. R. (2012). Anytime coalition structure generation in multi-agent systems with positive or negative externalities. *Artificial Intelligence (186),* pp. 95-122.

Rapoport, A. (1970). *N-person game theory: Concepts and applications* Courier Corporation.

Rolland, E., Pirkul, H., & Glover, F. (1996). Tabu search for graph partitioning. *Annals of Operations Research, 63*(2), pp. 209-232.

Sandholm, T., Larson, K., Andersson, M., Shehory, O., & Tohmé, F. (1998). Anytime coalition structure generation with worst case guarantees. *Proceedings of the National Conference on Artificial Intelligence,* Madison, Wisconsin USA, pp. 46-53.

Sandholm, T., Larson, K., Andersson, M., Shehory, O., & Tohmé, F. (1999). Coalition structure generation with worst case guarantees. *Artificial Intelligence, 111*(1), pp. 209-238.

Sandholm, T. W., & Lesser, V. R. (1997). Coalitions among computationally bounded agents. *Artificial Intelligence. 94*(1-2), pp. 99-137. doi:10.1016/S0004-3702(97)00030-1

Sen, S., & Dutta, P. S. (2000). Searching for optimal coalition structures. *Proceedings of the Fourth International Conference on Multiagent Systems (2000),* pp. 287-292.

Service, T. C., & Adams, J. A. (2010). Anytime dynamic programming for coalition structure generation. *Proceedings of the 9th International Conference on Autonomous Agents and Multiagent Systems,* pp. 1411-1412.

Service, T. C., & Adams, J. A. (2011). Randomized coalition structure generation. *Artificial Intelligence, 175*(16), pp. 2061-2074.

Shehory, O., & Kraus, S. (1998). Methods for Task allocation via Agent Coalition Formation *Artificial Intelligence, 101*, pp. 165-200

Starkweather, T., Mcdaniel, S., Whitley, D., Mathias, K., Whitley, D., & Dept, M. E. (1991). A comparison of genetic sequencing operators. *Proceedings of the Fourth International Conference on Genetic Algorithms,* pp. 69-76.

Sukstrienwong, A. (2011). Searching optimal buyer coalition structure by ant colony optimization. *International Journal of Mathematics and Computers in Simulation, 5*, pp. 352-360.

Thrall, R. M., & Lucas, W. F. (1963). N-person games in partition function form. *Naval Research Logistics Quarterly, 10*(1), pp. 281-298. doi:10.1002/nav.3800100126

Trelea, I. C. (2003). The particle swarm optimization algorithm: Convergence analysis and parameter selection. *Information Processing Letters, 85*(6), pp. 317-325.

Triola, M. F. (2006). *Elementary statistics,* Pearson/Addison-Wesley, Reading, MA.

Tsvetovat, M., & Sycara, K. (2000). Customer coalitions in the electronic marketplace. *Proceedings of the Fourth International Conference on Autonomous Agents (AGENTS '00)*, pp. 263-264. doi:10.1145/336595.337479

Wooldridge, M. J. 1. (2009). *An introduction to multiagent systems*. Chichester, U.K.: John Wiley & Sons.

Yeh, D. Y. (1986). A dynamic programming approach to the complete set partitioning problem. *BIT Numerical Mathematics 1986, Volume 26, Issue 4*, pp. 467-474

Yi, S. (2003). Endogenous formation of economic coalitions: A survey on the partition function approach. *The Endogenous Formation of Economic Coalitions, Carlo Carraro, Ed., Edward Elgar, London.*

Yong, G., Li, Y., Wei-Ming, Z., Ji-chang, S., & Chang-ying, W. (2003). Methods for resource allocation via agent coalition formation in grid computing systems. *Proceedings of the 2003 IEEE International Conference on Robotics, Intelligent Systems and Signal Processing, 1*, pp. 295-300.

Zlotkin, G., & Rosenschein, J. S. (1994). Coalition, cryptography, and stability: Mechanisms for coalition formation in task oriented domains. *In Proceedings of the Twelfth National Conference on Artificial Intelligence (AAAI94)*, pp. 432-437.

Appendix I – Generating the Input Data

Python comes with pseudo-random number generators for various distributions. This is included in the Python Library random.py (see https://docs.python.org/2/library/random.html for more information).

The following functions from the Python random module was used to generate the values for each subset:

1. random.uniform(*a*, *b*)
2. random.triangular(*low*, *high*, *mode*)
3. random.betavariate(*alpha*, *beta*)
4. random.expovariate(*lambd*)
5. random.gammavariate(*alpha*, *beta*)
6. random.normalvariate(*mu*, *sigma*)

The parameters chosen are given in Chapter 6.

Here are some example pseudocode for the data generation for the 25-agent CFG, 27-agent CFG and 10-agent PFG.

An example pseudocode for generating the coalition value for the uniform distribution:

```
for each C ∈ C^Ag
      cardinality = |C|
      coalitionvalue = random.uniform(0,cardinality)

write to inputfile.csv coalitionvalue
```

Another example for the beta distribution:

```
for each C ∈ C^Ag
      cardinality = |C|
      coalitionvalue = cardinality × random.betavariate(0.5,0.5)

write to inputfile.csv coalitionvalue
```

For the 27-agent PFG where the value of coalitions of size k are different from other coalition sizes, here are the example pseudocode for generating the coalition values.

For the uniform distribution:

```
Initialize n = number of agents
Initialize k = 2

for each C ∈ C^Ag
      cardinality = |C|
            if cardinality == k
            coalitionvalue = random.uniform(n² , 2(n²))
      else
            coalitionvalue = random.uniform(0, n)

write to inputfile.csv coalitionvalue
```

An example pseudocode for the beta distribution:

```
Initialize n = number of agents
Initialize k = 2

for each C ∈ C^Ag
      cardinality = |C|
            if cardinality == k
            coalitionvalue = n² × random.betavariate(0.5,0.5)
      else
            coalitionvalue = random.betavariate(0.5,0.5)

write to inputfile.csv coalitionvalue
```