

Simultaneous coalition formation and task assignment in a real-time strategy game

by **Fredrik Prántare**

Supervisor : Ingemar Ragnemalm
Examiner : Fredrik Heintz

Copyright

The publishers will keep this document online on the Internet – or its possible replacement – for a period of 25 years starting from the date of publication barring exceptional circumstances. The online availability of the document implies permanent permission for anyone to read, to download, or to print out single copies for his/hers own use and to use it unchanged for non-commercial research and educational purpose. Subsequent transfers of copyright cannot revoke this permission. All other uses of the document are conditional upon the consent of the copyright owner. The publisher has taken technical and administrative measures to assure authenticity, security and accessibility. According to intellectual property law the author has the right to be mentioned when his/her work is accessed as described above and to be protected against infringement. For additional information about the Linköping University Electronic Press and its procedures for publication and for assurance of document integrity, please refer to its www home page: <http://www.ep.liu.se/>.

Upphovsrätt

Detta dokument hålls tillgängligt på Internet – eller dess framtida ersättare – under 25 år från publiceringsdatum under förutsättning att inga extraordinära omständigheter uppstår. Tillgång till dokumentet innebär tillstånd för var och en att läsa, ladda ner, skriva ut enstaka kopior för enskilt bruk och att använda det oförändrat för ickekommersiell forskning och för undervisning. Överföring av upphovsrätten vid en senare tidpunkt kan inte upphäva detta tillstånd. All annan användning av dokumentet kräver upphovsmannens medgivande. För att garantera äktheten, säkerheten och tillgängligheten finns lösningar av teknisk och administrativ art. Upphovsmannens ideella rätt innefattar rätt att bli nämnd som upphovsman i den omfattning som god sed kräver vid användning av dokumentet på ovan beskrivna sätt samt skydd mot att dokumentet ändras eller presenteras i sådan form eller i sådant sammanhang som är kränkande för upphovsmannens litterära eller konstnärliga anseende eller egenart. För ytterligare information om Linköping University Electronic Press se förlagets hemsida <http://www.ep.liu.se/>.

Abstract

In this thesis we present an algorithm that is designed to improve the collaborative capabilities of agents that operate in real-time multi-agent systems. Furthermore, we study the *coalition formation* and *task assignment* problems in the context of real-time strategy games. More specifically, we design and present a novel anytime algorithm for multi-agent cooperation that efficiently solves the *simultaneous coalition formation and assignment problem*, in which disjoint coalitions are formed and assigned to independent tasks simultaneously. This problem, that we denote the problem of *collaboration formation*, is a combinatorial optimization problem that has many real-world applications, including assigning disjoint groups of workers to regions or tasks, and forming cross-functional teams aimed at solving specific problems.

The algorithm's performance is evaluated using randomized artificial problem sets of varying complexity and distribution, and also using *Europa Universalis 4* — a commercial strategy game in which agents need to cooperate in order to effectively achieve their goals. The agents in such games are expected to decide on actions in real-time, and it is a difficult task to coordinate them. Our algorithm, however, solves the coordination problem in a structured manner.

The results from the artificial problem sets demonstrates that our algorithm efficiently solves the problem of collaboration formation, and does so by automatically discarding suboptimal parts of the search space. For instance, in the easiest artificial problem sets with 12 agents and 8 tasks, our algorithm managed to find optimal solutions after only evaluating $\frac{2000}{68719476736} \approx 0.000003\%$ of the possible solutions. In the hardest of the problem sets with 12 agents and 8 tasks, our algorithm managed to find a 80% efficient solution after only evaluating $\frac{4000}{68719476736} \approx 0.000006\%$ of the possible solutions.

Sammanfattning

I denna uppsats presenteras en ny algoritm som är designad för att förbättra samarbetsförmågan hos agenter som verkar i realtidssystem. Vi studerar även *koalitionsbildnings- och uppgiftstilldelningsproblemen* inom realtidsstrategispel, och löser dessa problem optimalt genom att utveckla en effektiv anytime-algoritm som löser det *kombinerade koalitionsbildnings- och uppgiftstilldelningsproblemet*, inom vilket disjunkta koalitioner formas och tilldelas uppgifter. Detta problem, som vi kallar *samarbetsproblemet*, är en typ av optimeringsproblem som har många viktiga motsvarigheter i verkligheten, exempelvis för skapandet av arbetsgrupper som skall lösa specifika problem, eller för att ta fram optimala tvärfunktionella team med tilldelade uppgifter.

Den presenterade algoritmens prestanda utvärderas dels genom att använda simulerade problem av olika svårighetsgrad, men också genom att använda verkliga problembeskrivningar från det kommersiella strategispel *Europa Universalis 4*, vilket är ett spel som agenter måste samarbeta i för att effektivt uppnå deras mål. Att koordinera agenter i sådana spel är svårt, men vår algoritm åstadkommer detta metodiskt genom att systematiskt söka efter de optimala agentgrupperingarna för ett antal givna uppgifter.

Resultaten från de simulerade problemen visar att vår algoritm effektivt löser samarbetsproblemet genom att systematiskt sälla bort suboptimala delar av sökrymden. I dessa tester lyckas vår algoritm generera högkvalitativa anytime-lösningar. Till exempel, i de enklaste problemen med 12 agenter och 8 uppgifter lyckas vår algoritm hitta den optimala lösningen efter det att den endast utvärderat $\frac{2000}{68719476736} \approx 0.000003\%$ av de möjliga lösningarna. I de svåraste problemen med 12 agenter och 8 uppgifter lyckas vår algoritm hitta en lösning som är 80% från den optimala lösningen efter det att den endast utvärderat $\frac{4000}{68719476736} \approx 0.000006\%$ av de möjliga samarbetsstrukturerna.

Acknowledgments

In this section, I would like to acknowledge the people who have encouraged, influenced, helped, and supported me during the duration of this study — but also for the last few years that lead up to this thesis.

I would like to start by thanking and acknowledging my examiner **Fredrik Heintz** and my supervisor **Ingemar Ragnemalm** for their support, their positive attitude towards solving algorithmic problems, and for sharing their extensive knowledge in computer science and artificial intelligence with me. Thank you for continuously challenged me with interesting ideas and problems during the last few years. Your challenges have made me into a better problem solver, and has given me a self-confidence I didn't have before.

I would also like to thank all of the amazing people at the *Paradox Development Studio*. Special thanks goes to **Rickard Lagerbäck**, **Martin Hesselborn**, and my supervisor **Marko Korhonen**, whom all welcomed me to Stockholm with kindness and heart. The discussions we had were invaluable, and you always answered my questions — even in hectic situations and during out-of-office hours. *(it was an honor excommunicating your adversaries)*

Thanks to my exceptionally talented classmates **Max Halldén**, **Andreas Larsson**, and **David Lindqvist**, whom I had so much fun with during my studies. Special thanks goes to Max for saving me so many times, and helping me getting through the electronics and hardware classes. A thousand thanks goes to David for being an amazing project partner in so many courses, and to Andreas for his cheerful attitude and for being my thesis opponent. Thanks to all of you three for all of the fun games we've had.

On a personal level, I would like to thank my girlfriend **Elisabet Hägerlind** for her unequivocal support during our ten years together. Without her, I would never have managed to do anything worthwhile, and I deeply admire her stamina, work ethics, benevolence, ambitions, and loving kindness. Also, many thanks goes to her family for their tireless support. We had so much fun during the last ten years! **Ulrika and Sven-Erik**: thanks for welcoming me into your wonderful family, and for always being there for me. I'm eternally grateful that you helped shaping me into a better person with sound morals and ethics. **Simon, Johannes and Mats**: thank you for sharing so many fun games with me, and for all of the interesting discussions on various subjects we've had — you three are like brothers to me.

Special thanks goes to my brother **Oskar**, who has always been a great role model. Oskar introduced me to so many cool games when we were children (Europa Universalis included), and has influenced me in so many ways. He has always been the better of the two of us (sorry for pushing your buttons and being a crazy little brother at times), and has inspired me to become a better person.

Finally, I would like to thank my two amazing parents. **Elisabeth and Hans**: thank you for always encouraging me to challenge myself, to become a better person, to relax, and to have fun. Thanks for the discussions we've had, and for everything you have taught me. I'm sincerely grateful to have you as my parents, and could not have wished for better parents.

Contents

Acknowledgments	i
Contents	ii
List of figures	iii
List of tables	iv
List of algorithms	v
List of abbreviations	vi
1 Introduction	1
1.1 Background and problem context	2
1.2 Previous work in collaborative reasoning	9
1.3 Motivation	12
1.4 Aim	15
1.5 Research questions	15
1.6 Delimitations	15
2 Collaboration formation	17
2.1 Background theory and related work	17
2.2 Nomenclature	19
2.3 Problem definition	20
2.4 Algorithmic overview	21
2.5 Partitioning of the collaboration structure search space	22
2.6 Calculation of coalition values	24
2.7 Calculation of the initial upper and lower bounds of partitions	25
2.8 Deciding on an order for partition expansion	26
2.9 Searching for the optimal collaboration structure	28
2.10 An algorithm for collaboration formation	31
3 Evaluation	33
3.1 Performance benchmarking and quality evaluation	33
3.2 Implementation and equipment	36
3.3 Evaluation and benchmarking results	37
4 Discussion	43
4.1 Analysis of the experiments conducted in Europa Universalis 4	43
4.2 Analysis of the benchmarks on simulated problem instances	44
4.3 Pay-off distribution and collaboration structure stability	46
4.4 Resumability	46
5 Conclusion	47
5.1 Final words	47
5.2 Future work	48
Bibliography	50

List of figures

1.1	A photography of the famous Go match between AlphaGo and Lee Sedol	5
1.2	An image of a typical scenario in the strategy game StarCraft 2	6
1.3	An image of a typical player’s view in the game Europa Universalis 4	12
3.1	A scatter plot of the samples that were generated using EU4	37
3.2	A graph of the time it takes for CSGen to find the optimal solution in artificial problem instances of 6 tasks.	39
3.3	A graph of the time it takes for CSGen to find the optimal solution in artificial problem instances of 12 tasks.	39
3.4	A graph of the time it takes for CSGen to find the optimal solution in artificial problem instances of 18 tasks.	40
3.5	A graph of the time it takes for CSGen to find the optimal solution in artificial problem instances of 10 agents.	40
3.6	A graph that shows how far from finding the optimal solution the algorithm is when it is interrupted prior to finishing an exhaustive search (subpartitions).	41
3.7	A graph that shows how far from finding the optimal solution the algorithm is when it is interrupted prior to finishing an exhaustive search (collaboration structures).	41
3.8	A graph that show how the performance was affected by using the order of precedence from algorithm 2.	42
3.9	A graph that show how the performance was affected by using the order of precedence from algorithm 3.	42

List of tables

2.1	The three partitions P_3 , $P_{2,1}$, and $P_{1,1,1}$ for the possible coalition structures of the three agents $\{a_1, a_2, a_3\}$	23
2.2	All collaboration structures that can be mapped to the integer partition $\{2, 1\}$. . .	23
3.1	A table of the results from running the first benchmark (Thirty Years War).	38
3.2	A table of the results from running the second benchmark (Seven Years War). . .	38

List of algorithms

1	A generator that generates all integer partitions of the number n with m or fewer addends. It then inserts zeroes to all integer partitions until they all have m members.	24
2	A sorting comparator, based on counting zeroes, used to generate the order of precedence for partition expansions.	27
3	A sorting comparator, based on the lower and upper bounds of partitions, used to generate the order of precedence for partition expansions.	28
4	An algorithm that searches through a given subpartition.	30
5	An algorithm that refines a subpartition	31
6	A collaboration structure search algorithm based on branch-and-bound.	32
7	A collaboration formation algorithm that solves SCAP instances optimally.	32
8	A recursive brute-force algorithm for collaboration formation.	36

List of abbreviations

AI	<i>Artificial Intelligence</i>
ANN	<i>Artificial Neural Network</i>
API	<i>Application Programming Interface</i>
BFS	<i>Breadth-First Search</i>
BWAPI	<i>Brood War Application Programming Interface</i>
CFG	<i>Characteristic Function Game</i>
CPU	<i>Central Processing Unit</i>
CSGen	<i>Coalition Structure Generator (the presented algorithm)</i>
DAI	<i>Distributed Artificial Intelligence</i>
DNN	<i>Deep Neural Network</i>
EUMC	<i>Europa Universalis Monte Carlo</i>
EU4	<i>Europa Universalis 4</i>
FSM	<i>Finite State Machine</i>
GAP	<i>General Assignment Problem</i>
GPU	<i>Graphical Processing Unit</i>
IBM	<i>International Business Machines Corporation</i>
JPS	<i>Jump Point Search</i>
MRTA	<i>Multi-Robot Task Allocation</i>
NDCS	<i>Normally Distributed Coalition Structures</i>
NPD	<i>Normal Probability Distribution</i>
ORTS	<i>Open RTS (Real-Time Strategy)</i>
RTS	<i>Real-Time Strategy</i>
SAP	<i>Simultaneous Assignment Problem</i>
SCAP	<i>Simultaneous Coalition Formation and Assignment Problem</i>
TPU	<i>Tensor Processing Unit</i>
UPD	<i>Uniform Probability Distribution</i>

Chapter 1

Introduction

In this thesis we present an algorithm that is designed to improve the collaborative capabilities of agents that operate in real-time multi-agent systems. Furthermore, we study the *coalition formation* and *task assignment* problems in the context of real-time strategy (RTS) games. More specifically, we design and present a novel anytime algorithm for multi-agent cooperation that efficiently solves the *simultaneous coalition formation and assignment problem* (SCAP), in which disjoint coalitions are formed and assigned to independent tasks. This problem, that we denote the problem of *collaboration formation*, is an optimization problem that has many real-world applications, and similar task allocation problems have been studied thoroughly during the last few decades in many different settings [17, 71, 49, 18, 66].

The algorithm we present is based on *branch-and-bound*, which is a design paradigm for algorithms that has been effective in solving NP-hard¹ problems that are similar to the problem of collaboration formation (e.g. coalition formation). Our algorithm also utilizes other techniques that have been shown to be effective in similar problem instances, such as the partitioning scheme that was presented by Rahwan et al. to create disjoint subspaces of the coalition structure search space. [64]

The presented algorithm also takes advantage of a few new techniques, such as a new order of precedence for partitions that dictates which "branches" to search first. We also present a new concept to improve search performance by using refinements of the partitions to decrease search times.

The algorithm's performance is evaluated in a real-world application using *Europa Universalis 4* (EU4) — a commercial strategy game in which *adversarial agents*² may need to cooperate in order to effectively achieve their own goals, and in which agents are expected to decide on actions in real-time [39]. In this game, our algorithm is used to improve the coordination of armies. The quality of the solutions (to the problem of army coordination) that our algorithm generates are compared to the solutions generated by a Monte Carlo algorithm (that is currently being used in the publicly released version of the game).

The presented algorithm is also tested using simulated problem instances, in which the utility values of task assignments are randomized using common probability distributions. Such tests can be replicated by anyone with a computer, and are conducted in an endeavor to deduce whether the presented algorithm could be efficient in other instances than those that are similar to games, but also in an attempt to improve the replicability of this thesis.

This first introduction chapter mainly focuses on introducing the reader to the problem of solving strategy games using algorithms and computer-based techniques, such as adversarial

¹The collaboration formation problem is a generalization of the coalition formation problem, since any coalition formation problem can easily be translated to a collaboration formation problem. The contrary is, however, not possible. Coalition formation is an NP-hard problem for which no polynomial-time solution is known to exist [70]. Therefore, collaboration formation is at least NP-hard, and it's thus not surprising that the presented algorithm has a time complexity that is exponential in the number of agents.

²Adversarial agents are agents that do not necessarily share goals with other agents, but may in some cases benefit from collaboration, even if the agents they collaborate with have other ultimate goals or utility functions.

search and machine learning. This is accomplished using three famous examples of strategy games: *Chess*, *Go*, and *StarCraft*.

This chapter also describes and discusses previous examples of development and research in artificial intelligence (AI) relevant to RTS games, motivates research in collaboration and RTS AI, and explains why we decided to design a novel collaboration algorithm to solve the simultaneous coalition formation and assignment problem. We also explain why our algorithm could potentially improve the collaborative skills of agents in EU4, and ultimately the cooperative capabilities of agents in other real-time systems.

We formulate a problem statement in the later sections of this chapter using a selection of relevant research questions, an aim, and a motivation. The research questions are questions that the rest of the thesis will discuss and try to answer, and are mainly related to solving the simultaneous coalition formation and task assignment problem, since the presented algorithm is the main focus of this thesis.

1.1 Background and problem context

Throughout the history of humanity, humankind has been fascinated with the idea of intelligent machines. A portion of that fascination is in part due to the thought of artificial entities that may replace humans in tasks we deem burdensome, outperform humans in tasks where such entities may increase efficiency or improve safety, act as a new form of entertainment, or as a substitute to humans in situations that require social interaction.

Machines have for long been able to outperform humans in certain tasks that require excessive strength, high precision, or extensive repetition, but historically known to fail in solving problems that require adaptability, flexibility, and cooperation. However, during the last decade, new techniques and algorithms have made promising progress in solving highly dynamic problems that require the ability of learning (e.g. speech recognition and classification) and advanced adversarial reasoning (e.g. economics and games) [50, 73, 52].

Many computer-based strategy games are a particular instance of such problems, and inherently offer safe simulations of complex environments in which both human players and computer-based agents are required to cooperate and adapt to dynamic situations in order to succeed — without the "real-world dangers" of failure. Additionally, computer-based strategy games make it possible to compare different algorithms and intelligences (e.g. human to artificial) in a structured and cost-efficient manner; since infrastructure is already in place, and conventional risks are almost non-existent. The type of reasoning that is required by agents in computer-based strategy games is central to many other problems, and the scope to which game-based algorithms can be applied to seem endless, as argued by Buro [9]. Finally, almost all computer-based strategy games are complex multi-agent systems with severe real-time performance requirements, making them suitable for testing algorithms that are to be deployed in real-world situations and other real-time systems.

Previous work in strategy games

An important aspect of many computer-based strategy games is artificial intelligence (AI), which is used to simulate intelligent processes (e.g. planning and pathfinding) and human-like agents. Artificial agents can be used to challenge the player, or as an ally who can assist the player in dire situations. Other creative usages of AI in strategy games are also possible, such as in the strategy game *Crusader Kings II*³, where AI techniques are used to create *emergent narratives*⁴ by simulating simplified human desires and social behaviours. Furthermore, simulated intelligent processes are used to solve a plethora of problems in computer-based games and simulations. For example, the problem of finding the optimal path between two locations

³*Crusader Kings II* is a computer-based strategy game developed by *Paradox Development Studio*.

⁴An emergent narrative is a story which emerge from the actions of the players, and not by a pre-defined story structure.

is often solved using common search algorithms, such as *breadth-first search* (also known by its acronym "BFS") or the *A* search algorithm*.

Due to continuous research over several years, many algorithms used in computer-based games have gradually improved. Computer-based games has, as such, become what one might describe as a "natural breeding ground" for new technologies and a number of improvements to already existing algorithms. For instance, many pathfinding algorithms, such as *jump point search* (JPS) and A*-based search algorithms, have been improved in specific instances in order to increase the performance of computer games [32, 21]. Additionally, computer games have continuously pushed the boundaries of research in other research fields than AI, such as in computer graphics (e.g. real-time rendering engines), graphics hardware (e.g. graphics processing units), and real-time approximations of physical systems (e.g. real-time physics engines).

Several different AI algorithms and techniques have already been developed and adapted to solve a diverse number of different problems that are inherent to strategy games. For example, the minimax algorithm has been used to outstrategize human and computer-based opponents in chess by searching for the best possible strategy, and machine learning has been used to improve intelligent agents in situations where classical search techniques fail [35, 73]. Even if humankind has managed to solve many problems that are inherent to strategy games, there are still many important problems that remain unsolved, especially in adaptive reasoning and cooperation, as we shall see in the forthcoming chapters of this report.

It is worth mentioning that there are instances where the goal of an agent is not to play better than its opponents, but instead to evoke certain emotions or behaviours in human players. In such instances, an optimal strategy (in terms of winning) might not be desirable. Also, irrationality, i.e. not playing toward the goal of the agent, might sometimes be desirable due to making the agent seem "less robotic". Luckily, irrationality can easily be achieved by adding a bit of randomness, or by making certain types of actions preferable over others — even if they lead to undesirable outcomes.

Solving and understanding the problem of creating artificial agents that are capable of evoking specific emotions (or behaviours) might be an interesting problem, but this report's main focus is on developing and presenting an algorithm that enhances agent collaboration in real-time systems. As such, we hope to make it possible for researchers and programmers to make agents in real-time systems "act better" in terms of acting toward a certain quantifiable goal by utilizing collaboration. This report shall therefore not discuss irrational agents thoroughly, even if it is obvious that collaboration and cooperation algorithms could be used for such purposes as well.

The first advancements in the practical solving of non-trivial strategy games

Although discussions and research in AI for strategy games has existed for a long time, it was first in the early 90s that the practical development of highly specialized software and hardware allowed computers to excel in playing certain non-trivial strategy games. A computer that took advantage of such specialized software and hardware was *Deep Blue*, a chess computer developed by the *International Business Machines Corporation* (IBM) during the 80s and 90s. [37]

In 1997, Deep Blue became the world's first computer to win an official chess match against a (former) world champion, and managed to defeat the world famous chess player Garry Kasparov in a historic $3\frac{1}{2} - 2\frac{1}{2}$ chess game. Deep Blue managed to do so using a highly specialized computer architecture that was designed to take advantage of a customized alpha-beta search algorithm (i.e. a search algorithm based on minimax and alpha-beta pruning). Its hardware featured 480 custom chess chips and multiple levels of parallelism, and its search algorithm was combined with a rather complex state evaluation function, and a database of solved games and opening strategies. [53, 16, 35]

Today, modern chess engines (e.g. *Stockfish*, *Houdini* and *Komodo*) outperform human chess players without difficulty, and many of them manage to do so using techniques that share similarities to those used by Deep Blue, such as chess databases, advanced evaluation functions, and search algorithms for adversarial reasoning. Additionally, the ever-increasing efficiency and capabilities of new computer hardware has made it possible for personal computers and chess engines to consistently outperform chess experts. [25, 34, 4]

With Deep Blue, humankind solved the problem of creating a machine that can outperform and defeat the best human chess players. Solved problems are seldom of interest to problem solvers, and researchers have since the 90s moved on to more complex strategy games, such as *Go* and *StarCraft*.

