Cranfield University

Pau Seguí-Gascó

Decentralised Multi-Robot Task Allocation Algorithms

Institute of Aerospace Sciences
School of Aerospace Transport and Manufacturing

2017

Supervisors:
Dr. Hyo-Sang Shin
Prof. Antonios Tsourdos

Submitted in partial fulfilment of the requirements for the degree of Doctor of Philosophy.

A l'alegria de ma vida.
Dic Isabel i segueisc el camí.

Abstract

Multi Robot Systems (MRS) are gaining increasing popularity both in the research community and in industry. A fundamental problem that underpins the effective coordinated operation of these systems is Task Allocation. To solve this problem, the MRS should be able to find an answer quickly, reliably, and effectively to the question: "given the robots available in the system and the tasks that ought to be carried out, what is the best allocation of these tasks among us?". In this thesis we focus on solving this problem in the decentralised setting, that is, when each agent only has access to its own utility function and does not have any knowledge of the functions corresponding to other agents, i.e. the utility function is *local* or *private*.

Our algorithms are based on improved versions of the measured continuous greedy algorithm for general matroid-constrained submodular maximisation. The first improvement is a new and smoother increment rule that enables us to reduce the number of steps required to solve the relaxation. The second improvement is to adapt the Decreasing-Threshold procedure for monotone submodular functions to work with non-monotone submodular functions.

Then, we present the first decentralised algorithm with constant-factor approximation guarantees for general submodular task allocation. Our algorithm provides an approximation factor of $1 - \frac{1}{e} - 4\epsilon$ ($\approx 63\%$) for monotone submodular utilities, and a factor of $\left(\frac{1}{e} - 3\epsilon\right)$ ($\approx 37\%$) for non-monotone submodular functions. To illustrate the possibilities enabled by non-monotone submodular task allocation, we present a submodular task allocation model for a multi-UAV surveillance mission. Our model features the allocation of heterogeneous surveillance tasks to a heterogeneous multi-UAV team under risk of enemy detection. We develop the model and present proofs to show that it is non-monotone submodular. Then, we run numerical experiments to study the effect of different parameters of our algorithm and compare its performance against the state-of-the-art.

To conclude the thesis, we take a completely different approach, the key idea is to trade constant-factor approximation guarantees in exchange for flexibility. We present a preliminary framework based on combinatorial auctions that can transfer centralised solution method to the decentralised Task Allocation domain while requiring a polynomial number of communication rounds. In other words, our framework provides a way to transfer successful methods to solve NP-Hard problems such as Metaheuristics, Mixed-Integer Programming, Constraint Programming, etc. to the decentralised setting.

Keywords: Algorithms, Combinatorial Optimization, Submodularity, Task Allocation, Decentralised Task Allocation, Distributed Task Allocation, Discrete Optimization, Submodular Maximization, Matroid, Non-Monotone Submodular.

Acknowledgements

I would like express my gratitude to my supervisors, Sang and Antonios, for their guidance and support during all these years, and for their generosity during difficult times. Sang has been a great mentor, he has given me confidence when forces faltered, encouraged me in my research, and allowed me to grow as a researcher.

Thanks to all the ImSim family: Robin, Didac, Carl, Vittoria, David, Shampa, Gayathri, Gary, Raymond, Dan, Leo, and Stephen, for their generosity and patience during this last year. I would like to specially thank Robin and Didac for their insightful comments and constant encouragement, they were a great support that helped me cross the line. And yes Carl, it is finished!

All these years would not have been the same without all the amazing people I met. Thanks to my fellow PhD students Dario, Quintain, Sang-Jun, and Inmo, for the good company and all the laughter. Thanks to all the Cranfielders: Vicente, Mirian, Octavio, Viktoria, Yazan, Tonia... for the great times we had inside this little countryside island.

I would like to thank my family, my father Jesus, my mother Ana and my sister Mireia. I will be forever indebted with them for all their love, support and encouragement throughout my life, they have always been there when I most needed. I have been very fortunate that my father enthused me with his love and passion for engineering and who, with his infinite patience, spent countless hours sharing his wisdom. I would not be what I am today without them. I know I don't say this very often, in fact I never do, but I'd like to say that I love you all very much.

Last but not least, I am hugely grateful to my wife Isabel, the joy of my life. Her immense love, encouragement, and quiet patience were the bedrock upon which the past years of my life have been built. Without her generosity and unwavering support I would not have been able to finish this thesis. *"Tendrem la mida de totes les coses només en dir-mos que ens seguim amant."*

The support of the National PhD Programme of the Defence Science and Technology Laboratory (UK MoD) is gratefully acknowledged.

Contents

List of Figures

List of Tables

Nomenclature

$\mathcal{A}$	Set of Agents.
CBBA	Consensus-Based Bundle Algorithm.
CGA	Continuous Greedy Algorithm.
$\underline{d}$	Minimum marginal value.
ΔF_i	Marginal value of the Multilinear Extension of f, wrt element i.
ΔF_{aj}	Marginal value of the Multilinear Extension of f_a, wrt to task j.
$\bar{d}$	Maximum marginal value.
E	Ground set.
F	Multilinear Extension of f, $F : [0,1]^E \to \mathbb{R}^+$
f	Utility function $f : 2^E \to \mathbb{R}^+$.
F_a	Multilinear Extension of f_a, $F : [0,1]^E \to \mathbb{R}^+$
f_a	Utility function of agent $a \in \mathcal{A}$ $f : 2^E \to \mathbb{R}^+$.
f_S	Marginal value function wrt the set S, $f_S(i) \triangleq f(S \cup \{i\} - f(S)$.
GA	Greedy Algorithms.
$\mathcal{I}$	Independence set of a matroid.
$\mathcal{K}$	The relaxation polytope of the task allocation problem.
$\mathcal{M}$	Matroid.
MRS	Multi-Robot Systems.
OPT	Optimal discrete solution to a problem.
$\mathcal{P}(\mathcal{M})$	The relaxation polytope of the matroid $\mathcal{M}$.
r	Rank of a matroid.
$\mathcal{S}^*$	Discrete solution of an algorithm $\mathcal{S}^* \subseteq \mathcal{A} \times \mathcal{T}$.
$\mathcal{T}$	Set of Tasks.
UAV	Unmanned Aerial Vehicle.
$\mathbf{y}$	A point in the relaxation.
$\mathbf{y}_a$	A fractional allocation relaxation corresponding to agent $a \in \mathcal{A}$.

Chapter 1

Introduction

1.1 Background and Motivation

1.1.1 Multi-Robot Systems

Multi-robot systems (MRS) are gaining increasing popularity as an alternative to single asset solutions due to its versatility, resilience and its distributed nature [7, 82]. This is a paradigm shift from few but very capable, expensive, and complex systems towards systems of relatively low-tech hardware coupled with sophisticated cooperation. A good example of this trend is the *new space* sector [3]. Traditionally, space companies built constellations of a handful of large and complex satellites weighting hundreds of kilograms, costing hundreds of millions each and with decades-long service lives. Nowadays companies such as Planet Labs [103] are starting to build viable business models based on constellations of numerous but smaller and simpler satellites. Their Flock-1 satellite constellation is composed of tens of cubesat-based satellites weighting less than 10kg apiece. These satellites are launched in groups, which bring down the cost