Go (which literally means "encircling game") is a turn-based board game in which two players challenge each other in strategical and intuitive reasoning. The game originates from ancient China, and is often played on a game board of 19x19 positions using playing pieces called stones [3].

In spite of its simple rules and minimalist appearance, Go is far more complex than chess. Its state-space is huge due to a relatively large branching factor, and it has, on average, many more possible moves per turn [80]. A huge state-space with complex branching, and no apparent easy way to determine whether a certain state is on the path to victory, makes it hard to create successful artificial players using blunt search algorithms, such as the aforementioned algorithms that were successful in chess. Techniques that lack the disadvantages of "shrewd" brute-force algorithms are thus required, since adversarial search algorithms are generally too slow, or require too much memory.

It was only about a year ago that *AlphaGo*, an advanced artificial Go player developed by *Google DeepMind*, managed to become the first AI in the world to outperform one of the best human Go players in an official supervised match. This was accomplished in a match against Lee Sedol, a professional Go player from South Korea. Technically speaking, AlphaGo managed to do so using new hardware and state-of-the-art AI techniques that were very different from what Deep Blue used. Its hardware was based on more than 1000 central processing units (CPUs) and 100 graphical processing units (GPUs). It also took advantage of the *Tensor Processing Unit* (TPU), a new processing unit specialized in machine learning developed by *Google*. Its main algorithm was based on a variation of Monte Carlo tree search, guided by artificial neural networks (ANNs), and was combined with huge databases of moves, and an advanced evaluation function based on extensively trained deep neural networks (DNNs). The match ended 4 - 1 in AlphaGo's favor. Figure 1.1 shows a photograph of said match, where a human carries out moves as ordered by Deep Blue. [73, 23]



Figure 1.1: A photography of the famous *Go* match between *AlphaGo* (left), an artificial *Go* player, and *Lee Sedol* (right), a human 9 dan⁵ *Go* player.
Image courtesy of Google DeepMind.

An introduction to real-time strategy games and *StarCraft*

Even though *Go* is a complex game with a huge number of legal positions, the complexity of many popular computer-based strategy games overshadow the complexity of *Go*. Many of them — such as *Age of Empires* and *Hearts of Iron* — not only have an immense number of legal positions, but are also stochastic, played in real-time with severe time constraints, partially observable (i.e. they are imperfect information games), dynamic (in the sense that the opponent can make moves at the same time as you can), and mostly continuous. The complexity of such games make many of the previously mentioned approaches (e.g. minimax search and databases with solved games) impractical and inefficient, making it very difficult to create intelligent agents that are on similar skill-levels as human experts, even when utilizing highly specialized hardware and software.

One of the most popular computer-based strategy games is the military science fiction game *StarCraft 2*, in which the player has to gather resources, build bases, produce units, and wage war in order to succeed. The goal of the game is to defeat all opponents by destroying all their buildings, but games almost always end in the voluntary resignation of the losing players.

Many strategy games that are played in real-time (*StarCraft 2* included) are often discussed and analyzed in terms of certain recurring concepts. They include, but are not limited to:

- ***Build order***: a pre-defined production progression (analogous to chess openings).
- ***Micro and macro***: the extensive and detailed management of units and production.

⁵Professional *Go* rankings are 1 through 9 dan, where a 9 dan player is considered the strongest.

- **Strategy:** a grand plan of action that often dictates how to react to different hypothetical build orders of the opponent.

Such concepts are examples of abstraction layers that human players use in order to analyze and play *StarCraft 2* more efficiently. A typical scenario when playing *StarCraft 2* is depicted in figure 1.2, where the red player is on the verge of defeat due to not having reacted appropriately to an aggressive strategy executed by the green player.



Figure 1.2: An image of a typical scenario in the strategy game *StarCraft 2*.

The *StarCraft* and *Europa Universalis*⁶ game series are part of an important sub-genre of strategy games denoted *real-time strategy* (RTS) games. The notion "real-time" means that an RTS game is expected to be able to be simulated at a high update rate (i.e. many updates per second), and that it is **not** turn-based — in other words, if a game is played in real-time, then it is a dynamic game in which all players can act and make decisions simultaneously at almost any given moment. In practice, RTS games are usually implemented using discrete time steps in a non-continuous update loop, in contrast to what the name might suggest. However, RTS games seldom perceptually appear to progress incrementally due to often using higher update rates than the human brain can perceive, thus tricking the human brain into making the game appear continuous. Most RTS games can thus be treated as discrete-time systems with high requirements on performance — the *StarCraft* and *Europa Universalis* game series included.

The fact that RTS games are highly complex and played in real-time makes them extremely hard to solve. According to Usunier et al., a game of *StarCraft: Brood War* (the predecessor of *StarCraft 2*) has at least $16384^{400} \approx 10^{1685}$ possible states when only considering the positions of 400 units on a single 128x128 grid (i.e. 16384 different grid positions), in contrast to *Go* which is estimated to have about 10^{170} possible states (when played on a 19x19 board), and

⁶Some people could perhaps argue that the games in the *Europa Universalis* game series are not part of the real-time strategy (RTS) games sub-genre, since they are games in which human players are allowed to execute commands while the games are paused. Even though it is true that human players are allowed to execute commands while the games are paused, artificial players are not allowed to do so, and are thus expected to be able to play the games at high update rates (without pauses). Therefore, we argue that the *Europa Universalis* games are RTS games — at least from the perspective of artificial intelligence — since they share all the important technicalities (e.g. dynamic, real-time with high update rates, not turn-based, multi-agent, strategical) with other RTS games (e.g. *StarCraft* and *Age of Empires*). We shall therefore classify *Europa Universalis* as an RTS game throughout this report.

to chess with its upper bound of about 10^{46} possible states [82, p. 1] [80, p. 29]. Much higher state-spaces are achieved for Brood War if we take other important factors into consideration, such as unit types, resources gathered, and technology advancements. Ontañón et al. conservatively estimated the branching factor of Brood War to $b \in [10^{50}, 10^{200}]$, with an estimated average depth $d \approx 36000$, which are extreme numbers in comparison to the branching factors and depths of chess and Go (chess with $b \approx 35$ and $d \approx 80$, and Go with $b \in [30, 300]$ and $d \in [150, 200]$) [57, pp. 2-3]. Unfathomably huge state-spaces and branching factors are to be expected for most RTS games, including StarCraft 2 and Europa Universalis 4. This is due to the fact that there are often many more variables in play in RTS games than there are in chess or Go.

Interestingly, and despite the complexity that computers struggle with, humans are able to play RTS games very well. Buro and Churchill hypothesize that this is due to our brains being able to create hierarchical state and action abstractions [10, pp. 106-107]. In any case, no known computer-based player has managed to outperform human experts in playing real-time strategy games such as StarCraft 2, Brood War, or Europa Universalis 4. This is perhaps due to flawed hierarchical state and action abstractions, and lack of collaborative reasoning.

Previous work in artificial intelligence for real-time strategy games

Some of the first mentionable RTS games were released in the early 90s, and include games such as *Dune II: The Building of a Dynasty* (1992), *Command & Conquer: Red Alert* (1996), *Age of Empires* (1997) and StarCraft (1998). Work and research on AI for RTS games have since then continuously pushed the boundaries of what artificial agents in RTS games can do.

In 2003, Buro identified six fundamental AI research areas in RTS games [9]:

- **Resource management.**
- **Decision making under uncertainty.**
- **Spatial and temporal reasoning.**
- **Collaboration.**
- **Opponent modeling and learning.**
- **Adversarial real-time planning.**

Most of the six areas have been subject to substantial effort since then. However, collaboration has been left mostly untouched [57, pp. 3-7]. There has also been few formal attempts at holistic approaches, i.e. methods that tries to solve all RTS-related problems using a single algorithm or technique, such as the *Darmok system* and *CAT* [55, 2]. This may not be surprising, since RTS games inherently consists of several subsystems that researchers and game developers often treat separately. For example, a single computer-based player in EU4 consists of many "sub-agents" (i.e. artificial sub-systems that handle reasoning about different parts of the game) that communicate with each other, each consisting of many thousands lines of code.

A majority of the formal research in AI for RTS games has been conducted in a predecessor of StarCraft 2, namely the game titled *StarCraft: Brood War* (1998), which basically has the same game mechanics as its successor. The *Brood War Application Programming Interface* (BWAPI), released in 2009, has been used to test numerous AI techniques and algorithms in Brood War [12]. BWAPI is a programming interface that makes it possible for programmers and researchers to create artificial agents that can play Brood War using actuators and sensor-like abstractions, such as orders and visibility lists. For example, Synnaeve and Bessier used BWAPI to test Bayesian modelling for build order prediction, and Cadena and Garrido used BWAPI to conduct research on fuzzy case-based reasoning for managing strategy [74, 15].

Prior to BWAPI, research and experiments in AI for RTS games was mainly conducted using open-source RTS engines, such as the *ORTS* (Open RTS) game engine developed at the University of Alberta, and the RTS game *Wargus*⁷ [8, 79]. Multiple algorithms and techniques have been developed and tested using ORTS. For example, Hagelbäck and Johansson explored multi-agent potential fields to control the navigation of tanks, and Chang et al. tested Monte Carlo algorithms for real-time planning [31, 19]. In *Wargus*, Ponsen et al. used evolutionary algorithms to automatically generate new tactics, Weber et al. presented a case-based reasoning system for selection of opening strategies and build orders, and Ontañón et al. explored real-time case-based planning using supervised learning [59, 84, 56].

Most commercial strategy games, such as the games in the *Company of Heroes* and *Total War* series, don't have open source *application programming interfaces* (APIs), such as BWAPI, for AI development — thus making them impractical to use for research in algorithms. However, in the last year, some of the biggest IT-companies have started to invest resources into creating new programming interfaces, state-of-the-art research infrastructure, and machine learning libraries for AI research in commercial RTS games. For example, DeepMind (a subsidiary of Google) has announced that they are collaborating with *Blizzard Entertainment*⁸ to "release StarCraft 2 as an AI research environment" [24]. In their announcement, they motivated research in StarCraft, and wrote:

"StarCraft is an interesting testing environment for current AI research because it provides a useful bridge to the messiness of the real-world. The skills required for an agent to progress through the environment and play StarCraft well could ultimately transfer to real-world tasks."

Additionally, a research team at *Facebook AI Research* has recently published a paper on using machine learning for handling unit micromanagement in StarCraft [82]. They also developed *TorchCraft*, a programming library that is focused on enabling deep learning research for RTS games [75].

Even though most RTS AI researchers today seem to focus on machine learning, game companies still extensively use hard-coded rule-based AI for their games. Rule-based approaches makes it possible for game programmers to give their agents fixed "personalities" (e.g. aggressiveness or conscientiousness), and predictable simple behaviours. However, such behaviours are often inflexible, which makes them unusable in many competitive instances due to being easily exploitable by adaptable agents, or directly countered by other hard-coded agents.

The most common hard-coded rule-based RTS agents are based on finite state machines (FSMs) [28]. Such rule-based agents are in theory very simple, but may in reality become highly complex due to very complicated rules for state transitions. Rule-based agents have been improved over several years in many different commercial environments, but are in most cases (e.g. computer-based strategy games) not even close to reaching human-like skill levels. In such games, collaboration in RTS games is often emergent (or "implicit") from the rule-based nature of the artificial players, and is almost never explicitly coordinated. [57, p. 5]

Finally, there has also been a considerable amount of informal development in RTS AI that lack open source information and published scientific data. For example, programmers have put a lot of work into developing and exploring different approaches to improving computer-based players in competitive instances. AI competitions, in which only computer programs are allowed to compete, have given an incentive for developers and researchers to continuously improve their algorithms and game-playing agents over the last decade. Such competitions include the "ORTS RTS Game AI Competition" (2006-2009), the "AIIDE StarCraft AI Competition" (since 2010), the "CIG StarCraft RTS AI Competition" (since 2011), and the "Student StarCraft AI Tournament (SSCAIT)" (since 2011) [6, 7, 77, 78]. Many successful agents that

⁷Wargus is a *WarCraft* clone, and shares many similarities to StarCraft 2 and Brood War.

⁸Blizzard Entertainment is the creator of the StarCraft game series.

have been used in such instances are often based on FSMs and rule-based techniques, but are generally assisted by artificial intelligent processes, such as pathfinding algorithms, and priority-based reasoning. [12, 20, 57, 10]

1.2 Previous work in collaborative reasoning

As mentioned previously, collaboration has been left mostly untouched in RTS games. However, there has been considerable work and research on the topic in other instances of real-time systems and research fields, and the formation of organizational groups has been used to enable cooperation in several different scenarios and multi-agent systems [29, 42, 41, 86, 43].

Coalition formation is a technique that has been used to enable cooperation among agents in multi-agent environments by creating coalitions⁹ (groups) of agents aimed at achieving a certain goal or performing a set of tasks. This is generally accomplished by evaluating the utility of different coalition structures (i.e. groups of disjoint coalitions), and choosing the coalition structure with the highest utility value. The formed coalitions may then be used to perform tasks that require several agents in order to be accomplished efficiently. The problem of coalition formation is NP-hard, and coalition formation algorithms often have to evaluate a huge number of possible coalitions in order to find the optimal coalition structure, since the number of possible coalition structures is exponential in the number of agents. [63, 70]

Algorithms that solve the coalition formation problem are generally based on approaches that use optimized search techniques and evaluate many different solutions (i.e. coalition structures) [71, 72, 44, 46, 60, 64, 62], or genetic algorithms that gradually improve already existing structures by using processes that are inspired by natural selection [89, 36, 51]. Shehory and Kraus describes several anytime algorithms that tackles the coalition formation problem in the more general case, which allows agents to partake in multiple groups with overlapping tasks [72]. Additionally, Klusch and Gerber explored solving the problem of coalition formation in dynamic, open and distributed environments, and Kraus et al. did research on coalition formation algorithms in stochastic environments where agents lack complete information [44, 46]. Some of the most recent state-of-the-art coalition formation algorithms are based on anytime branch-and-bound, or dynamic programming [64, 62].

By assigning tasks to coalitions (using a bijection), coalition formation can be used to solve a generalization of the many-to-one *assignment problem*¹⁰, where each task may be assigned to any number of agents instead of assigning a single agent to each task. This can either be done by "treating" each coalition as a task, or by assigning tasks to the coalitions in a second step after the coalitions have already been formed. Coalition formation combined with task assignment can thus be used to create structured collaboration in multi-agent systems by utilizing what is denoted as *multi-robot task allocation* (MRTA) [30]. Many variations on solutions to different problems in the domain of MRTA have already been suggested, and many such problems have been shown to be instances of other well-studied combinatorial optimization problems [30, p. 1]. Many problems in the domain of MRTA can often be formulated as *integer programming problems*, such as the *generalized assignment problem* (GAP). [58]

The task assignment problem (i.e. the problem of optimally assigning tasks to already existing coalitions or agents) can be solved optimally by using the *Hungarian algorithm* to map tasks to coalitions [47]. An improved version of the Hungarian algorithm has a time complexity of $O(n^3)$, which is a relatively low time complexity in comparison to the computational complexities of the algorithms that exist for optimal coalition formation (since they

⁹Agents within a coalition often cooperate in order to achieve the goals of the coalition, but do seldom coordinate or cooperate with agents in other coalitions. Coalitions are described as being goal-oriented, short-lived, and often designed to serve a specific purpose. In practice, however, it's possible for them to be both long-lived (e.g. permanent), and serve no specific purpose. Furthermore, coalitions are usually coordinated using flat organizational structures, but may in some cases assign specific agents as leaders, or assign leadership roles to certain agents. [64, p. 522].

¹⁰The assignment problem consists of finding a maximum weight matching in a weighted bipartite graph.

are all exponential-time algorithms). However, using the Hungarian algorithm for task assignment does not guarantee an optimal collaboration formation — in this case an optimal solution to the combined problem of coalition formation and task assignment — even though the task assignment in itself is optimal, since an optimal coalition structure (in terms of a utility function) might not be the coalition structure that is best suited for the tasks at hand. As such, using coalition formation and the Hungarian algorithm in two separate steps does not guarantee an optimal solution to the combined problem of coalition formation and task assignment. However, if we integrate task assignment into the formation of coalitions, we can use branch-and-bound to guarantee optimality, decrease code complexity (since we don't need two separate algorithms to solve a single problem), and give worst-case guarantees when an exhaustive search is interrupted prior to finishing. This gives rise to the generalized coalition formation and task assignment problem, that we denote the *simultaneous coalition formation and assignment problem* (SCAP).

Previous work on SCAP is, as far as we understand, non-existent. There has, however, been extensive work in related instances. Apart from the previously mentioned work in coalition formation and task assignment, the *simultaneous assignment problem* (SAP) has been studied by Yamada and Nasu [88]. SAP is similar to SCAP, but with the major difference that agents in SAP are treated as super-additive. In SAP, the value of a group that is assigned to a task is defined as the sum of the values of assigning each individual agent to that group. Therefore, the algorithms that solve SAP cannot — in the general case — solve problems where **optimal** disjoint coalitions are needed. For instance, if we want to coordinate doctors and nurses (with different specializations and properties) to efficiently help patients, a good coordination scheme (or formation of disjoint working groups) should be based on creating groups that have all qualities that are required by the tasks that the groups are intended to handle. For example, assigning multiple doctors (with the same specialization) to helping the same patient would not be efficient in the general case. Modelling group requirements that prevent such solutions is trivial in SCAP, and can be accomplished using explicitly defined requirements for each task. However, this cannot be trivially accomplished if the problem is modelled as a simultaneous assignment problem.

On communication in real-time strategy games

It is worth mentioning that, in the domain of RTS games, all agents can communicate with each other during any update call from the game engine. Many game engines are using multiple threads to improve performance (e.g. Europa Universalis 4), and agents may in such cases need to synchronize data before forming valid coalitions. Most RTS games — Europa Universalis 4 included — can take advantage of perfectly stable and centralized communication techniques, but may require synchronization and optimizations (e.g. performance or memory optimizations) in order to run at real-time frame rates. In other instances, agents may suffer from unstable communication, and communication may need to be dealt with by having agents use decentralized approaches in order to form groups and coalitions (e.g. auction-based or face-to-face communication). Such aspects might have to be taken into consideration when "translating" an algorithm from the domain of RTS games into the domain of the real world. In any case, agents in RTS games can use collaboration and coalition formation algorithms that supports centralized communication to form optimal coalitions for collaboration, and may then use the coalitions as a basis to assist each other in solving complex tasks that require multiple agents.

Simultaneous coalition formation and task assignment in Europa Universalis 4

RTS games like Europa Universalis 4 (EU4) consists of multiple agents that need to cooperate in order to maximize their expected utility. In the domain of RTS games, there has been none to

very few attempts at using algorithms for collaboration in order to improve agent cooperation. We therefore introduce collaboration formation to the game EU4 in order to explore, deduce, and discuss the value that such techniques hold in the domain of RTS games. However, most algorithms for cooperation are not specifically designed for RTS games. Existing algorithms may therefore need to be adjusted (or redesigned) in order to maximize their efficiency in the domain of RTS games.

EU4 is a computer-based real-time *grand strategy*¹¹ game that was developed by the Swedish game developer *Paradox Development Studio*, and was released to the public in 2013. In EU4, the player manages a nation from the Late Middle Ages through the early Modern Period (1444 - 1821 AD), and has to deal with numerous simulations of real-world systems and processes, including military, politics, religion, logistics, economy, industry, trade, and technology. The simulations are simplified, but almost all the previously mentioned areas affect the others, creating complex conflicts of interest and prioritization. For example, a diminished military may not be able to survive the attacks of a military super power, and a highly militaristic country may deteriorate in the long run due to becoming economically unstable and technologically obsolete. The player is thus forced to balance many different areas in order to create a flourishing country that is able to withstand the test of time. Also, a strategical (or tactical) military decision could turn the tide of battle, leading to huge implications for a country; both politically and economically. The AI in EU4 need to take all these complications into consideration in order to be successful, making it an interesting game for research in complex cases of AI programming and algorithms.

In EU4, there are multiple different areas that may require cooperation in order to increase the utility of a given player (i.e. a given nation). For instance, optimal army management may require that tasks are performed by several agents, and alliances may need to be formed in order to achieve a certain strategic goal. Coalition formation algorithms can be used in order to create alliances, by deciding which coalitions have the highest utility values. Coalition formation can also be used to create collaboration by creating groups of agents (e.g. armies or merchants), and assigning them to tasks that are of interest to multiple stakeholders (e.g. defend a strategical position, or steer trade toward a region). In our case, we will use a new algorithm that solves SCAP to assign groups of armies to handle (e.g. attack, defend, or suppress rebellion) regions of interest (e.g. strategically interesting regions, or regions with high development).

The armies in EU4 consists of three different unit types: cannons, infantry, and cavalry. However, there exists an extensive number of different units within each unit type, and each army may have numerous unique properties — including a general that may improve the combat skills and maneuverability of the army he or she is assigned to. Therefore, to optimally assign armies to regions of interest, each army has to be modelled as a unique entity. Furthermore, each army is always positioned in a province (node) on the world map. The world map consists of over 4000 provinces. Each province is also part of a larger region (i.e. a group of nodes), of which some may be of greater interest than others (for various reasons). In contrast to StarCraft, EU4 is not played on a grid, and each province may have any number of neighbours. As such, and from the perspective of graph theory, the adjacency of provinces can be modelled as a forest (i.e. a disjoint group of trees), and implemented using an adjacency matrix or adjacency list.

The problem of assigning armies to regions can obviously be modelled as a SCAP, since armies can be seen as agents, and "regions of interests" as tasks. In EU4, this problem is solved using a simple Monte Carlo algorithm that is guided by a utility function, in which several randomized assignments are created. The randomized assignments are then "polished" in an attempt to find local optima. This technique is really fast at finding solutions, since it almost involves no precomputations, but require many iterations before finding solutions with high

¹¹A grand strategy game is a game where focus is on the higher abstraction levels of strategy. In contrast to games such as StarCraft, the player may have to control an entire nation instead of controlling a smaller set of units and buildings (in StarCraft, the unit count is limited to 200 units per player).

utility values. In practice, it almost never finds a value that is close to the global optimum due to being too inefficient.

Finally, an image of a typical player’s view in EU4 can be seen in figure 1.3, in which the human player and a few allies (played by the AI) are waging war against their opponents (also played by the AI).



Figure 1.3: An image of a typical player’s view in the game EU4. In this image, the player is playing as the nation Sweden, and currently conducts a war for independence against Denmark.

1.3 Motivation

This section motivates why research in RTS AI is of interest, and why applying collaboration techniques to RTS games may have commercial, scientific and societal benefits. Some motivations have already been stated in the previous subsections, but are discussed further here. We also motivate research in algorithms that solve SCAP, and why SCAP is of great interest to other instances than RTS games.

It’s also worth mentioning that Buro and Furtak wrote a paper that is aimed at motivating AI research in the area of RTS games [11]. Their paper manages to describe many key points to why AI research in RTS games is of interest. Most of their motivations are also true for this thesis, and their paper can thus be used as a complement to this section.

This thesis has been conducted in cooperation with the Paradox Development Studio, and we could therefore use the RTS game Europa Universalis 4 for testing and evaluating the algorithm that this thesis present. This is beneficial, since the goal of this thesis is to improve the knowledge and understanding of multi-agent collaboration algorithms for real-time systems (e.g. EU4). We attempt to achieve this by discussing and exploring how collaboration formation can affect RTS games that are inherently designed for several agents that operate in environments that resemble the real world. EU4 is one of the most popular games of such kind, and we are certain that EU4 is a multi-agent system in which computer-based players can benefit from improvements to their collaboration and multi-agent coordination capabilities.

Having full access to the source code of a complex commercial strategy game is a rarity (as discussed earlier), and this thesis was a one-of-a-kind opportunity to use a game of EU4’s calibre to conduct research on state-of-the-art RTS AI algorithms. An alternative to EU4 would have been to either use Brood War (and BWAPI) or ORTS instead. However, none of these games resemble the real world to the same degree as EU4, and they are **not** specifically

designed for hundreds of **adversarial** agents. We therefore hypothesize that using Brood War or ORTS could lead to less valuable results in terms of possibilities of translating the results to the real world, or would have required more work in terms of implementing and incorporating the collaboration algorithms into the games in a meaningful way. As such, we argue that EU4 was the better choice, and perhaps the best domain for research on real-time collaboration algorithms that was available during the time when this thesis was written. Another approach would be to use a system that is designed for real-time AI research, such as the *RoboCup rescue simulator* [13]. However, and as previously mentioned, having full access to the source code of a complex commercial strategy game is a rarity, and we therefore decided to use EU4 to benchmark our algorithm.

The commercial and societal motivation

One of the motivations to why research in RTS AI is interesting is based on the commercial aspects of computer games, and the commercial aspects of computer-based strategy games in particular. Strategy games are a huge part of the commercial computer and video game industry, and amounted to 36.4% of all sold computer games in the United States in the year 2015, according to the *Entertainment Software Association* [26]. RTS games are a huge part of the computer-based strategy games market. Additionally, many RTS games have sold hundreds of thousands of retail copies, including EU4 which has sold more than 1 million units, and StarCraft 2 which has sold several million copies [40, 1]. RTS games are thus a considerable part of the game industry, and may have an economical impact on the society as a whole.

Artificial agents (e.g. artificial players) and the simulation of intelligent processes (e.g. pathfinding and collaboration) are undoubtedly vital aspects of RTS games, thus making innovations and improvements to AI algorithms able to potentially increase the commercial value of the games in which the improved algorithms are used. Innovations to RTS AI may then be adapted to other real-time domains, and can therefore increase the commercial and societal benefits of other types of products. For example, collaboration algorithms that have been developed and evaluated using RTS games could potentially be used to improve the management of disaster emergency teams by integrating AI-based collaborative decision support into their systems, or the efficiency of industrial robots by supporting a more adaptable and dynamic task assignment scheme.

Additionally, RTS games can possibly be used for educational purposes in a wide range of subjects. For example, EU4 depicts and describes many real historical events (e.g. the *French revolution*, and the *Wars of the Roses*), and the world it reenacts is based on real geography, geopolitics and climate. There are however many historical and geographical inaccuracies, often due to the AI making other choices than those that would've been historically correct, but also because EU4 is, by design, a very approximative non-deterministic (i.e. a game played in the exact same way may have different outcomes due to random variables) simulation of a few hundred years of our recorded history. However, we hypothesize that the game mechanics of EU4 ultimately gives an incentive to the player to learn about many different topics in order to improve their skill, since good decisions are rewarded, and consistently making good decisions requires developed capabilities in reasoning, knowledge of history, and intuition. RTS games, such as EU4, could therefore be used as tools for education, since they inherently teaches the player about the real world by giving the player clear incentives to learn about history, geography, and processes in economics and logistics. The previously mentioned subjects are just a few examples of the topics that RTS games often cover, and we hypothesize that games can be specifically designed in ways that would allow them to be used as educational tools in many other subjects as well.

In order to strengthen the educational properties of RTS games, artificial agents should be able to act in a way that supports the educational processes of the game in which they are used. In order to do so, artificial agents may need to be able to act cooperatively and manage

to collaborate. In some instances, it might be important that the agents use collaboration to appear more human-like (e.g. history, and psychology), while in others they may have to use cooperation to increase their success rate at completing a certain task (e.g. strategy games, adversarial tasks, and economics). As such, improving the collaborative skills of artificial agents may aid the development of RTS games as educational tools, and thus give educators a wider (and perhaps better) array of tools at their disposal.

Finally, RTS games have become a worldwide cultural phenomenon, and there are companies that regularly host huge RTS game tournaments, such as *Intel*¹² with the Intel Extreme Masters, and Blizzard Entertainment with the World Championship Series [38, 5]. Improving RTS AI and collaboration would make it possible for players to train themselves against artificial players, in a similar fashion to what many chess players have done against artificial chess players. Agents with collaborative skills could also assist the player, or act as social companions in multi-agent environments.

The scientific motivation

AI research in the area of RTS games is not only motivated by the importance of strategy games in terms of entertainment or commerce, but also in the sense that RTS games can be seen as simplified simulations of real-world situations, processes, and systems. For example, many RTS games simulate logistics, economics, military, and politics. Such simulations make it possible to use RTS games to benchmark and test AI algorithms in scenarios that resemble real-world situations, which makes RTS games interesting to use as safe environments for research in real-time algorithms and AI techniques. In our case, we use RTS games to benchmark and evaluate an algorithm that has many potential real-world fields of application. As such, RTS games can assist us in understanding the limits of our algorithm, and discovering any potential problems it may have, before we attempt to use it in a real-world scenario (e.g. to coordinate doctors and nurses in a hospital), in which the slightest oversight or error may have disastrous real-world consequences.

Most RTS games offer many AI research problems that require a wide variety of techniques to solve. Such techniques include algorithms based on task allocation, collaboration, pathfinding, spatial and temporal reasoning, learning, resource management, and decision-making under uncertainty. Creating advanced software where all of the aforementioned problems can be simulated simultaneously is both expensive and complicated, and many RTS games are inherently designed and programmed to make it possible to do so. Already existing RTS games can thus be used as relatively cheap simulation environments for many real-time AI research problems, since it is expensive to develop new advanced simulation software from scratch.

On a side note: if one of our goals is to build human-like artificial agents, then we need to find a way to measure the progress of our development in creating such agents. This leads us to coming up with a way to experimentally show that an intelligent agent can act (what humans perceive as) indistinguishable from human players. Interactive games could act as confined Turing tests, where AI programmers attempt to make artificial agents act like human players (in contrast to making the AI act optimally in terms of winning a game or successfully performing a job) based on the limitations enforced by the game. Since human players are able to cooperate, artificial agents would need to be able to do so too, at least in order to act indistinguishable from human players in situations where humans are expected to collaborate. Without collaborative skills, artificial agents would merely be non-human entities that are capable of mimicking certain traits (but not all). Studying multi-agent coalition formation and task allocation could ultimately improve our understanding of how to create agents that are capable of human-like collaborative reasoning.

¹²Intel (i.e Intel Corporation) is a technology company that manufactures and creates new central processing units, but also produces motherboard chip-sets, new software for deep learning, etc.

1.4 Aim

The ultimate goal of this thesis is to improve and increase the knowledge and understanding of algorithms that are designed to improve agent collaboration. More specifically, the aim of this thesis is to solve the simultaneous coalition formation in the context of real-time multi-agent systems.

Additionally, we aim to explore some of the possibilities to improving the utility and collaborative capabilities of artificial agents in real-time strategy games. We intend to accomplish this by evaluating algorithms for collaboration formation using Europa Universalis 4, which is a real-time strategy game in which such algorithms could potentially improve army coordination and cooperation.

1.5 Research questions

The aim of this thesis, and the current status of research in collaboration algorithms for real-time strategy games (as described in the previous subsections), leads us to the following research questions, which we study and discuss in the remaining chapters of this thesis:

1. Can multi-agent task allocation be integrated into the formation of coalitions?
2. Can a collaboration formation algorithm be applied to real-time strategy games in order to improve the utility and capabilities of artificial agents?
3. What are the limitations of algorithms that solve the simultaneous coalition formation and assignment problem in the domain of real-time systems?

We deem that this thesis will have made significant progress in understanding the simultaneous coalition formation and assignment problem in the context of real-time multi-agent systems if any of these questions can be answered during this study.

1.6 Delimitations

With the previous research questions in mind, it is worth discussing whether our study is delimited by any factors that may influence its results, or whether there are any apparent weaknesses that stem from our research questions.

As is made evident by the previous sections, this thesis will only look at a specific organizational structure of agents: the **coalition**. The reason is simple: to look at every type of organizational structure would require time and resources that are not available for a single thesis. We thus have to choose a single and specific organizational structure that we want to explore and study further.

The main reason to why we choose the coalition as our organizational structure of study is that coalition formation algorithms are part of one of the most simple and powerful classes of agent organizational algorithms, and that the coalition formation problem has already been extensively treated and explored in other research environments (e.g. theoretical multi-agent systems, information gathering, economics, and logistics) [81, 54, 22, 67, 45]. However, coalitions have never been explored and evaluated in the domain of strategy games — at least not in a formal and practical setting, i.e. in a thesis or research paper conducted with focus on a multi-agent system that is already being used extensively in the real world (e.g. Europa Universalis 4). We therefore deem that exploring the coalition formation problem in the context of RTS games could be one of the next logical steps for research in multi-agent collaboration in RTS games, since agents in such games often lack collaborative reasoning.

Other organizational structures and paradigms such as hierarchies, holarchies, federations, teams, congregations, societies and markets each have their own benefits and weaknesses [33], and could be subject for future research.

Additionally, we will further constrain our focus by only studying the simultaneous coalition formation and assignment problem. Even though our main focus is on real-time strategy games, we also try to generalize our discussions and tests as much as possible, so that our results can be translated to other instances. With that said, we do not intend to exhaustively explore all possible ways to apply collaboration formation algorithms to real-time strategy games, and instead look at the specific problem of using collaboration formation to assign armies to tasks. Finally, the same is true for when we will look at the limitations of collaboration formation in the domain of real-time systems, since we will only look at performance characteristics and the value of the created collaboration structures, and more specifically the computational limitations in:

- Tests conducted for army coordination in the game Europa Universalis 4.
- Randomized tests, where utility values are based on normal and uniform probability distributions.

The fact that some of our tests are delimited to using EU4 may influence the replicability of this thesis. This is because the source code of EU4 is not open to the public, and as such, it'll be hard — if not impossible — to perfectly replicate our tests conducted in EU4. Additionally, EU4 is continuously updated, and previous game versions may not be available in the future. As such, the replicability of this thesis may suffer. In an attempt to circumvent this problem, we have tried to generalize and discuss our usage of collaboration formation in a way that makes it possible to translate our tests to other multi-agent systems and environments, and in that way increase the replicability of this study by making it replicable in other instances (e.g. other games or real-time systems). Since we will also use tests with randomized utility values based on common probability distributions, we hope to improve the replicability, since such experiments only require a computer to replicate.

Chapter 2

Collaboration formation

The main focus of this chapter is to give the reader an extensive and in-depth presentation of our collaboration formation algorithm. We will, however, begin this chapter by discussing agents, multi-agent systems, coalitions, coalition formation, and task assignment, since these are fundamental concepts that will be used frequently throughout the rest of this thesis. It is important that we establish a common understanding of these concepts in order to make it possible for us to make a reasonable formalization for our other most essential building blocks (e.g. collaboration structures and collaboration formation). We will not define these fundamental concepts ourselves, and instead look to the literature and use common definitions. This will hopefully reduce the risk of any possible misconceptions.

2.1 Background theory and related work

This section presents definitions for the most fundamental concepts that are used in collaboration formation (e.g agents, multi-agent systems, coalition formation, and task assignment).

Agents and multi-agent systems

An *agent* is commonly described as an autonomous entity that can perceive, reason and act in the environment in which it exists. This means that agents can, without human intervention, control their own actions, reason about their environment, and perceive their surroundings, at least to some extent. Such agents are often described as either task-oriented or goal-driven, and may exist in environments in which they have to act independently, competitively, or even cooperatively in order to achieve their goals. [65, p. 34] [85, p. 4] [83, p. 1]

A *rational agent* is an intelligent agent that, given its own perception and knowledge of the system in which it exists, acts in a way that the agent expects to maximize its own performance measure (e.g. its utility) [65, pp. 36-38].

Multi-agent systems are systems in which multiple intelligent agents exist. The research field of *distributed artificial intelligence* (DAI) is a subfield of artificial intelligence that is concerned with treating multiple agents collectively in an attempt to improve processes such as problem solving, cooperative reasoning, planning, and learning for agents in multi-agent systems. Perhaps more importantly for this thesis, however, is the fact that agents in multi-agent systems often affect other entities, and can thus affect other agents. Rational agents in multi-agent systems may therefore have to reason about other agents in order to **effectively** navigate their own state space. Agents that co-exist in the same system may have adversarial goals (e.g. armies in Europa Universalis 4), and may try to affect the environment in ways that turn out to be detrimental to the goals of their adversaries. In order to counter such events, agents may have to counteract and predict the actions of other agents. One way to make it possible for an agent to improve its capabilities of countering the possible actions of other agents is to improve the agents' collaborative and cooperative skills, since goals of different agents may not necessarily be diametrical. Agreements can be established, and may

ultimately increase the performance measure of all the agents that partake in the agreements, and increase their chances at achieving certain goals. [83, p. 1] [85, p. 121] [65, pp. 36-38]

Solving the problem of cooperation and collaboration in multi-agent systems may require advanced adversarial reasoning, but solving the problem of creating efficient groups aimed at solving complex tasks is a step forward towards achieving this.

With the previous statements and discussions in mind, how do we represent rational agents and multi-agent systems in a way that makes it possible to effectively reason about them? Since this thesis is mainly concerned with coalitions that are assigned to tasks, we shall focus on a representation that allow us to reason about multi-agent systems in a way that helps us understand how such collaboration structures operate, form, and affect their surroundings. We therefore first introduce coalition structures as a concept, and use coalitions as our main building block for multi-agent representation. We can then use coalitions as a way to define the more complex concept of collaboration structures, which will be the basis for our algorithm that solves the problem of assigning agents to groups (i.e. coalitions), where each group is aimed at achieving a goal or completing a task.

Coalitions and tasks

A *coalition* is defined as a set of agents $C = \{a_1, a_2, a_3, \dots, a_n\}$, where n is the number of agents in the coalition. Additionally, we often denote the number of agents in a coalition C by its cardinality $|C|$, and a coalition with 0 agents (i.e. $|C| = 0$) by the empty coalition \emptyset .

Agents within a coalition often cooperate in order to achieve the goals of the coalition, but do seldom coordinate or cooperate with agents in other coalitions. Coalitions are described as being goal-oriented, short-lived, and often designed to serve a specific purpose. In practice, however, it's possible for them to be both long-lived (e.g. permanent), and serve no specific purpose [33, p. 6]. Furthermore, coalitions are usually coordinated using flat organizational structures, but may in some cases assign specific agents as leaders, or assign leadership roles to certain agents. [64, p. 522]

A *coalition structure* is a set of disjoint¹ coalitions $S = \{C_1, C_2, C_3, \dots, C_n\}$, where n is the number of coalitions in the coalition structure [68, p. 1]. From another perspective, a coalition structure is a partition of a set of agents into coalitions [69, p. 2]. The process of deciding on which coalition structure to form is denoted the process of *coalition formation*; i.e. the algorithmic procedure of assigning all agents to disjoint coalitions [63].

In the literature, the term **coalition formation** has been used to denote both the creation of task-oriented coalition structures, but also to denote assigning agents to utility-based coalitions [72, 63]. To counteract any ambiguities and reduce possible misinterpretations, we will distinguish between what we denote *collaboration structures* and coalition structures. The first of the two, i.e. collaboration structures, will be used to denote task-oriented coalition structures in which the coalitions have a specific order of precedence (that denotes which task a specific coalition should be assigned to). The second of the two, i.e. coalition structures, will be used to denote the wider definition of disjoint coalitions.

In the general case of coalition structures, coalitions do not necessarily need to be assigned to specific tasks. In our case, however, we use a new collaboration formation scheme that evaluates coalition structures and task assignments (to the coalitions) at the same time, thus making it possible to form coalition structures (where each coalition is assigned a task) efficiently, and thus solving the *simultaneous coalition formation and assignment problem* (SCAP) optimally.

¹Two coalitions (or sets) are said to be **disjoint** if they have no agents in common.

2.2 Nomenclature

To make it easier for us to discuss simultaneous (or combined) coalition formation and task assignment, we define the **collaboration structure** as a *strict total ordered* coalition structure $\Phi = \langle C_1, C_2, C_3, \dots, C_n \rangle^2$, where each coalition is assigned to a task. The reason we make this definition will become evident in later chapters, when we use strict total ordered coalition structures to simultaneously form coalitions and assign tasks using branch-and-bound. Additionally, we denote the process of deciding on which collaboration structure to form (given a set of agents and a set of tasks) the process of **collaboration formation**.

By a **strict total ordered** coalition structure, we mean that all the coalitions C in a collaboration structure Φ are *strictly comparable*, i.e. that:

$$\forall C_i, C_j \in \Phi : i \neq j \implies i < j \text{ or } i > j.$$

In our case, we'll be using the indices of the coalitions to keep track of the ordering. This might seem redundant in relation to the previous statement about comparability, but will in fact turn out to be important when dealing with task assignments during collaboration formation. If we have the coalition structure $\{C_1, C_2, C_3, \dots, C_n\}$ induced by the collaboration structure Φ , then the order of the coalitions in the collaboration structure is given by the list $I_\Phi = \langle 1, 2, 3, \dots, n \rangle$. In other words, coalitions in collaboration structures adhere to:

$$\forall i, j \in I_\Phi : i < j \iff C_i < C_j.$$

Given a set of tasks $T = \{t_1, t_2, t_3, \dots, t_m\}$, where m is the number of tasks in the list, we will denote the *task assignment* of the coalition $C \in S$ to the task $t \in T$ by the pair (C, t) . By our definition, collaboration structures have tasks assigned to all coalitions, and we will always³ use a one-to-one mapping (i.e. a bijection) of coalitions to tasks for all collaboration structures in this thesis. As such, the number of tasks m will always be the same as the number of coalitions n in any valid collaboration structure. Additionally, we will be using the same ordering for tasks as for coalitions, and the tasks will be assigned to the coalitions using their indices. Therefore, the task assignment (C_i, t_i) is true for all $i \in I_\Phi$.

The complete collaboration structure of n coalitions (of agents A) and task assignments (of tasks T) can now be given by its *assignment vector*⁴ $\Phi_{A,T}$, which we will use to denote valid collaboration structures:

$$\Phi \equiv \Phi_{A,T} \equiv \langle (C_1, t_1), (C_2, t_2), \dots, (C_n, t_n) \rangle.$$

Furthermore, we will use \mathbb{S} to denote the space of all possible coalition structures, and \mathbb{P} to denote the space of all possible collaboration structures. Additionally, for a given set of agents A and a set of tasks T , we will use $\mathbb{S}_A \subseteq \mathbb{S}$ to denote the subspace of all valid coalition structures, and $\mathbb{P}_{A,T} \subseteq \mathbb{P}$ to denote the subspace of all valid collaboration structures. Finally, we will use \mathbb{C} to denote the space of all possible coalitions, and $\mathbb{C}_A \subseteq \mathbb{C}$ to denote the subspace of all valid coalitions that are composed by agents from the set of agents A . For example, if $A = \{a_1, a_2\}$, then $\mathbb{C}_A = \{\{a_1, a_2\}, \{a_1\}, \{a_2\}, \emptyset\}$.

Rahwan et al. described a general approach to the process of coalition formation [64, pp. 522-523]. Their approach is — as many others — based on the assumption that the value of a coalition structure is the sum of the values of all of its coalitions. The value of any coalition

²We will use angle brackets, instead of parentheses or braces, to denote any tuples or lists.

³In cases where multiple coalitions should be assignable to the same tasks, the tasks can be duplicated to handle this (e.g. make C_1 and C_2 have the same utility functions). Also, in task assignments where there exists tasks that should have zero agents assigned to them, the coalitions that corresponds to those tasks can be denoted by the empty coalition \emptyset . By using these strategies, our algorithm will be able to handle cases where a bijection would limit the assignment of tasks or the creation of collaboration structures.

⁴Throughout this thesis, we will be using *vector* as a synonym to *list* (which we use as commonly defined in mathematics).

structure S with n coalitions and accompanying *coalition values* $\{v_1, v_2, v_3, \dots, v_n\}$, is thus given by a utility function $V : \mathcal{S} \mapsto \mathbb{R}$ that is defined as $V(S) = \sum_{i=1}^n v_i$.

The same is almost true for our approach to collaboration formation, and our evaluation of collaboration structures. The only difference is that we will base a coalition's value in a collaboration structure on the task it is assigned to, i.e. the coalition values are calculated using the task assignments of the collaboration structure. As such, the **value** of a collaboration structure Φ with n disjoint task assignments with accompanying coalition values $\{v_1, v_2, v_3, \dots, v_n\}$, is given by a utility function $V : \mathcal{P} \mapsto \mathbb{R}$ that is defined as $V(\Phi) = \sum_{i=1}^n v_i$. Additionally, we will denote the number $u \in \mathbb{R}$ given by $u = V(\Phi)$ as the *utility value* of the collaboration structure Φ .

2.3 Problem definition

The problem of collaboration formation, i.e. the problem that our presented algorithm solves, can now be stated as the problem of how to (efficiently) find the collaboration structure Φ in $\mathbb{P}_{A,T}$ that maximizes $V(\Phi)$, and can be formalized as follows:

Given a set of agents $A = \{a_1, a_2, a_3, \dots, a_n\}$ with n members, a list of tasks $T = \langle t_1, t_2, t_3, \dots, t_m \rangle$ with m members, and the performance measure $v(C, t)$ of assigning a coalition $C \in S$ to a task $t \in T$, how do we find the disjoint list of coalitions $S = \langle C_1, C_2, C_3, \dots, C_m \rangle$, where $C_i \in 2^A$, that maximizes the sum $\sum_{i=1}^m v(C_i, t_i)$?

The collaboration formation (and coalition formation) problem can also be formulated as a binary integer programming problem⁵. This is because if we are given $\mathbb{P}_{A,T}$, the task assignments (C, t) in a collaboration structure $\Phi \in \mathbb{P}_{A,T}$ can be represented using a 0 (not part of the collaboration structure) or 1 (part of the collaboration structure).

In any case, using an algorithm based on brute force to solve SCAP is obviously not a viable approach for a higher number of agents and tasks. For example, if we have 15 agents that we want to assign to 5 disjoint tasks, we would have $5^{15} = 30517578125 > 10^{10}$ possible ways to arrange the agents (i.e. 5^{15} possible collaboration formations), since the number of arrangements for a given instance of the collaboration formation problem is m^n , where m is the number of tasks that n agents should be assigned to. Evaluating all such collaboration formations would take a lot of time, even with today's most efficient hardware, and especially if the evaluation of task assignments is time-consuming. Continuously running such processes is obviously not viable in systems where agents are required to act and reason in real-time, and we therefore need an efficient strategy when searching for optimal collaboration structures.

Our approach to collaboration formation is, as previously mentioned, partially based on the *IP-algorithm* for coalition formation that was presented by Rahwan et al. in [64], and adheres to the same general approaches. Their algorithm does not assign tasks to the coalitions it forms, and thus solves a special case of the collaboration formation problem. In other words, the collaboration formation problem is a generalization of the coalition formation problem. In fact, our algorithm solves the coalition formation problem optimally if all task assignments are evaluated using the same utility function. In such case, our algorithm is very similar to the IP-algorithm, and manages to exhibit similar properties. On the other hand, if we have the same number of agents and tasks, then our algorithm solves the task assignment problem optimally. As such, our algorithm solves a generalization of both the task assignment and coalition formation problems. Finally, the algorithm can be used to solve other combinatorial optimization problems as well (e.g. the multiple knapsack problem), but its efficiency in terms of performance in solving such problems is debatable, and is perhaps best described by the idiom "using a sledgehammer to crack a nut".

⁵The *binary integer programming problem* is a special case of the more general integer programming problem in which the unknowns are either 0 or 1.

2.4 Algorithmic overview

In order to work well in real-world situations and real-time systems (e.g. RTS games), our algorithm needs to satisfy the following properties:

- **Anytime.**

In order to run efficiently in real-time environments, the algorithm needs to be able to return an acceptable solution (i.e. a valid collaboration structure) at any given time, since there might not be enough time to run an exhaustive search. The longer the algorithm runs, the better the returned result should be.

- **Able to prune the search space.**

If the algorithm is not able to remove suboptimal subspaces from the collaboration structure search space, search times would suffer greatly. Since the space of collaboration structures is huge — even for relatively small numbers of agents and tasks — the algorithm would fail in many instances without the ability to prune.

- **Optimal.**

Since utility functions are often created and approximated by programmers (i.e. not fine-tuned by machine learning or derived optimally), there is no guarantee that an optimal collaboration structure (in terms of having the highest utility value) is the best collaboration structure in practice. However, the utility values are often the best approximations that were practically achievable at the time when the utility functions were defined, and there are instances where the coalition values perfectly reflect the practical utility of the task assignments (e.g. some theoretic games, and in systems where the mechanics that affect the agents are completely understood and explored). In such instances, the algorithm should be able to guarantee that an exhaustive search always returns the optimal collaboration structure, at least if it is beneficial for the system in which the algorithm is used.

- **Resumable.**

In real-time systems, the world might not change much during two time-adjacent system updates. Instead of having to restart the algorithm in such situations, it would be beneficial if the algorithm could resume from a previous search state. Therefore, if the algorithm is interrupted, the algorithm should be able to resume search without having to restart the search procedure from the beginning. In any case, a previous world-state can in many scenarios be used to approximate the current world-state, which can potentially be used to great benefit by a resumable collaboration formation algorithm.

Our collaboration formation algorithm that manages to satisfy all of these properties can be described using the following steps:

1. **Partition the collaboration structure search space.**

In order to make it possible to prune subspaces that contain suboptimal solutions, we must first partition the collaboration structure search space into subspaces, so that every possible collaboration structure is included in only one subspace.

2. **Calculate the coalition values for all possible coalition to task assignments.**

In order to calculate upper and lower bounds for the partitions (which we will use to prune the search space), we first calculate all coalition values, and then use the coalition values to calculate upper and lower bounds for all partitions.

3. **Calculate the upper and lower bounds for all partitions.**

We cannot know whether a specific partition can be pruned if we don't have a way to deduce whether the best possible solution in the partition can be discarded. We therefore

calculate the upper and lower bounds for all partitions, and use the bounds to effectively remove suboptimal collaboration structure subspaces.

4. Decide on an order for partition expansion.

The order for which we search the partitions may affect the efficiency of the algorithm, since the partitions may contain solutions of different quality (and thus affect global bounds). As such, we must decide on an order of precedence for the expansion of partitions. The order should be beneficial, at least in the sense that it should increase the performance of the algorithm, and preferably increase the quality of the anytime-solutions that the algorithm generates.

5. Search for the optimal collaboration structure using branch-and-bound.

In order to find the optimal collaboration structure, we search for the best collaboration structure by using branch-and-bound and subpartition pruning. Branch-and-bound makes it possible to discard groups of suboptimal solutions, and keep track of the bounds for each partition. Subpartition pruning means that we can generate non-redundant refinements for each partition, which makes it possible for us to discard specific sets of solutions that are found in each partition. These strategies makes it possible for us to prevent evaluating collaboration structures that cannot possibly be optimal, and thus increase the performance of our algorithm.

In the four following sections, the aforementioned steps will be thoroughly described and discussed.

2.5 Partitioning of the collaboration structure search space

Before finding the optimal collaboration structure, we first use a partitioning algorithm to divide the space of possible solutions into subspaces. Partitioning is, by definition, the act of grouping the elements of a set (in our case, the space of all possible collaboration structures) into non-empty subsets, so that every element of the set is included in only one of the subsets. Therefore, partitioning does not give rise to any redundancy, since a partitioning of the collaboration structure search space would create partitions in which no collaboration structure exists in two separate partitions at the same time.

For example, the set of agents $A = \{a_1, a_2, a_3\}$ has five distinct possible coalition structures: $\{\{a_1\}, \{a_2\}, \{a_3\}\}$, $\{\{a_1, a_2\}, \{a_3\}\}$, $\{\{a_1\}, \{a_2, a_3\}\}$, $\{\{a_1, a_2, a_3\}\}$, and $\{\{a_1, a_3\}, \{a_2\}\}$. These coalition structures can be partitioned into different subsets that we denote the *partitions* of the possible coalition structures. These partitions can then be used for pruning (when searching for the optimal collaboration structure), since it is possible to calculate bounds for each partition (i.e. the worst and best possible solutions that can be found in a given partition). For systems with few agents, this step might seem excessive and unnecessary, but as the number of agents increases, the number of coalition structures grows exponentially. The partitioning of the coalition structure search space can thus be used to reduce the time required for collaboration formation, since we will be able to remove suboptimal subspaces — even if we haven't even attempted to search them.

In any case, our partitioning scheme is based on integer partitions, which is a rather simple way to partition the search space, and has been used to great effect in a previous algorithm for coalition formation [64]. An *integer partition* of n is defined as a multiset of the natural numbers of which the multiset's members add up to n . For example, for the integer $n = 3$, the distinct integer partitions of n are the multisets $\{3\}$, $\{2, 1\}$, and $\{1, 1, 1\}$. What is more important for our algorithm, however, is the fact that any coalition structure with n agents can be directly mapped to one of the possible distinct integer partitions of the integer n . For instance, $\{\{a_1, a_3\}, \{a_2\}\}$ can be mapped to $\{2, 1\}$, and $\{\{a_1, a_2, a_3\}\}$ can be mapped to $\{3\}$. We can use this mapping to partition the collaboration formation search space, since any

collaboration structure can be mapped to a specific coalition structure. See table 2.1 for an example of the partitioning of the possible coalition structures for three agents.

P_3 :	$\{\{a_1, a_2, a_3\}\}$
$P_{2,1}$:	$\{\{a_1, a_2\}, \{a_3\}\}, \{\{a_1, a_3\}, \{a_2\}\}, \{\{a_2, a_3\}, \{a_1\}\}$
$P_{1,1,1}$:	$\{\{a_1\}, \{a_2\}, \{a_3\}\}$

Table 2.1: The three partitions P_3 , $P_{2,1}$, and $P_{1,1,1}$ for the possible coalition structures of the three agents $\{a_1, a_2, a_3\}$.

Recall that a collaboration structure Φ is a list of pairs of coalitions and tasks, and that all collaboration structures in $\mathbb{P}_{A,T}$ have the same number of pairs as there are tasks. We can therefore achieve distinct integer partitions that map to collaboration structures by inserting zeroes to the integer partitions that are of smaller size than $|T|$. For example, if we have $|T| = 3$ and $|A| = 3$, the assignment vector $\Phi_1 = \langle (\{a_1, a_3\}, t_1), (\emptyset, t_2), (\{a_2\}, t_3) \rangle$ can be mapped to $\{2, 1, 0\}$, and the assignment vector $\Phi_2 = \langle (\emptyset, t_1), (\emptyset, t_2), (\{a_1, a_2, a_3\}, t_3) \rangle$ can be mapped to $\{3, 0, 0\}$. This mapping can also be made for all permutations of Φ_1 and Φ_2 , since the partitions are represented by unique multisets of numbers. As an example, all collaboration structures that can be mapped to the partition that is represented by the multiset $\{2, 1\} = \{1, 2\}$ are shown in table 2.2.

$\langle 2, 1 \rangle$:	$\Phi_1 = \langle (\{a_1, a_3\}, t_1), (\{a_2\}, t_2) \rangle, \Phi_2 = \langle (\{a_1, a_2\}, t_1), (\{a_3\}, t_2) \rangle$
	$\Phi_3 = \langle (\{a_2, a_3\}, t_1), (\{a_1\}, t_2) \rangle$
$\langle 1, 2 \rangle$:	$\Phi_4 = \langle (\{a_1\}, t_1), (\{a_2, a_3\}, t_2) \rangle, \Phi_5 = \langle (\{a_2\}, t_1), (\{a_1, a_3\}, t_2) \rangle$
	$\Phi_6 = \langle (\{a_3\}, t_1), (\{a_1, a_2\}, t_2) \rangle$

Table 2.2: All collaboration structures that can be mapped to the integer partition $\{2, 1\}$.

In a later subsection, we will denote a multiset permutation of a given multiset representation of a partition P as a subpartition (i.e. refinement) p of P , and we will show how we can discard subpartitions to further increase the performance of our algorithm. We will also denote a partition by its multiset representation (i.e. when we write *partition*, and when it does not invoke any ambiguities, we consider its *multiset representation*).

By mapping collaboration structures to integer partitions in the aforementioned manner, the number of partitions of $\mathbb{P}_{A,T}$ becomes the same as the number of distinct integer partitions of $n = |T|$, which grows exponentially with n . Even though there exists an exponential number of distinct integer partitions for any integer, this is not a problem, since this exponential number is insignificant compared to $|\mathbb{P}_{A,T}| = |T|^{|A|}$ [64, p. 533]. For instance, given $|A| = 15$ and $|T| = 5$, the number of possible collaboration structures is $5^{18} > 10^{12}$, and the number of distinct integer partitions (of the integer 18) is 141. Finally, we present *Algorithm 1*, which is an algorithm that can be used to recursively generate all possible integer partitions of size $m = |T|$ for an integer $n = |A|$ (with inserted zeroes as described above).

Algorithm 1 A generator that generates all integer partitions of the number n with m or fewer addends. It then inserts zeroes to all integer partitions until they all have m members.

```

1: function GENERATEALLINTEGERPARTITIONSOF(n, m)
2:    $C \leftarrow \emptyset$ 
3:    $P \leftarrow \emptyset$ 
4:   GENERATOR( $n, n, m, P, C$ )
5:   add zeroes to the members of  $P$  until all members in  $P$  have  $m$  members
6:   return  $P$ 
7: end function

8: procedure GENERATOR( $n, k, m, P, C$ )
9:    $s \leftarrow$  size of  $C$ 
10:  if  $s > n$  then
11:    return
12:  end if
13:  if  $n = 0$  then
14:    insert  $C$  to  $P$ 
15:  return
16:  end if
17:   $i \leftarrow \min(n, k)$ 
18:  while  $i \geq 1$  do
19:    insert  $i$  to the back of  $C$ 
20:    GENERATOR( $n - i, i, m, P, C$ )
21:    remove the back from  $C$ 
22:     $i \leftarrow i - 1$ 
23:  end while
24:  return
25: end procedure

```

2.6 Calculation of coalition values

In order to prune and remove subspaces (e.g. partitions) that do not contain the optimal solution, we need to calculate the upper bounds for all partitions. In our case, we do this using the $m2^n$ possible task assignments (where n is the number of agents, and m is the number of tasks). In order to do so, we first calculate the value for every possible coalition to task assignment. This might seem costly, but is very cheap compared to actually searching for the optimal collaboration structure, even in instances where the evaluation functions have relatively high computational complexities. The computational complexity of a brute force search is $O(m^n)$, since $m2^n \ll m^n$ for big n and m , while the computational complexity of calculating all possible coalition values is:

$$O\left(\max_{t \in T, C \in \mathbf{C}_A} f_t(C) m 2^n\right)$$

where $f_t(C)$ denotes the number of operations it takes to calculate the coalition value for the coalition $C \in \mathbf{C}_A$ that is assigned to the task $t \in T$. For example, if each task evaluation has a time complexity of $O(g)$, where g is the number of agents assigned to a task, then the time complexity would become $O(gm2^n) = O\left(\frac{nm2^n}{2}\right) = O(nm2^n)$, since there always exists a possible coalition with n members (which is often denoted the *grand coalition*). If $n = 15$ and $m = 10$, then $nm2^n = 4915200 < 10^7 < m^n = 10^{15}$. Additionally, as n grows, the difference grows by an exponential factor. This is obvious, since if $m > 8$ and $n > 3$, we have that:

$$\frac{m^n}{nm2^n} = \frac{m^{n-1}}{n2^n} > \frac{8^{n-1}}{n2^n} = \frac{(2^3)^{n-1}}{n2^n} = \frac{2^{3n-3-n}}{n} = \frac{2^{n+n-3}}{n} > \frac{2^{n+3-3}}{n} = \frac{2^n}{n}$$

And if m grows, and $n > 3$, the difference grows faster than by a factor of $m^2/16$, since:

$$\frac{m^n}{nm2^n} = \frac{1}{nm} \left(\frac{m}{2}\right)^n \geq \frac{1}{2^{n-1}} \left(\frac{m}{2}\right)^{n-1} = \left(\frac{m}{4}\right)^{n-1} > \left(\frac{m}{4}\right)^{3-1} = \frac{m^2}{16}$$

Similar relative growths can be shown if the evaluation functions have any degree k of polynomial complexity, since we would then end up with n^k in the denominator of the first of the two equations. From the second equation, we can also conclude that if both n and m grows, the difference grows by a factor that is bigger than $\left(\frac{m}{4}\right)^{n-1}$, which translates to rather staggering values. This is not surprising, since we expect the number of tasks and agents to have an exponential growth in the number of required evaluations during search, due to the increasing number of possible partition permutations and solutions. Finally, the above equations are based on the assumption that the utility values are calculated before searching.

All known optimal coalition formation algorithms have far worse non-polynomial time complexities⁶ than 2^n (e.g. [64, 62, 60]). This is due to the time it takes to search the coalition structure search space, which is a search space with many more search nodes than the number of possible coalitions. Since the collaboration formation problem (i.e. SCAP) is a generalization of the coalition formation problem, higher computational complexities than those found in algorithms used for coalition formation are to be expected for collaboration formation algorithms. As such, the exponential computational complexity of calculating all possible coalition values is unlikely a problem for our algorithm.

However, if memory is a problem (i.e. if we cannot possibly store the $m2^n$ values in memory), one can simply recalculate the coalition values every time they are needed. This can potentially make the algorithm much slower, especially if it is a complex matter to calculate the coalition values, since it is generally not known beforehand how many times the coalition values may have to be recalculated before we find the optimal coalition structure.

Another approach for memory-management would be to use memoization (caching) to only calculate the coalition values once (and only when necessary). However, this might not be as advantageous as it appears, since all coalition values are often needed in order to find the optimal coalition structure. In such cases, this approach would only add unnecessary overhead compared to calculating all coalition values beforehand.

As an example to the previous explanation of calculating coalitions values: if we have the set of agents $A = \{a_1, a_2\}$, the four possible coalitions are $C_1 = \{a_1, a_2\}$, $C_2 = \{a_1\}$, $C_3 = \{a_2\}$, and $C_4 = \emptyset$. If we also have a list of tasks $T = \langle t_1, t_2 \rangle$, then each of the four coalitions C_1 , C_2 , C_3 and C_4 can potentially be assigned to either task t_1 or task t_2 . Therefore, we calculate the value v_{ij} for each possible pair (C_i, t_j) of coalitions and tasks: $\{v_{11}, v_{12}, v_{21}, v_{22}, v_{31}, v_{32}, v_{41}, v_{42}\}$, which we then use to calculate the bounds of partitions.

2.7 Calculation of the initial upper and lower bounds of partitions

While calculating all possible coalition values, we also calculate the values that we denote *cardinal values*. These represent the minimum, average and maximum values for a given size of coalitions. The cardinal values can then be used to calculate the lower and upper bounds of partitions, and give an indication to the potential quality of a given solution (i.e. worst-case guarantees).

⁶Even though there exists no coalition formation algorithm that runs in polynomial time (in the number of agents), there exists coalition formation algorithms that are polynomial in the size of the input, where the input is defined as the number of coalitions (e.g. [64]).

Let $Q_{min(n)}$, $Q_{avg(n)}$, and $Q_{max(n)}$ denote the minimum, average, and maximum cardinal values of size n of coalitions, respectively. We now define:

$$\begin{aligned} Q_{min(n)} &= \min_{\{(i,j) \in J(n)\}} (v_{ij}) \\ Q_{max(n)} &= \max_{\{(i,j) \in J(n)\}} (v_{ij}) \\ Q_{avg(n)} &= \sum_{\{(i,j) \in J(n)\}} (v_{ij}) \end{aligned}$$

where $J(n) = \{(i, j) | i \in I_{\Phi} \wedge j \in I_{\Phi} \wedge n = |C_i|\}$; i.e. all pairs of indices for all coalition to task assignments, in which the size of the coalition in the task assignment is exactly n .

Now, recall that a partition is a multiset of integers $P = \{i_1, i_2, \dots, i_k, \dots, i_n\}$, where i_k is the k :th addend in one of the possible distinct integer partitions P of $|A|$. We now denote the lower and upper bounds of P as L_P and U_P , respectively. A loose lower bound of P can now be calculated as:

$$L_P = \sum_{i \in P} Q_{min(i)}$$

However, we can calculate a better (i.e. higher) lower bound of P by using $Q_{avg(n)}$, as shown by Rahwan et al. in [64, p. 559]:

$$L_P = \sum_{i \in P} Q_{avg(i)}$$

Finally, an upper bound of the partition P can be calculated as:

$$U_P = \sum_{i \in P} Q_{max(i)}$$

Using these bounds, we can also calculate initial lower and upper global bounds for the optimal collaboration structure that can be found in \mathbb{P}_{AT} . First, we denote L_G and U_G as the lower and upper global bounds, respectively. Initial lower and upper global bounds can now be calculated as $L_G = \max_{P \in \mathbb{Q}} (L_P)$ and $U_G = \max_{P \in \mathbb{Q}} (U_P)$, where $\mathbb{Q} = \{P_1, P_2, \dots\}$ denotes the set of all partitions.

In order to calculate even tighter bounds during the search for the refinements of partitions, we calculate the values that we denote as the *cardinal task values*. The cardinal task values are similar to the cardinal values above, but instead denote the minimum, average, and maximum values of every task assigned to a coalition of a given size. As such, we have a set of cardinal task values for every task $t \in T$ that can be used to calculate tighter bounds for subpartitions. The cardinal task values (and the bounds for every subpartition) are calculated in almost the exact same manner as the cardinal values (and partition bounds) above. However, there is one major difference: the cardinal task values are calculated per task, and we use the cardinal task values to calculate the bounds of the subpartitions.

2.8 Deciding on an order for partition expansion

In order to increase performance, and decrease the number of partitions that we actually expand (i.e. search), we can decide on an order for the expansion of partitions, instead of just expanding blindly. The question we ask ourselves is, on what criteria should we sort the partition expansion order?

There are many criteria that we can use, including lower bounds, upper bounds, and the number of a certain integer in a given partition. Other criteria can also be used, and specific problem descriptions may include information that can be used to improve the expansion order (e.g. the characteristics of agents and tasks).

In [64], Rahwan et al. first searches the partitions that they denote $P_n = \{n\}$ (which only represents the grand coalition) and $P_1 = \{1, 1, 1, \dots\}$. This makes it possible to establish tighter global bounds, but also makes it possible to improve performance by using a rather elegant coalition ordering mechanism [61]. This ordering mechanism makes it possible to quickly calculate the values of coalition structures. In our case, we do not do this, even though there could potentially be a way to use a coalition ordering mechanism to improve the performance of our algorithm. Using such ordering mechanisms for collaboration formation could be the subject of future research, and is not further discussed here.

In any case, we present two different approaches to generating the order of precedence for partition expansions.

Algorithm 2 A sorting comparator, based on counting zeroes, used to generate the order of precedence for partition expansions.

```

1: function SORTINGCOMPARATOR( $P_i, P_j$ )
2:    $Z_i \leftarrow$  the number of zeroes in  $P_i$ 
3:    $Z_j \leftarrow$  the number of zeroes in  $P_j$ 
4:   if  $Z_i \neq Z_j$  then
5:     if  $Z_i > Z_j$  then
6:       return  $P_i > P_j$ 
7:     else
8:       return  $P_i < P_j$ 
9:     end if
10:  else
11:     $U_i \leftarrow$  the upper bound of  $P_i$ 
12:     $U_j \leftarrow$  the upper bound of  $P_j$ 
13:    if  $U_i = U_j$  then
14:       $L_i \leftarrow$  the lower bound of  $P_i$ 
15:       $L_j \leftarrow$  the lower bound of  $P_j$ 
16:      if  $L_i > L_j$  then
17:        return  $P_i > P_j$ 
18:      else
19:        return  $P_i < P_j$ 
20:      end if
21:    else
22:      if  $U_i > U_j$  then
23:        return  $P_i > P_j$ 
24:      else
25:        return  $P_i < P_j$ 
26:      end if
27:    end if
28:  end if
29: end function

```

The first approach to generating the order of precedence for partition expansions is based on the intuition that it is possible for us to quickly search through partitions with many zeroes (since there is only one way to assign zero agents to a given task). Therefore, it seems promising to use an expansion order that is based on the number of zeroes per partition, so that we search partitions that have the most zeroes first (in order to establish new global bounds early). Also, there can be many partitions with the same number of zeroes in a single problem instance (e.g. $\{5, 1, 0, 0, 0\}$, $\{4, 2, 0, 0, 0\}$, and $\{3, 3, 0, 0, 0\}$ for $|A| = 6$ and $|T| = 5$), making it hypothetically possible to simultaneously compute new upper bounds for multiple subpartitions.

If we decide to base our sorting on the number of zeroes in each partition, we can define a secondary sorting criterion when two partitions have the same number of zeroes. It seems

rather intuitive to use the bounds of the partitions as this criterion, since the bounds give an indication to the quality of the solutions that we can find inside of a given partition. As such, we can use the sorting comparator in *Algorithm 2* to dictate the order of precedence for the partition expansions, in which the return statement $P_i > P_j$ denotes that the partition P_i should be expanded before the partition P_j .

The second approach to generating the order of precedence for partition expansions is using the secondary sorting criterion from the first approach. This sorting comparator is detailed in *Algorithm 3*.

Algorithm 3 A sorting comparator, based on the lower and upper bounds of partitions, used to generate the order of precedence for partition expansions.

```

1: function SORTINGCOMPARATOR( $P_i, P_j$ )
2:    $U_i \leftarrow$  the upper bound of  $P_i$ 
3:    $U_j \leftarrow$  the upper bound of  $P_j$ 
4:   if  $U_i = U_j$  then
5:      $L_i \leftarrow$  the lower bound of  $P_i$ 
6:      $L_j \leftarrow$  the lower bound of  $P_j$ 
7:     if  $L_i > L_j$  then
8:       return  $P_i > P_j$ 
9:     else
10:      return  $P_i < P_j$ 
11:    end if
12:  else
13:    if  $U_i > U_j$  then
14:      return  $P_i > P_j$ 
15:    else
16:      return  $P_i < P_j$ 
17:    end if
18:  end if
19: end function

```

2.9 Searching for the optimal collaboration structure

Our search algorithm is based on searching through (i.e. expanding) one partition $P \in Q$ at a time. When a partition is expanded, it is first divided into several subpartitions (also denoted refinements). Each subpartition is then evaluated by calculating its bounds, and discarded⁷ (pruned) if it has an upper bound that is lower than (or equal) to the value of the best solution that we have already found. This procedure is shown in *Algorithm 6*, which we present after we have discussed evaluation, generation, refinement and pruning of subpartitions.

If a subpartition p is not pruned, we start searching p by iterating over all $\Phi \in p$. During this search (i.e. when we are generating collaboration structures), we can keep track of the best **possible** value that we can achieve (not to be confused with the best value found thus far) during a search within that subpartition, and can therefore discard sub-optimal solutions before they have been completely generated. If a valid collaboration structure has been generated, however, it is evaluated and compared to the previously best found collaboration structure. If its value is better (i.e. higher) than any value that we have previously found, the new collaboration structure is stored as a potential optimal solution, and the old candidate is safely discarded.

Also, if the algorithm is prompted to pause, then store the current search state and terminate. If the algorithm is prompted to terminate, then return the best collaboration structure found so far.

⁷We can also use the global lower bound to discard subpartitions, since subpartitions have tighter bounds than their parent-partitions.

The aforementioned searching phase can now be described using the following four procedures:

1. Expand partitions. If any expanded partition has a low upper bound, discard it.
2. Divide expanded partitions into subpartitions (i.e. refinements), and iterate over them. In other words, for each subpartition of a given partition, calculate its upper bound, and if the upper bound is too low, discard it. Otherwise, search through it by generating the collaboration structures it represents. If a collaboration structure that is being generated cannot possibly be better than the best solution that has been found so far, stop generating said collaboration structure immediately.
3. Continuously update the global lower bound with the value of the best solution found so far, and interrupt any recursion that cannot possibly produce an optimal (or "good enough") solution.
4. If the search is prompted to pause, store the search state. If the search is prompted to terminate, return the best collaboration structure found so far.

The way that the subpartitions (i.e. refinements of partitions) work is integral to the performance of our algorithm. The subpartitions contain no redundancy, and are created using the fact that each partition is a **unique** multiset of integers (i.e. none of the partitions have exactly the same integer members).

In the case of coalition formation, the order in which coalitions are formed does not matter, and the (unordered) partition $P = \langle 2, 0, 0 \rangle$ would represent a set of possible coalition structures. In our case, P represents a much larger set of possible collaboration structures, since the order of coalitions in a collaboration structure matters. For instance, the lists $X = \langle 2, 0, 0 \rangle$, $Y = \langle 0, 2, 0 \rangle$ and $Z = \langle 0, 0, 2 \rangle$ — that we shall later denote as the subpartitions of P — represents the same set of coalition structures, while they represent three very different disjoint sets of collaboration structures.

We now define the **subpartition** p as the list of integers in which its members are a multiset permutation of the integers in its (unordered) "parent"-partition P . For example, the partition $P = \{2, 0, 0\}$ induces the 3 different subpartitions $p_1 = \langle 2, 0, 0 \rangle$, $p_2 = \langle 0, 2, 0 \rangle$, and $p_3 = \langle 0, 0, 2 \rangle$.

Recall that there are equally many tasks as there are members in a partition. This is also true for subpartitions, since every subpartition has equally many members as its parent-partition (i.e. the partition for which the subpartition is a multiset permutation of). Therefore, $|T| = |P| = |p|$, and we can map each coalition assigned to a task $t \in T$ in a collaboration structure $\Phi_{A,T}$ to an element in the subpartition p using their indices. As such, we can use any subpartition $p = \langle k_1, k_2, k_3, \dots \rangle$ to represent a set of collaboration structures, in which k_i corresponds to assigning k_i agents to task i . More specifically, if $|C_i|$ denotes the number of members assigned to the task with index $i \in I_\Phi$, and k_i denotes the i :th element of p , we can use p to represent all collaboration structures in which $|C_i| = k_i$ is true for all indices $i \in I_\Phi$.

Since every partition P is (represented by) a unique set of integers, each subpartition is also a unique list of integers. The reason is that each subpartition can only have the same members as other subpartitions of its parent-partition (since partitions have unique members, as previously discussed), and since each subpartition is a multiset permutation of its parent-partition, two subpartitions of a single partition cannot be the same. Therefore, every subpartition is unique, and represents a unique set of collaboration structures. We can therefore guarantee that no collaboration structure is generated more than once. The algorithm we use to search through a subpartition p is presented in *Algorithm 4*. In the pseudocode for our algorithms, we will be using \emptyset to denote an assignment vector Φ with no assigned agents.

Algorithm 4 An algorithm that searches through a given subpartition.

```

1: procedure SEARCHSUBPARTITION( $p, v_{intermediary}, u_{current}, i, \Phi_{temp}, \Phi_{best}$ )
2:   if  $i > |A|$  then
3:     if  $\Phi_{best} = \emptyset$  then
4:        $L_G \leftarrow \max(L_G, v_{intermediary})$ 
5:        $\Phi_{best} \leftarrow \Phi_{temp}$ 
6:     else if  $v_{intermediary} > L_G$  then
7:        $L_G \leftarrow v_{intermediary}$ 
8:        $\Phi_{best} \leftarrow \Phi_{temp}$ 
9:     end if
10:    return
11:  end if
12:   $v_{possible} \leftarrow u_{current} + v_{intermediary}$ 
13:  for all  $k \in p$  do
14:    if  $\Phi_{best} \neq \emptyset$  then
15:      if prompted to interrupt or terminate then
16:        return
17:      end if
18:    end if
19:    if  $v_{possible} < L_G$  then
20:      return
21:    end if
22:    if the task in  $\Phi_{temp}$  that corresponds to  $k$  has  $k$  agents assigned then
23:      continue
24:    end if
25:    assign agent  $a_i$  to the coalition in  $\Phi_{temp}$  that corresponds to  $k$ 
26:     $v_{temp} \leftarrow 0$ 
27:     $u_{temp} \leftarrow 0$ 
28:    if the task in  $\Phi_{temp}$  that corresponds to  $k$  has  $k$  agents assigned then
29:       $v_{temp} \leftarrow$  the coalition value of the coalition that corresponds to  $k$ 
30:       $u_{temp} \leftarrow$  the maximum cardinal task value of the task that corresponds to  $k$ 
31:    end if
32:     $v_{new} \leftarrow v_{intermediary} + v_{temp}$ 
33:     $u_{new} \leftarrow u_{current} - u_{temp}$ 
34:    SEARCHSUBPARTITION( $p, v_{new}, u_{new}, i + 1, \Phi_{temp}, \Phi_{best}$ )
35:    de-assign agent  $a_i$  from the coalition in  $\Phi_{temp}$  that corresponds to  $k$ 
36:  end for
37: end procedure

```

We haven't discussed how to generate multiset permutations of a given partition. However, there already exists several ways to generate all permutations (i.e. subpartitions) of a given multiset. One very simple way to do so is to generate all possible (single-set) permutations, and discard multisets until we have no redundancy (i.e. copies of the same multiset permutation). A better solution would be to use an algorithm that generates multiset permutations without redundancy, such as the tree-traversal algorithm proposed by Takaoka in [76], or the algorithm based on loopless generation of multiset permutations presented by Williams in [87]. There also exists several programming libraries that can generate multiset permutations efficiently, such as the C++ standard library (e.g. `std::next_permutation`, which we will be using).

We previously mentioned that we can search through partitions with many zeroes quickly. This is due to the fact that, given a partition P , we generate a number of subpartitions that each has the same number of zeroes as P . For every zero in P , we can calculate a tighter bound for the subpartitions of P (before searching), since a zero in a subpartition represents a task that should have zero agents assigned to it, and there is only one way to do so (per task). We can therefore instantly calculate the value that all such tasks would have contributed

to the final result, and use this information to update our subpartition bounds (and in some cases even discard the subpartition entirely). Additionally, we can always discard the partition addends that corresponds to these tasks, since there is no point in further evaluating the tasks that they correspond to, as long as we keep track of their contributed value. We use *Algorithm 5* to refine a subpartition prior to searching (or discarding) it.

Algorithm 5 An algorithm that refines a subpartition

```

1: procedure REFINESUBPARTITION( $p, T = \{t_1, t_2, \dots\}$ )
2:    $pre\_fined \leftarrow \emptyset$ 
3:    $u \leftarrow 0$ 
4:    $v \leftarrow 0$ 
5:    $i \leftarrow 1$ 
6:   while  $i \leq |p|$  do
7:     if  $p[i] = 0$  then
8:        $t \leftarrow$  the utility value of assigning 0 agents to task  $t_i$ 
9:        $v \leftarrow v + t$ 
10:    else
11:       $t \leftarrow$  the upper bound of assigning  $p[i]$  agents to task  $t_i$ 
12:       $u \leftarrow u + t$ 
13:      insert  $p[i]$  to  $pre\_fined$ 
14:    end if
15:     $i \leftarrow i + 1$ 
16:  end while
17:  the upper bound of  $p \leftarrow u$ 
18:  the intermediary value of  $p \leftarrow v$ 
19:   $p \leftarrow pre\_fined$ 
20: end procedure

```

We have now discussed and shown:

1. How to partition the collaboration structure search space.
2. How to search through partitions using subpartition pruning.
3. How to potentially improve search performance by generating refinements.
4. Two different approaches to deciding the precedence order for the expansion of partitions.

In the next section, we show how assemble all of the aforementioned techniques and concepts into a final algorithm for collaboration formation.

2.10 An algorithm for collaboration formation

In this chapter, we present our main algorithm for collaboration formation. First, however, we present *Algorithm 6*, which is the aforementioned search algorithm that finds the optimal collaboration structure given a set of partitions $Q = \{P_1, P_2, \dots\}$:

Algorithm 6 A collaboration structure search algorithm based on branch-and-bound.

```

1: function FINDOPTIMALCOLLABORATIONFORMATION( $Q = \{P_1, P_2, \dots\}$ )
2:    $\Phi_{best} \leftarrow \emptyset$ 
3:   for all  $P \in Q$  do
4:     for all subpartitions  $p$  induced by  $P$  do
5:       REFINESUBPARTITION( $p$ )
6:        $u \leftarrow$  the upper bound of  $p$ 
7:       if  $u > L_G$  then
8:          $v \leftarrow$  the intermediary value of  $p$ 
9:         SEARCHSUBPARTITION( $p, v, u, 1, \emptyset, \Phi_{best}$ )
10:      end if
11:      if prompted to interrupt or terminate then
12:        go to 13
13:      end if
14:      if prompted to pause then
15:        save search state
16:        go to 13
17:      end if
18:    end for
19:  end for
20:  return  $\Phi_{best}$ 
21: end function

```

Finally, we present *Algorithm 7*, which is our main algorithm that uses all the aforementioned algorithms and techniques to solve a given collaboration formation problem optimally:

Algorithm 7 A collaboration formation algorithm that solves SCAP instances optimally.

```

1: function SOLVESCAPOPTIMALLY( $A = \{a_1, a_2, \dots\}, T = \{t_1, t_2, \dots\}$ )
2:    $n \leftarrow |A|$ 
3:    $m \leftarrow |T|$ 
4:    $Q \leftarrow$  GENERATEALLINTEGERPARTITIONSOFSIZE( $n, m$ )
5:   calculate all coalition values
6:   calculate all cardinal values
7:   calculate all cardinal task values
8:   calculate the lower and upper bounds of all partitions in  $Q$ 
9:   calculate the global bounds  $L_G$  and  $U_G$ 
10:  count the number of zeroes for each partition in  $Q$ 
11:  sort  $Q$  as ordered by SORTINGCOMPARATOR
12:  return FINDOPTIMALCOLLABORATIONFORMATION( $Q$ )
13: end function

```

Chapter 3

Evaluation

This chapter describes how the experiments of this thesis are to be conducted, and how the results of this thesis are to be achieved. Furthermore, it discusses the validity, reliability, and replicability of our tests. Additionally, we present a few arguments to why our tests should be able to empirically show whether our algorithm manages to successfully satisfy the properties it was designed for (e.g. anytime). Finally, we discuss the implementation that we will use for the tests, and the equipment that we will use to conduct the experiments (i.e. the performance benchmarking and quality evaluation).

3.1 Performance benchmarking and quality evaluation

As previously discussed, we will be evaluating the performance and quality of our algorithm in two different scenarios:

1. Solving the army to region assignment problem in Europa Universalis 4.
2. Solving (abstract) simulated standardized problem instances.

These two scenarios are discussed in the next two subsections.

Tests and experiments using Europa Universalis 4

By using our algorithm to solve real SCAP instances, or more specifically simultaneous coalition formation and assignment problems that already exists in a commercial real-time strategy game, we hope to show that our algorithm can be used to improve collaborative reasoning of agents in real-world multi-agent systems. This evaluation will be accomplished by looking at the specific problem of assigning armies to regions in the real-time strategy game Europa Universalis 4 (EU4). To test our algorithm, we will use problem instances taken straight from EU4, where utility values are calculated by using previously defined utility functions that are already being used in the released (i.e. public) version of the game. As such, we have not tweaked any utility function to make our algorithm more efficient prior to integrating it into EU4.

In EU4, there exists predefined scenarios denoted *bookmarks* [27]. These bookmarks are available to anyone who owns a license of EU4, and are frequently played by human players. We will use two of these bookmarks to evaluate our algorithm, since they perfectly represent the problem instances that occur during play. More specifically, we will use the two following bookmarks to evaluate our algorithm:

1. **Thirty Years War** (23 May, 1618)
2. **Seven Years War** (15 May, 1756)

These scenarios were chosen due to the fact that they are widely different in terms of initial setup, and can thus hopefully provide a wide set of different assignment problems. They also

provide many assignment problems in a short duration of time, since many nations are at war with each other when the scenarios are started. Using these two scenarios, we can not only deduce whether our algorithm can actually process real SCAP instances in real-time, but also compare our algorithm to the algorithm that is currently being used in EU4 to solve the army to region assignment problem. We will denote the algorithm that is currently being used in EU4 the *Europa Universalis Monte Carlo* (EUMC) algorithm, since it generates thousands of randomized assignment samples to find collaboration structures. Additionally, the EUMC is specifically designed to handle army to region assignments, while our algorithm is a generalized collaboration formation algorithm that can solve any SCAP. Therefore, it would show great promise if our algorithm manages to generate good solutions more efficiently than EUMC, since that would indicate that our algorithm is a viable option to specialized algorithms that are already being used in real-world applications.

All problem sets that are generated during a simulated scenario consists of a set of armies (agents) and a set of regions (tasks). Since each region is unique, each region must be treated as a unique type of task.

If searching for the optimal collaboration structure takes too long, or if there are too many agents or tasks so that the algorithm don't have time to calculate the coalition values, we will force our algorithm to interrupt and return an anytime solution, since armies in EU4 need to reason and decide on actions in real-time. In this way, we can test the anytime characteristics of our algorithm, and deduce whether it has managed to prune sub-spaces that only contain bad solutions. We will also look at the search times for collaboration formation, and see if our algorithm manages to increase performance when searching for optimal solutions.

If $|T| \cdot 2^{|A|}$ is too large (i.e. if $|T| \cdot 2^{|A|} > 10^6$, we will not even attempt to use our algorithm to find a collaboration structure, since in such cases, the pre-computational (i.e. calculating the coalition and cardinal values) phase is too computationally expensive. Therefore, if this happens, we will fall back on generating a randomized solution, and use that as the basis for comparison. When this is the case, we expect our algorithm to generate much worse solutions than EUMC. However, $|T| \cdot 2^{|A|}$ being that large is not very likely, since nations in EU4 rarely controls more armies than 8 and rarely attempts to assign armies to more regions than 30, and $30 \cdot 2^8 < 10^4 \ll 10^6$.

Since EU4 is non-deterministic, we will run the two scenarios several times each, since each run can generate different problem sets. This can hopefully increase the validity and reliability of our tests, since it makes it possible to generate a broader set of different assignment problems from the same initial problem description.

When a scenario is run, we will let the game run for 20 in-game years, which translates to roughly 7300 in-game (daily) updates (i.e. ~ 7300 days). Both our algorithm and the EUMC will be called whenever the computer-based players decide to assign armies to regions, and their solutions will be stored, so that the results of the two algorithms can be compared and discussed.

We argue that the aforementioned tests will deduce whether our algorithm is sufficiently anytime for real-time scenarios, and whether our algorithm is able to sufficiently prune the collaboration structure search space, since this would indeed be the case if our algorithm's anytime solutions are better than those generated by EUMC.

Finally, and before moving on to the next subsection, we first want to give the reader an idea of how the army assignment utility function in EU4 works. First of all, the utility function used for army assignment in EU4 is based on a wide range of criteria and variables. As such, it's rather complex, and the description here will be rather vague. However, this shouldn't be a problem, since we are only interested in comparing the performance characteristics of EUMC and our algorithm. In any case, the utility function is based on calculating a relative army strength per region, which is used as a basis to decide whether it is advantageous to assign an army to a region. Collaboration formation can thus be used to decide whether it is a good idea to assign several armies to the same region, or whether certain assignments would create weaknesses in other regions.

Performance benchmarking and quality evaluation in simulated standardized problem instances

A common approach to evaluating performance of search algorithms is to use standardized problem instances for benchmarking. In the case of collaboration formation (and to our knowledge), no such standardized problem instances exist. Therefore, we look at standardized problem instances from a similar domain. More specifically, we translate the standardized problem instances used for benchmarking coalition formation algorithms to the domain of collaboration formation. Larson and Sandholm [48] provided such instances for the coalition formation problem, namely using normal and uniform probability distributions (NPD and UPD, respectively) to provide randomized coalition values. These probability distributions have then been used to evaluate the performance of state-of-the-art coalition formation algorithms (e.g. [64]).

In the case of collaboration formation, coalition values represent the values of assigning a coalition to a given task. Therefore, we can directly translate the coalition values provided by the probability distributions to coalition values in SCAP.

Since the coalition formation problem is closely related to SCAP, we hypothesize that these probability distributions will be able to push our algorithm to its limits, and that they will give a clear indication of whether our algorithm is a viable approach to collaboration formation. Finally, these probability distributions are easy to replicate and re-create, and we deem that they will increase the replicability of this study.

However, using these distributions to generate coalition values lead to biased optimal coalition structures, as shown by Rawhan et al. in [64, p. 548]. It is reasonable to assume that this is also the case for collaboration structures, since the bias is due to the fact that small coalitions are given similar values as large, thus making coalition structures with many small coalitions have a higher probability of receiving a high utility value. Rawhan et al. [64, p. 549] proposed another probability distribution that do not generate such biased coalition structures, namely NDCS (Normally Distributed Coalition Structures). We will also use NDCS to benchmark our algorithm, since NDCS will be able to generate unbiased coalition values for collaboration structures (for the same reason as it does so for coalition structures). More specifically, the probability distributions that we use to generate utility values are:

- **NPD:** $v(C, t) \sim |C| \times \mathcal{N}(\mu, \sigma^2)$, where $\sigma = 0.1$ and $\mu = 1$.
- **NDCS:** $v(C, t) \sim \mathcal{N}(\mu, \sigma^2)$, where $\sigma = \sqrt{|C|}$ and $\mu = |C|$.
- **UPD:** $v(C, t) \sim |C| \times \mathcal{U}(a, b)$, where $a = 0$ and $b = 1$.

(\mathcal{V} is the random variable used to assign coalition values)

The result of each experiment will be produced by calculating the average of the resulting values (e.g. time measures or utility values) from 100 generated problem sets per probability distribution. We deem that this should be sufficient to give a clear indication to the behaviour of the algorithms.

Finally, we will compare our algorithm to brute-force in a few simple problem instances where there are few agents and tasks. When there are many agents and tasks, this is not possible, since brute-force is simply too slow. The reason to why we compare our algorithm to brute-force is because such comparisons can give us an indication to how much of an improvement our algorithm is compared to simply enumerating all possible solutions. The algorithm we use for such purposes is presented in *Algorithm 8*, and is a very simple collaboration formation algorithm that is based on using recursion to enumerate all possible collaboration structures.

Algorithm 8 A recursive brute-force algorithm for collaboration formation.

```

1: procedure BRUTEFORCECF( $A = \{a_0, a_1, \dots\}, T = \{t_0, t_1, \dots\}, \Phi_{temp}, \Phi_{best}, v_{best}, i$ )
2:   if  $i = |A|$  then
3:      $v_{temp} \leftarrow 0$ 
4:     for all  $j \in I_\Phi$  do
5:        $v_{coalition} \leftarrow$  value of assigning coalition  $C_j \in \Phi_{temp}$  to  $t_j$ 
6:        $v_{temp} \leftarrow v_{temp} + v_{coalition}$ 
7:     end for
8:     if  $\Phi_{best} = \emptyset$  or  $v_{temp} > v_{best}$  then
9:        $\Phi_{best} \leftarrow \Phi_{temp}, v_{best} = v_{temp}$ 
10:    end if
11:  end if
12:  for all  $t \in T$  do
13:    assign agent  $a_i$  to the coalition in  $\Phi_{temp}$  that corresponds to  $t$ 
14:    BRUTEFORCECF( $A, T, \Phi_{temp}, \Phi_{best}, v_{best}, i + 1$ )
15:    de-assign agent  $a_i$  from the coalition in  $\Phi_{temp}$  that corresponds to  $t$ 
16:  end for
17: end procedure

```

3.2 Implementation and equipment

All algorithms we used for benchmarking and evaluation were implemented in C++11. Additionally, all implementations used the C++ standard library [14]. For example, we used `std::algorithm::next_permutation` to generate multiset permutations (subpartitions), and `std::sort` to generate the order of precedence for the partitions prior to searching them. The source code was compiled using GCC (version 5.3.0), with the default optimization level.

All time measurements were made using `std::chrono`, and more specifically using the monotonic clock `std::chrono::steady_clock` for time keeping.

Finally, all probability distributions were generated using the `std::random` header from the C++ standard library, and more specifically the random number distribution generator `std::random::normal_distribution<double>` for NDCS and NPD, and `std::random::uniform_real_distribution<double>` for UPD.

We also used `std::algorithm::random_shuffle` to generate randomized precedence orders in our two final tests.

The functions, headers and generators that we used are all used in a multitude of real-world applications, and any major bugs or defects should already have been found and dealt with. We therefore argue that the random numbers generated by `std::random` are of good enough quality for our rather blunt experiments, and that `std::chrono` should be sufficient for our time keeping, even though its precision for small sample sizes may influence the results slightly.

All tests were conducted using a computer with Windows 10 (x64), an Intel 7700K central processing unit with a base frequency of 4200MHz, and 16GB of DDR4 memory (3000MHz CL15). Today, this is regarded as a high-end computer, and we deem that using such computer is appropriate for testing an algorithm that could potentially be used in many future computer games and real-world applications. However, it has a much higher computational power than most of the computers that are being used by humans playing EU4, and we will therefore use a rather short timer for premature interruption for the experiments that are conducted in EU4. More specifically, if our algorithm cannot find a solution before 0.01 seconds has elapsed, it will be interrupted, and an anytime solution will be returned.

3.3 Evaluation and benchmarking results

This section presents the results that were achieved by using the procedures (i.e. methods) discussed in the previous sections. In the first subsection, we provide the results from the tests that were conducted in Europa Universalis 4 (EU4). In the second subsection, we present the results from the experiments conducted in the simulated problem instances (i.e. the problem sets generated by UPD, NPD and NDCS). In order to make this chapter less prone to misconceptions, we will denote our algorithm **CSGen** (short for *Collaboration Structure Generator*) from now on. Additionally, and if nothing else is mentioned, we will be using the approach based on counting zeroes for generating the precedence order of partition expansions.

Results of the experiments conducted in Europa Universalis 4

The way CSGen (i.e. our algorithm) and EUMC (i.e. the algorithm that is currently being used in EU4) behave depend significantly on the number of agents (armies) and tasks (regions) in the problem descriptions that they are given. In the case of CSGen, if these numbers are too high, it may fail to find any solution at all, since its pre-computation phase may take too long. In such case, it will just return a random result. In the case of EUMC, the generated solutions will quickly get worse as the number of agents and tasks increases. This is mainly due to how un-guided Monte Carlo algorithms works, and due to EUMC not taking advantage of the problem description (e.g. EUMC doesn't know before-hand on which of the samples that have a high probability of being "good"). The scatter plot in figure 3.1 shows a visualization of the samples that were generated using EU4, and gives an indication to the sizes of the problems that were generated during our experiments, as to give a hint on the suitability of using the two different algorithms for army assignment in EU4.

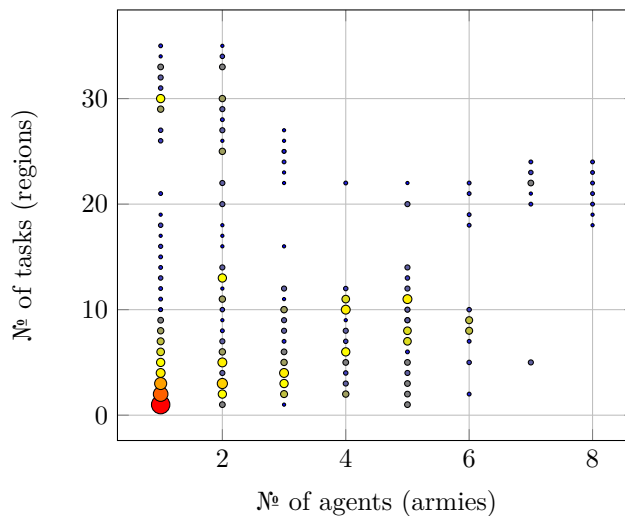


Figure 3.1: A visualization of the samples that were generated using EU4. A larger scatter mark indicates that there were more samples for that given number of agents and tasks.

During our tests, CSGen never failed to find a solution. In other words, our algorithm didn't have to fall back on generating a randomized collaboration structure, since $|T|^{|A|}$ was rather small in all of the samples (as can be seen in the scatter plot).

Now, in the next two subsections, we present the results from each of the two experiments conducted in EU4.

Benchmark 1: The Thirty Years War

We ran the first scenario 3 times (i.e. we let EU4 run 20 in-game years from the initial setup of bookmark 1) and gathered a total number of 1983, 2179, and 1748 samples from run 1, 2, and 3, respectively. The results of these runs are shown in table 3.1.

Algorithm	Time Elapsed [ms]		Utility Value	
	Total	Average	Total	Average
EUMC (<i>run 1</i>)	21162.4	10.7	2692.2	1.4
CSGen (<i>run 1</i>)	2967.2	1.5	12747.0	6.4
EUMC (<i>run 2</i>)	24244.2	11.1	2897.5	1.3
CSGen (<i>run 2</i>)	3829.6	1.8	16486.8	7.6
EUMC (<i>run 3</i>)	18359.1	10.5	2301.8	1.3
CSGen (<i>run 3</i>)	2695.7	1.5	10177.4	5.8

Table 3.1: The results from running the first bookmark (i.e. **Thirty Years War**) for 20 in-game years.

Benchmark 2: The Seven Years War

We ran the second scenario 3 times (i.e. we let EU4 run 20 in-game years from the initial setup of bookmark 2) and gathered a total number of 2729, 2571, and 2712 samples from run 1, 2, and 3, respectively. The results of these runs are shown in table 3.2.

Algorithm	Time Elapsed [ms]		Utility Value	
	Total	Average	Total	Average
EUMC (<i>run 1</i>)	37462.3	13.7	3508.0	1.3
CSGen (<i>run 1</i>)	13275.0	4.9	17905.0	6.6
EUMC (<i>run 2</i>)	37803.6	14.7	3450.1	1.3
CSGen (<i>run 2</i>)	9000.9	3.5	21496.1	8.4
EUMC (<i>run 3</i>)	44288.9	16.3	3591.0	1.3
CSGen (<i>run 3</i>)	11634.4	4.3	25441.0	9.4

Table 3.2: The results from running the second bookmark (i.e. **Seven Years War**) for 20 in-game years.

Results of the benchmarks on simulated problem instances

The time it takes for CSGen to find optimal solutions for the fixed numbers of 6, 12, and 18 tasks (with evaluation functions that return values as determined by the aforementioned probability functions NPD, UPD and NDCS) are plotted using a log scale in figure 3.2, 3.3 and 3.4, respectively. A plot of the search times for the brute-force algorithm is used as a comparison.

The brute-force algorithm is indifferent to the choice of probability distribution for generating coalition values, since it is indifferent to the values of the coalitions (it just enumerates all solutions). Therefore, the choice of coalition values does not matter when testing the brute-force algorithm, and we simply used the coalition values generated by UPD for the tests conducted with the brute-force algorithm.

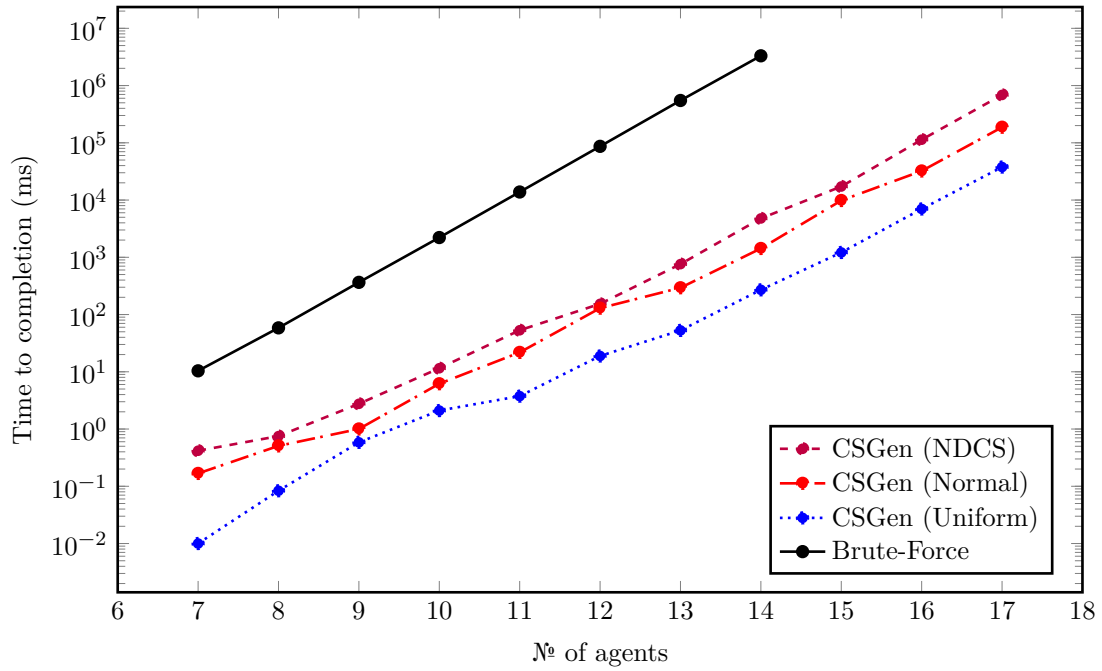


Figure 3.2: A graph that shows the time it takes to find the optimal solution for CSGen in the simulated problem instances that have 6 tasks that agents are to be assigned to.

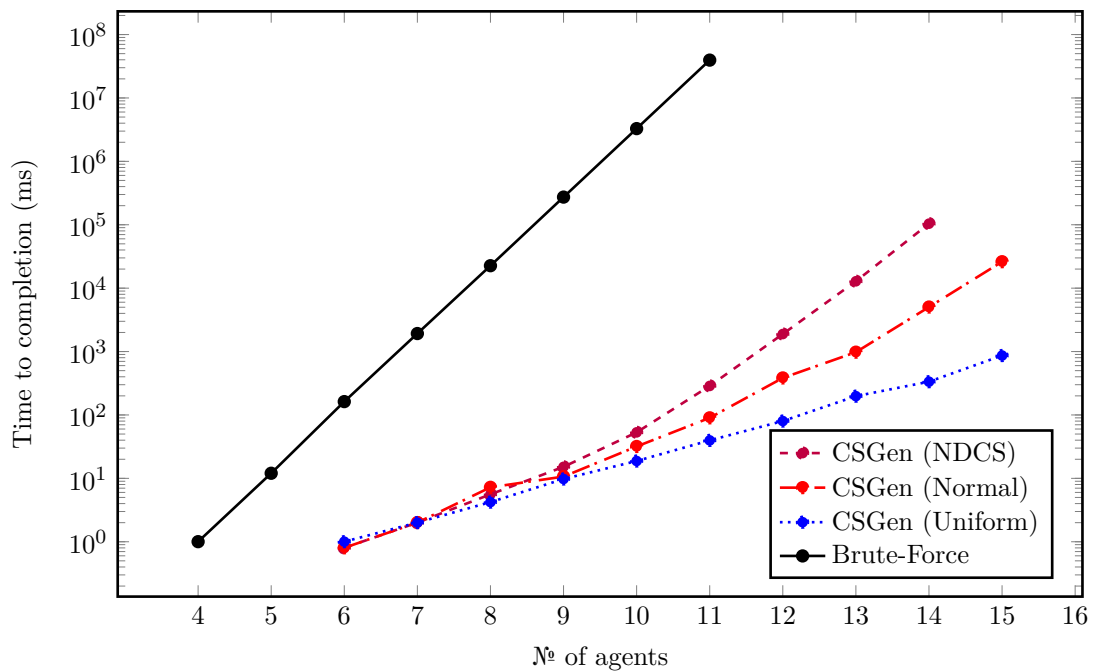


Figure 3.3: A graph that shows the time it takes for CSGen to find the optimal collaboration structure in simulated problem instances with 12 tasks that agents are to be assigned to.

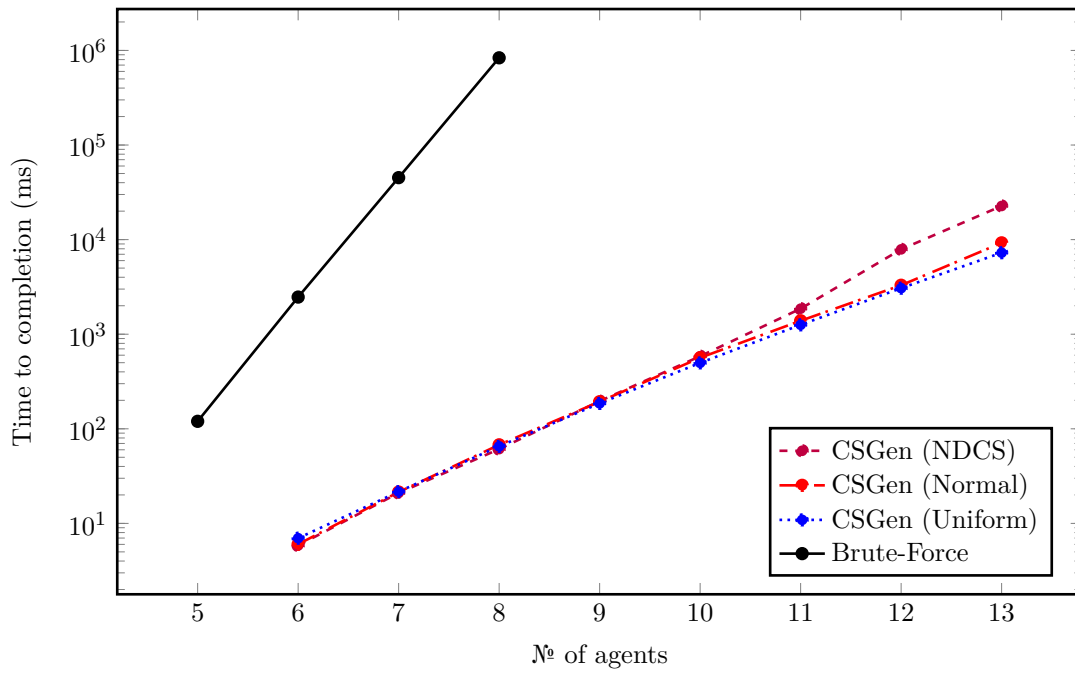


Figure 3.4: A graph that shows the time it takes for CSGen to find the optimal collaboration structure in simulated problem instances with 18 tasks that agents are to be assigned to.

In the next graph, i.e. in figure 3.5, we fix the number of agents to 10, and instead look at how the number of tasks affect performance.

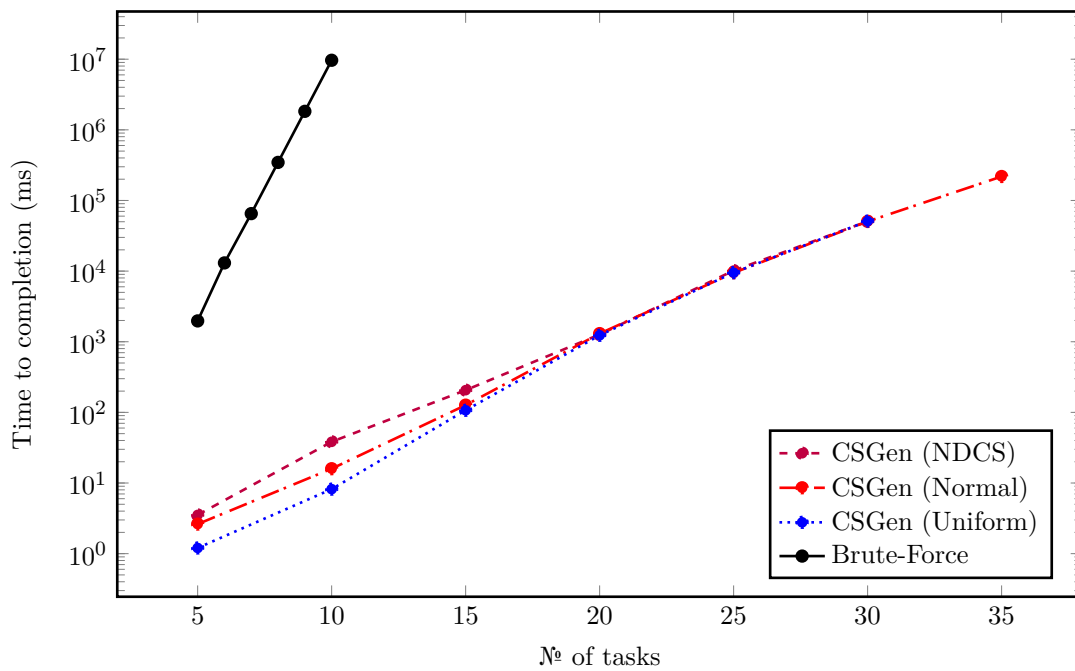


Figure 3.5: A graph that shows the time it takes for CSGen to find the optimal collaboration structure in simulated problem instances with 10 agents that are to be assigned to tasks.

Now that we have shown our results from the benchmarks and the experiments that were made to evaluate the performance of exhaustive searches, we move on to the experiments that were made to evaluate the any-time solutions that CSGen generates. We used 12 agents and 8 tasks for this purpose, and interrupted the algorithm during search by only letting it refine a fixed number of subpartitions. The total number of subpartitions for 12 agents and 8 tasks is 50388. The results of this experiment can be seen in figure 3.6. On the y -axis, we show the utility value U of the collaboration structures that CSGen had found on interruption, divided by the value U_{opt}^* of the optimal collaboration structure (as found by a brute-force search). We show the same test in figure 3.7, where we used the number of evaluated collaboration structures instead of subpartitions.

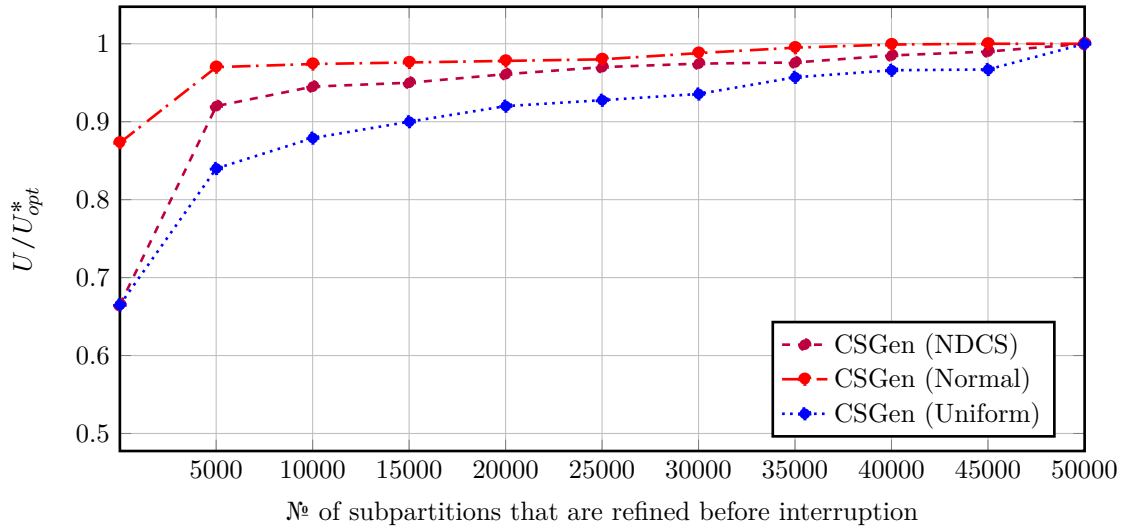


Figure 3.6: A graph that shows how far from the optimal solution the algorithm is when it is interrupted prior to finishing an exhaustive search.

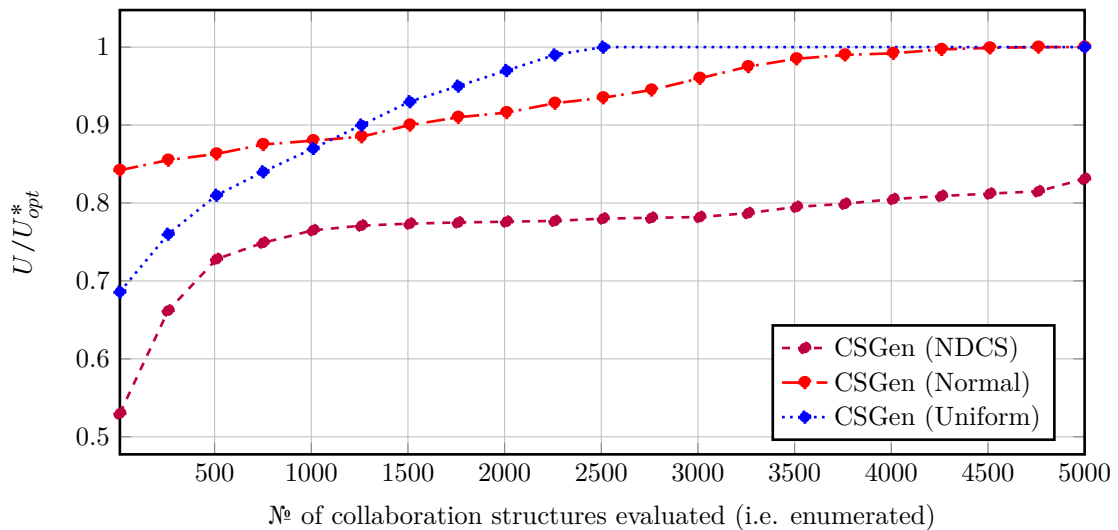


Figure 3.7: A graph that shows how far from the optimal solution the algorithm is when it is interrupted prior to finishing an exhaustive search.

To conclude this evaluation chapter, we now present the results of our two final tests. These tests were aimed at showcasing how the precedence order of partitions can affect the performance of CSGen. In order to make a reasonable and valid comparison, we compared the two different approaches to using a randomly generated order of precedence. We did this by plotting the quotient T_{random}/T_{order} , where T_{random} denotes the average search time for using a randomly generated order of precedence, and T_{order} denotes the average search time for using one of the two aforementioned approaches to generating precedence orders, in two separate graphs. 10 agents were used in all of these tests, and we used 1000 samples per test case.

In figure 3.8, we show how the performance was affected by using the order of precedence from algorithm 2 (i.e. the precedence order that is based on counting the number of zeroes in each partition). In figure 3.9, we show how the performance was affected by using the order of precedence from algorithm 3 (i.e. the precedence order that is based on the upper and lower bounds of each partition).

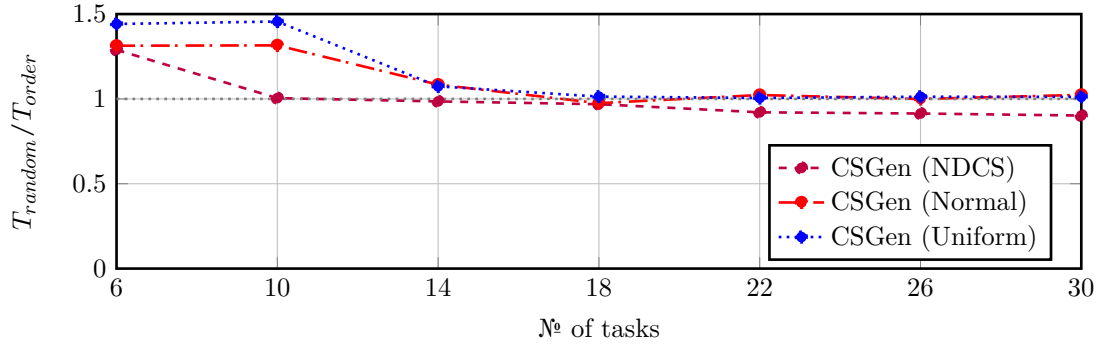


Figure 3.8: This graph shows how the performance was affected by using the order of precedence from algorithm 2 (i.e. the precedence order that is based on counting the number of zeroes in each partition). T_{random} denotes the search time for using a random order of precedence, while T_{order} denotes the search time for using the precedence order based on counting the numbers of zeroes in each partition.

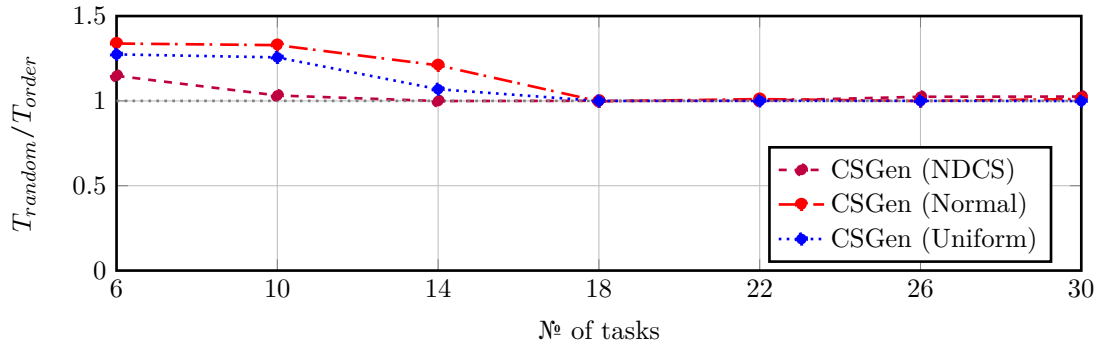


Figure 3.9: This graph shows how the performance was affected by using the order of precedence from algorithm 3 (i.e. the precedence order that is based on the upper and lower bounds of each partition). T_{random} denotes the search time for using a random order of precedence, while T_{order} denotes the search time for using the precedence order based on the upper and lower bounds for each partition.

Chapter 4

Discussion

In this chapter, we discuss and analyze the applied method and our results. We begin by analyzing the results from Europa Universalis 4 (EU4), and then move on to looking at the results from the simulated problem instances.

4.1 Analysis of the experiments conducted in Europa Universalis 4

As can be seen in the scatter plot in figure 3.1, the problem set generated by EU4 is neither distributed using a uniform nor normal probability distribution. Additionally, the number of agents (armies) is much lower than the number of tasks (regions) in almost all samples. This is due to the fact that the computer-based players in EU4 normally only have a few armies at their disposal, while there may be many regions of which every army can be considered to be assigned to. However, a few bigger nations, such as Russia and France, may sometimes have up to 8 armies.

In any case, the largest problem instance that was generated during all of the subsequent runs was an instance of 8 agents (armies) and 24 tasks (regions). A problem description with 8 agents and 24 tasks has $24^8 > 10^{11}$ possible solutions (collaboration structures), but only $24 \cdot 2^8 = 6144$ different task to coalition assignments (and the same number of coalition values). As such, the pre-computational phase of CSGen was never too expensive, since the utility function used for army to region assignment is linear in the number of regions, with the implication of a pre-computational phase of $O(|T|^2 \cdot 2^{|A|})$ time complexity.

Most samples from EU4 were small problem instances, and consisted of only 1-4 agents and 1-12 tasks. Our algorithm was able to solve these small problem instances optimally, while the previously used algorithm, i.e. EUMC, achieved a lower performance measure (i.e. sum of utility values) due to failing to do so. The reason is that EUMC generates several randomized solutions of which many can be the same solution (e.g. even if EUMC generates $|T|^{|A|}$ possible solutions, none of them are guaranteed to be optimal). As such, EUMC generates a lot of redundancy, while CSGen generates none.

By looking at all runs from the first benchmark, i.e. the **Thirty Years War**, we can see that CSGen managed to generate collaboration structures with a utility that is, on average, approximately 503% higher than the utility values of the collaboration structures generated by EUMC. Finally, CSGen also has a much higher computational efficiency, and only needed about 15% of the time that EUMC needed.

By looking at the next scenario, i.e. the **Seven Years War**, we can see that CSGen managed to generate collaboration structures with even higher utility values than those it generated in the first scenario. In this experiment, CSGen generated solutions with utility values that were, on average, approximately 614% higher than the utility values of the solutions that were found by EUMC. CSGen needed, on average, about 28% of the time that EUMC needed to generate its solutions.

On average, CSGen only needed 3.12 milliseconds to generate collaboration structures. It is reasonable to argue that this low number is due to the fact that most of the problem

descriptions were small, as seen in figure 3.1. The performance of EUMC is not affected as much as CSGen by the input size, since it is based on generating a fixed number of solutions, without bothering about how many agents and tasks there are, and then picking the best. However, the quality of the solutions that EUMC provides are highly affected by the input size (e.g. $|T|$ and $|A|$) due to the same reason, while CSGen always manages to provide relatively good solutions (if they exist), at least for all of the sample sizes that were generated by EU4.

The high performance and quality that CSGen exhibits in our tests is plausible, since it is intuitively reasonable that an un-guided Monte Carlo algorithm (EUMC) is outperformed by an optimized algorithm that is based on branch-and-bound (CSGen). Even though our numbers are calculated from just a few scenarios, they should still give a reasonable indication to how CSGen and EUMC performs in other real-world applications. Also, we deem that these tests convey a bigger picture: that CSGen is an efficient generalized approach to collaboration formation, and that CSGen manages to surpass a specialized algorithm (i.e. EUMC) — in both quality and performance.

Our algorithm managed to provide much better solutions than EUMC, even when it was interrupted prior to finishing a search. Due to the small sizes of the samples, and the relatively low average search time, it is reasonable to assume that CSGen often managed to find the optimal solutions, even when it hadn't finished an exhaustive search.

4.2 Analysis of the benchmarks on simulated problem instances

In this section, we will look at the results of the different experiments that were conducted in artificial problem instances. We will start by looking at the performance evaluations and benchmarks, and then at the anytime experiments.

Optimality experiments

In the results of the performance tests in which we had a fixed number of tasks (i.e. in figure 3.2, 3.3 and 3.4), we can see that our algorithm is considerably faster than brute-force for all distributions (i.e. NDCS, NPD and UPD). This was expected, since branch-and-bound has been used to solve similar NP-hard problems effectively (as previously discussed).

CSGen solved the problem sets generated by UPD the quickest, followed by the sets generated by NPD, and then NDCS. This is also not surprising, since it is reasonable to expect that CSGen exhibits performance characteristics that are similar to those exhibited by similar algorithms used for coalition formation (e.g. the IDP and IP algorithms by Rahwan et al. [64, 62]). In the case of 6 tasks, 12 agents, and coalition values generated by UPD, our algorithm is, on average, 862 times better than brute-force (i.e. it takes approximately 0.12% of the time to find the optimal solution). As the number of agents increases, this factor also increases. For example, in the case of 6 tasks and 14 agents, our algorithm is, on average, 1535 times better (i.e. it finds optimal solutions in approximately 0.07% of the time it takes for the brute-force algorithm).

If we increase the number of tasks, the gains in performance increases considerably. This is not surprising, since if there are many subpartitions in which there are tasks with 0 assigned agents, our algorithm can discard considerable sizes of the search space.

Increasing the number of tasks does not impact performance as much as increasing the number of agents. This is empirically shown in figure 3.5, where we fixate the number of agents, and look at how the number of tasks affect the performance. In this experiment, we can see that CSGen is extremely fast in comparison to brute-force at solving problem sets with a high task to agent ratio. The search times for the problem sets with different probability distributions seems to converge as the number of tasks is increased. A reasonable cause could be that CSGen is able to discard many partition refinements with integers that represent tasks with 0 assigned agents.

Anytime experiments

In the two last experiments, we looked at the quality of the anytime solutions that CSGen generates. In figure 3.6, we can see that UPD generates the problem sets for which CSGen is required to refine the most subpartitions before it can return a solution that is close to optimal. This may sound surprising, but is in fact not. The reason is that, for problem sets generated by UPD, CSGen can discard subpartitions very quickly. This can be seen in figure 3.7, where CSGen is only needed to evaluate a few collaboration structures problem sets generated by UPD, before it manages to find the optimal solution. However, this is not necessarily the case for NPD and NDCS, as can be seen in figure 3.6 and 3.7.

In figure 3.7, we can see that CSGen don't have to evaluate many collaboration structures before it finds the optimal value. For 12 agents and 8 tasks, in the problem sets generated by UPD, CSGen only needs to evaluate about 2500 collaboration structures before it finds the optimal solution. This is remarkable, since there exists a total of $8^{12} = 68719476736 > 10^{10}$ collaboration structures in the problem sets with 12 agents and 8 tasks, and CSGen only has to evaluate about 0.00004% of the possible collaboration structures before it finds the optimal solution. This indicates that our strategies for pruning, expanding partitions, and refining subpartitions are outstanding, and that the solution manages to prune the search space efficiently. CSGen exhibits a similar efficiency in the problem sets generated by NPD. Finally, the problem sets generated by NDCS are much tougher for CSGen to solve, and it takes many more collaboration structure evaluations before it manages to reach utility values that are close to optimal.

In any case, it doesn't take many subpartitions or collaboration structure evaluations before CSGen finds close-to-optimal collaboration structures — for any of the three probability distributions. For NPD, the utility values of the collaboration structures that are evaluated in the first 0.02% subpartitions are often within a bound of being 12.5% from optimal. For NDCS, CSGen finds a solution that is within a bound of 10% from optimal after having only evaluated 10% of the possible subpartitions. For UPD, it takes a few more subpartitions before CSGen has found a close-to-optimal solution. However, solutions that CSGen generates for UPD are often within a bound of 20% of being optimal after having evaluated roughly 10% of the subpartitions.

These results show that CSGen doesn't necessarily have to search for very long before it can find a good solution. This is beneficial for many real-time systems, including real-time strategy games in which optimal solutions are not always required (e.g. for army to region assignment in EU4, "good" solutions are often "good enough"). They also indicate that the much quicker anytime solutions that CSGen generates are sufficient for many purposes (e.g. real-time strategy games).

Experiments on the order of precedence for partition expansions

As can be seen in figure 3.8 and figure 3.9, the order of precedence did affect the performance of CSGen. However, when the number of tasks was increased, the relative difference in performance became much smaller.

When using the first order of precedence, i.e. the order based on the numbers of zeroes in a partition, NDCS generated problem sets for which, on average, it was better to use a random order of precedence. This is not surprising, even though we hypothesized that this approach would improve search times when there were many zeroes. The reason is that, for NDCS, this is not the case, since coalitions with 0 agents always have a utility value (or performance measure) of 0. Therefore, and by sorting on the number of zeroes, we will expand many unnecessary low-value partitions.

It's worth noting that our second order of precedence, i.e. the order based on the upper and lower bounds of each partition, had stable results, even for the problem sets generated by

NDCS. This is not surprising, since even in cases where there are partitions with many zeroes, the second approach would search promising partitions first.

The most interesting aspect of these experiments was perhaps the fact that they indicate that different problems can potentially benefit from different precedence orders. If this is the case, then it would be interesting to use machine learning to improve the order of precedence for any domain that CSGen is applied to.

Validity of the experiments

The results of the experiments does not showcase and specific oddities. This is mainly due to the number of times that each problem description was run. If we did fewer runs, then the results didn't converge, and artifacts appeared. We therefore deem that the validity of our experiments is high. However, there is always room for improvement. One such improvement would be to use confidence intervals (e.g. 95% confidence intervals) for every problem description and probability distribution. This would give a clearer indication to how big any error might be, and thus increase the validity of our tests. Also, this would also be able to give a clear indication to whether the null hypothesis is validated.

4.3 Pay-off distribution and collaboration structure stability

In practice, when the optimal (or best) collaboration structure has been found, all that is left is to actually assign the agents to their new coalitions. In this phase, it's possible to add pay-off distributions to the collaboration formation scheme, where any agent that partake in a collaboration structure is given a reward that tells the system that the agent is less interested in joining a new coalition. Recall that a specific task could, if the system wants it to, indicate that the agent shouldn't be assigned to a new coalition. This could potentially make the formed coalitions more stable, since rewards can be used to decrease the probability of the agent joining a new task-oriented coalition (i.e. not the task that indicates a null assignment). If the formed coalitions are unstable, then new coalitions may keep on forming without anything actually being accomplished. Rewards or pay-offs can be used to counteract such behaviour.

4.4 Resumability

Finally, we didn't include any experiments on the resumability of our algorithm in our results. The reason is that we deem that its obvious that the design of our algorithm supports such behaviour, and that we have already sufficiently demonstrated that this is the case. The presented algorithm can easily store a search state, and resume from any search state when required. This is also possible when the world state has changed, since it builds its own internal model of the world by using coalition values, partitions, and bounds.

Chapter 5

Conclusion

In this chapter, we conclude this thesis. We first give a few final words on the problems that we have addressed, discuss whether we have managed to reach our goal, and conclude whether we have answered the research questions that we presented in the introduction chapter. We then discuss future work aimed at improving collaboration formation, and possible improvements to the presented algorithm.

5.1 Final words

The problem of forming groups that are aimed at solving specific tasks is a central problem for collaboration and cooperation in multi-agent systems. In this thesis, we addressed and discussed several issues that are inherent to this problem. First, we introduced the problem of simultaneous coalition formation and task assignment by looking at real-time multi-agent systems. We presented the domain of real-time strategy games, which is a domain of multi-agent systems in which there hasn't been much research on algorithms for cooperation and collaboration. Second, we presented theory for the collaboration formation procedure, and an algorithm that efficiently solves the problem of collaboration formation by using branch-and-bound. Third, we presented results that gave a clear indication to that our algorithm is a viable solution to the simultaneous coalition formation and assignment problem in real-time systems with few agents and tasks. The results also established that our algorithm is superior to an algorithm that is currently being used in a commercial real-time strategy game (i.e. Europa Universalis 4), and that it is an efficient approach to forming optimal collaboration structures, even when subject to harder problem sets (e.g. NDCS). We also demonstrated that it has several important properties for real-time systems, including anytime, optimality, and the ability to prune the search space. Apart from these properties, our algorithm is also resumable, and able to give worst-case guarantees.

With these considerations in mind, we have managed to answer all of the research questions from chapter 1, as presented below:

1. Can multi-agent task allocation be integrated into the formation of coalitions?
Answer: Yes. This was accomplished by creating an order of precedence for the mapping of agents to tasks, and then using branch-and-bound to efficiently search for optimal coalition to task assignments (i.e. collaboration structures).
2. Can a collaboration formation algorithm be applied to real-time strategy games in order to improve the utility and capabilities of artificial agents?
Answer: Yes. We accomplished this by applying our algorithm to a collaboration formation problem that is inherent to real-time strategy games, i.e. the assignment of armies to regions, and showing that our algorithm improved the utility of coalitions.
3. What are the limitations of algorithms that solve the simultaneous coalition formation and assignment problem in the domain of real-time systems?
Answer: From our results, and the discussions on time complexity in the theory chapter,

it's obvious that our algorithm cannot handle too large problem sets. However, the problem sets generated by Europa Universalis 4 are all manageable. The main bounding factor — in terms of performance — is $|T|2^{|A|}f_{max}$, where f_{max} denotes the maximum computational complexity of the function that generates coalition values. This is due to the fact that if this factor is too high, the presented algorithm cannot provide an anytime solution quick enough. Other algorithms could potentially be used in systems when the bounding factor is too large. Also, our algorithm could potentially be adapted to handle such instances approximately by subdividing the search space **before** computing the coalition values, and then calculating partial results subsequently to give approximate solutions.

The type of problem sets that the algorithm needs to solve has a huge impact on the performance, as demonstrated by the results of the tests that were created using probability distributions. As such, the distribution of the coalition values may have a substantial effect on the limits of our algorithm.

Apart from answering our research questions, it is also clear that we have managed to reach our goal, which was to improve and increase the knowledge and understanding of algorithms that are designed to improve agent collaboration. By designing a novel algorithm for collaboration formation, we also managed to solve the simultaneous coalition formation and assignment problem. However, perfectly solving the problem of collaboration and cooperation may require advanced reasoning — but this thesis, and designing an algorithm that can be applied to real-time multi-agent systems (e.g. real-time strategy games), is a step forward in the right direction.

5.2 Future work

To finalize this thesis, we would like to discuss a few approaches that could potentially improve the presented algorithm:

- Subpartition refinements increased the performance by a significant margin in partitions where there are many collaboration structures with tasks that have zero assignments. It would be interesting to deduce whether it is possible to generalize this technique to tasks with any number of assignments. A relevant question would be: how much performance would we gain if we kept sorting the partitions on the cardinal numbers of each subpartition?
- A similar potential improvement would be to assign agents to the tasks that correspond to the smallest partition numbers first. This would make it possible to calculate tighter bounds quicker. However, this would probably be inefficient when solving unbiased problem sets (e.g. NDCS), but could perhaps increase the performance in biased problem sets (e.g. real-world multi-agent systems, EU4, UPD, and NPD).
- Discarding subspaces is important to make the presented algorithm viable for collaboration formation. It would be interesting to evaluate the performance gains and quality losses when making the algorithm remove subspaces that only contain solutions that are of a certain factor better than the one previously found. For instance, instead of checking if the upper bound of a partition is lower than the current solution, we can check if the upper bound multiplied by a given factor ≤ 1.0 is lower than the current solution. By doing this we would be able to discard many more subpartitions, while still giving worst-case guarantees on the solutions.
- It could be interesting to use the resumability of the presented algorithm to generate solutions that are returned to the system a few simulation updates after the algorithm has started its search. This could potentially work well with distributed computing, where solutions are generated using external computers or servers.

- Using machine learning to find and dictate the order of precedence for the expansion of partitions could potentially improve the performance of the algorithm.

Apart from improvements to the presented algorithm, there are numerous other approaches that could potentially be used to form collaboration structures:

- It would be interesting to study the effects of using parallel computing to improve performance. We did not make any such attempts during this study, since agents in Europa Universalis 4 are already doing their computations on separate threads (and the threads are already very busy!).
- Other partitioning schemes could affect performance drastically. As such, studying other partitioning schemes could be of interest in future studies. One could for example study dynamic partitioning schemes, where partitions are based on the characteristics of the problem set that the algorithm is trying to solve.
- Attempts to use other search techniques instead of branch-and-bound would also be of interest. Deep learning has recently shown great success in many different domains, and could perhaps be used to create a model that can guide a Monte Carlo-based collaboration formation algorithm. It is also a reasonable suggestion to use dynamic programming to form collaboration structures.

Bibliography

- [1] Activision. *Fourth Quarter and CY 2010 Results*. 2011. URL: <http://investor.activision.com/common/download/download.cfm?companyid=ACTI&fileid=440263&filekey=2a37de98-400f-4916-9bb3-ae5ddf1b86b8&filename=ATVI%5C%20C4Q10%5C%20slides%5C%20FINAL.pdf> (visited on 03/28/2017).
- [2] David W Aha, Matthew Molineaux, and Marc Ponsen. “Learning to win: Case-based plan selection in a real-time strategy game”. In: *International Conference on Case-Based Reasoning*. Springer. 2005, pp. 5–20.
- [3] American Go Association. *A Brief History of Go*. 2017. URL: <http://www.usgo.org/brief-history-go> (visited on 03/18/2017).
- [4] Priyadarshan Banjan and Albert Silver. *Komodo: Birth of a chess engine*. 2016. URL: <https://en.chessbase.com/post/komodo-birth-of-a-chess-engine> (visited on 04/18/2017).
- [5] Blizzard. *StarCraft II World Championship Series*. 2017. URL: <https://wcs.starcraft2.com/en-us/> (visited on 04/04/2017).
- [6] Michael Buro. *2009 ORTS RTS Game AI Competition*. 2009. URL: <https://skatgame.net/mburo/orts/AIIDE09/> (visited on 04/04/2017).
- [7] Michael Buro. *AIIDE Starcraft AI Competition*. 2016. URL: <http://www.cs.mun.ca/~dchurchill/starcraftaicomp/> (visited on 04/23/2017).
- [8] Michael Buro. *Open RTS Homepage*. 2017. URL: <https://skatgame.net/mburo/orts/> (visited on 05/20/2017).
- [9] Michael Buro. “Real-time strategy games: A new AI research challenge”. In: *IJCAI*. 2003, pp. 1534–1535.
- [10] Michael Buro and David Churchill. “Real-time strategy game competitions”. In: *AI Magazine* 33.3 (2012), p. 106.
- [11] Michael Buro and Timothy M Furtak. “RTS games and real-time AI research”. In: *Proceedings of the Behavior Representation in Modeling and Simulation Conference (BRIMS)*. Vol. 6370. 2004.
- [12] BWAPI. *The Brood War Application Programming Interface*. 2017. URL: <https://bwapi.github.io/> (visited on 03/20/2017).
- [13] Arnoud Visser (moderated by). *RoboCup Rescue Simulation League*. 2017. URL: http://wiki.robocup.org/Rescue_Simulation_League (visited on 05/20/2017).
- [14] *C++ Standard Library Reference*. 2017. URL: <http://en.cppreference.com/w/cpp> (visited on 06/02/2017).
- [15] Pedro Cadena and Leonardo Garrido. “Fuzzy case-based reasoning for managing strategic and tactical reasoning in StarCraft”. In: *Mexican International Conference on Artificial Intelligence*. Springer. 2011, pp. 113–124.
- [16] Murray Campbell, A Joseph Hoane, and Feng-hsiung Hsu. “Deep blue”. In: *Artificial intelligence* 134.1-2 (2002), pp. 57–83.

-
- [17] Vladimír Čern. “Thermodynamical approach to the traveling salesman problem: An efficient simulation algorithm”. In: *Journal of optimization theory and applications* 45.1 (1985), pp. 41–51.
- [18] Paul C Chu and John E Beasley. “A genetic algorithm for the generalised assignment problem”. In: *Computers & Operations Research* 24.1 (1997), pp. 17–23.
- [19] Michael Chung, Michael Buro, and Jonathan Schaeffer. “Monte Carlo Planning in RTS Games.” In: *CIG*. Citeseer. 2005.
- [20] David Churchill. *AIIDE Starcraft AI Competition*. 2017. URL: <http://www.cs.mun.ca/~dchurchill/starcraftaicomp/> (visited on 03/20/2017).
- [21] Xiao Cui and Hao Shi. “A*-based pathfinding in modern computer games”. In: *International Journal of Computer Science and Network Security* 11.1 (2011), pp. 125–130.
- [22] Viet Dung Dang, Rajdeep K Dash, Alex Rogers, and Nicholas R Jennings. “Overlapping coalition formation for efficient data fusion in multi-sensor networks”. In: *AAAI*. Vol. 6. 2006, pp. 635–640.
- [23] Google DeepMind. *AlphaGo*. 2016. URL: <https://deepmind.com/research/alphago/> (visited on 04/02/2017).
- [24] Oriol Vinyals (DeepMind). *DeepMind and Blizzard to release StarCraft II as an AI research environment*. 2016. URL: <https://deepmind.com/blog/deepmind-and-blizzard-release-starcraft-ii-ai-research-environment/> (visited on 03/07/2017).
- [25] Stockfish Developers. *Stockfish Source Code*. 2017. URL: <https://github.com/official-stockfish> (visited on 04/02/2017).
- [26] Entertainment Software Association (ESA). “Essential facts about the computer and video game industry”. In: *Sales, demographic and usage data* (2016).
- [27] *Europa Universalis 4 Bookmarks and Scenarios*. 2017. URL: <http://www.eu4wiki.com/Scenarios> (visited on 06/02/2017).
- [28] Dan Fu and Ryan Houlette. “The ultimate guide to FSMs in games”. In: *AI game programming Wisdom 2* (2004), pp. 283–302.
- [29] William A Gamson. “A theory of coalition formation”. In: *American sociological review* (1961), pp. 373–382.
- [30] Brian P Gerkey and Maja J Matarić. “A formal analysis and taxonomy of task allocation in multi-robot systems”. In: *The International Journal of Robotics Research* 23.9 (2004), pp. 939–954.
- [31] Johan Hagelbäck and Stefan J Johansson. “Using multi-agent potential fields in real-time strategy games”. In: *Proceedings of the 7th international joint conference on Autonomous agents and multiagent systems-Volume 2*. International Foundation for Autonomous Agents and Multiagent Systems. 2008, pp. 631–638.
- [32] Daniel Damir Harabor, Alban Grastien, et al. “Improving Jump Point Search.” In: *ICAPS*. 2014.
- [33] Bryan Horling and Victor Lesser. “A survey of multi-agent organizational paradigms”. In: *The Knowledge Engineering Review* 19.4 (2004), pp. 281–316.
- [34] Robert Houdart. *Houdini Chess Engine Website*. 2017. URL: <http://www.cruxis.com/chess/houdini.htm> (visited on 04/03/2017).
- [35] Feng-hsiung Hsu. “IBM’s deep blue chess grandmaster chips”. In: *IEEE Micro* 19.2 (1999), pp. 70–81.

-
- [36] Masaki Hyodo, Tokuro Matsuo, and Takayuki Ito. “An optimal coalition formation among buyer agents based on a genetic algorithm”. In: *International Conference on Industrial, Engineering and Other Applications of Applied Intelligent Systems*. Springer. 2003, pp. 759–767.
- [37] IBM. *IBM100 - Deep Blue*. 2017. URL: <http://www-03.ibm.com/ibm/history/ibm100/us/en/icons/deepblue/> (visited on 05/04/2017).
- [38] Intel. *Intel Extreme Masters*. 2017. URL: <http://en.intelxtrememasters.com/> (visited on 04/04/2017).
- [39] Paradox Interactive. *About Europa Universalis 4*. 2017. URL: <http://www.europauniversalis4.com/about> (visited on 03/20/2017).
- [40] Paradox Interactive. *Paradox Interactive Announces Grand Successes for Grand Strategy Titles*. 2016. URL: <https://www.paradoxplaza.com/news/Grand-success/> (visited on 03/20/2017).
- [41] Antony Iorio and Xiaodong Li. “A cooperative coevolutionary multiobjective algorithm using non-dominated sorting”. In: *Genetic and Evolutionary Computation—GECCO 2004*. Springer. 2004, pp. 537–548.
- [42] Nicholas R Jennings. “Controlling cooperative problem solving in industrial multi-agent systems using joint intentions”. In: *Artificial intelligence* 75.2 (1995), pp. 195–240.
- [43] Steven Ketchpel. “Forming coalitions in the face of uncertain rewards”. In: *AAAI*. Vol. 94. Citeseer. 1994, pp. 414–419.
- [44] Matthias Klusch and Andreas Gerber. “Dynamic coalition formation among rational agents”. In: *IEEE Intelligent Systems* 17.3 (2002), pp. 42–47.
- [45] Matthias Klusch and Onn Shehory. “A polynomial kernel-oriented coalition algorithm for rational information agents”. In: *Tokoro*, ed (1996), pp. 157–164.
- [46] Sarit Kraus, Onn Shehory, and Gilad Taase. “Coalition formation with uncertain heterogeneous information”. In: *Proceedings of the second international joint conference on Autonomous agents and multiagent systems*. ACM. 2003, pp. 1–8.
- [47] Harold W Kuhn. “The Hungarian method for the assignment problem”. In: *Naval research logistics quarterly* 2.1-2 (1955), pp. 83–97.
- [48] Kate S Larson and Tuomas W Sandholm. “Anytime coalition structure generation: an average case study”. In: *Journal of Experimental & Theoretical Artificial Intelligence* 12.1 (2000), pp. 23–42.
- [49] Eugene L Lawler. “The quadratic assignment problem”. In: *Management science* 9.4 (1963), pp. 586–599.
- [50] Yann LeCun, Yoshua Bengio, and Geoffrey Hinton. “Deep learning”. In: *Nature* 521.7553 (2015), pp. 436–444.
- [51] Hui-Yi Liu and Jin-Feng Chen. “Multi-robot cooperation coalition formation based on genetic algorithm”. In: *Machine Learning and Cybernetics, 2006 International Conference on*. IEEE. 2006, pp. 85–88.
- [52] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Andrei A Rusu, Joel Veness, Marc G Bellemare, Alex Graves, Martin Riedmiller, Andreas K Fidjeland, Georg Ostrovski, et al. “Human-level control through deep reinforcement learning”. In: *Nature* 518.7540 (2015), pp. 529–533.
- [53] Monty Newborn. *Kasparov versus Deep Blue: Computer chess comes of age*. Springer Science & Business Media, 2012.

-
- [54] Timothy J Norman, Alun Preece, Stuart Chalmers, Nicholas R Jennings, Michael Luck, Viet D Dang, Thuc D Nguyen, Vikas Deora, Jianhua Shao, W Alex Gray, et al. “Agent-based formation of virtual organisations”. In: *Knowledge-based systems* 17.2 (2004), pp. 103–111.
- [55] Santi Ontanón, Kinshuk Mishra, Neha Sugandh, and Ashwin Ram. “ON-LINE CASE-BASED PLANNING”. In: *Computational Intelligence* 26.1 (2010), pp. 84–119.
- [56] Santiago Ontañón, Kinshuk Mishra, Neha Sugandh, and Ashwin Ram. “Case-based planning and execution for real-time strategy games”. In: *International Conference on Case-Based Reasoning*. Springer. 2007, pp. 164–178.
- [57] Santiago Ontanón, Gabriel Synnaeve, Alberto Uriarte, Florian Richoux, David Churchill, and Mike Preuss. “A survey of real-time strategy game ai research and competition in starcraft”. In: *IEEE Transactions on Computational Intelligence and AI in games* 5.4 (2013), pp. 293–311.
- [58] David W Pentico. “Assignment problems: A golden anniversary survey”. In: *European Journal of Operational Research* 176.2 (2007), pp. 774–793.
- [59] Marc Ponsen, Héctor Munoz-Avila, Pieter Spronck, and David W Aha. “Automatically generating game tactics through evolutionary learning”. In: *AI Magazine* 27.3 (2006), p. 75.
- [60] Talal Rahwan. “Algorithms for coalition formation in multi-agent systems”. PhD thesis. University of Southampton, 2007.
- [61] Talal Rahwan and Nicholas R Jennings. “An algorithm for distributing coalitional value calculations among cooperating agents”. In: *Artificial Intelligence* 171.8-9 (2007), pp. 535–567.
- [62] Talal Rahwan and Nicholas R Jennings. “An improved dynamic programming algorithm for coalition structure generation”. In: *Proceedings of the 7th international joint conference on Autonomous agents and multiagent systems-Volume 3*. International Foundation for Autonomous Agents and Multiagent Systems. 2008, pp. 1417–1420.
- [63] Talal Rahwan, Tomasz P Michalak, Michael Wooldridge, and Nicholas R Jennings. “Coalition structure generation: A survey”. In: *Artificial Intelligence* 229 (2015), pp. 139–174.
- [64] Talal Rahwan, Sarvapali D Ramchurn, Nicholas R Jennings, and Andrea Giovannucci. “An anytime algorithm for optimal coalition structure generation”. In: *Journal of Artificial Intelligence Research* 34 (2009), pp. 521–567.
- [65] Stuart Russell, Peter Norvig, and Artificial Intelligence. *A Modern Approach*. 3rd ed. Pearson Education Limited, 2016.
- [66] Ayed Salman, Imtiaz Ahmad, and Sabah Al-Madani. “Particle swarm optimization for task assignment problem”. In: *Microprocessors and Microsystems* 26.8 (2002), pp. 363–371.
- [67] Tuomas W Sandholm and Victor RT Lesser. “Coalitions among computationally bounded agents”. In: *Artificial intelligence* 94.1-2 (1997), pp. 99–137.
- [68] Tuomas W Sandholm and Victor R Lesser. “Coalition formation among bounded rational agents”. In: *IJCAI (1)*. 1995, pp. 662–671.
- [69] Tuomas Sandholm, Kate Larson, Martin Andersson, Onn Shehory, and Fernando Tohmé. “Anytime coalition structure generation with worst case guarantees”. In: *arXiv preprint cs/9810005* (1998).
- [70] Travis C Service and Julie A Adams. “Coalition formation for task allocation: Theory and algorithms”. In: *Autonomous Agents and Multi-Agent Systems* 22.2 (2011), pp. 245–248.

- [71] Onn Shehory and Sarit Kraus. “Coalition formation among autonomous agents: Strategies and complexity (preliminary report)”. In: *European Workshop on Modelling Autonomous Agents in a Multi-Agent World*. Springer. 1993, pp. 55–72.
- [72] Onn Shehory and Sarit Kraus. “Methods for task allocation via agent coalition formation”. In: *Artificial intelligence* 101.1-2 (1998), pp. 165–200.
- [73] David Silver, Aja Huang, Chris J Maddison, Arthur Guez, Laurent Sifre, George Van Den Driessche, Julian Schrittwieser, Ioannis Antonoglou, Veda Panneershelvam, Marc Lanctot, et al. “Mastering the game of Go with deep neural networks and tree search”. In: *Nature* 529.7587 (2016), pp. 484–489.
- [74] Gabriel Synnaeve and Pierre Bessiere. “A bayesian model for opening prediction in rts games with application to starcraft”. In: *Computational Intelligence and Games (CIG), 2011 IEEE Conference on*. IEEE. 2011, pp. 281–288.
- [75] Gabriel Synnaeve, Nantas Nardelli, Alex Auvolat, Soumith Chintala, Timothée Lacroix, Zeming Lin, Florian Richoux, and Nicolas Usunier. “TorchCraft: a Library for Machine Learning Research on Real-Time Strategy Games”. In: *arXiv preprint arXiv:1611.00625* (2016).
- [76] Tadao Takaoka. “An $O(1)$ time algorithm for generating multiset permutations”. In: *International Symposium on Algorithms and Computation*. Springer. 1999, pp. 237–246.
- [77] IEEE CIG StarCraft AIC Team. *IEEE CIG StarCraft AI Competition - Sejong University*. 2017. URL: https://cilab.sejong.ac.kr/sc_competition/ (visited on 04/24/2017).
- [78] SSCAIT AI Tournament Team. *StarCraft AI Tournament Homepage*. 2017. URL: <http://sscaitournament.com/> (visited on 04/24/2017).
- [79] The Wargus Team. *Wargus Home*. 2017. URL: <http://wargus.sourceforge.net/index.shtml> (visited on 05/20/2017).
- [80] John Tromp and Gunnar Farneback. “Combinatorics of go”. In: *International Conference on Computers and Games*. Springer. 2006, pp. 84–99.
- [81] Maksim Tsvetovat and Katia Sycara. “Customer coalitions in the electronic marketplace”. In: *Proceedings of the fourth international conference on Autonomous agents*. ACM. 2000, pp. 263–264.
- [82] Nicolas Usunier, Gabriel Synnaeve, Zeming Lin, and Soumith Chintala. “Episodic Exploration for Deep Deterministic Policies: An Application to StarCraft Micromanagement Tasks”. In: *arXiv preprint arXiv:1609.02993* (2016).
- [83] Wiebe Van der Hoek and Michael Wooldridge. “Multi-agent systems”. In: *Foundations of Artificial Intelligence* 3 (2008), pp. 887–928.
- [84] Ben George Weber and Michael Mateas. “Case-Based Reasoning for Build Order in Real-Time Strategy Games.” In: *AIIDE*. 2009.
- [85] Gerhard Weiss. *Multiagent systems: a modern approach to distributed artificial intelligence*. MIT press, 1999.
- [86] Hyeongon Wi, Seungjin Oh, Jungtae Mun, and Mooyoung Jung. “A team formation model based on knowledge and collaboration”. In: *Expert Systems with Applications* 36.5 (2009), pp. 9121–9134.
- [87] Aaron Williams. “Loopless generation of multiset permutations using a constant number of variables by prefix shifts”. In: *Proceedings of the twentieth Annual ACM-SIAM Symposium on Discrete Algorithms*. Society for Industrial and Applied Mathematics. 2009, pp. 987–996.

- [88] Takeo Yamada and Yasushi Nasu. “Heuristic and exact algorithms for the simultaneous assignment problem”. In: *European Journal of Operational Research* 123.3 (2000), pp. 531–542.
- [89] Jingan Yang and Zhenghu Luo. “Coalition formation mechanism in multi-agent systems based on genetic algorithms”. In: *Applied Soft Computing* 7.2 (2007), pp. 561–568.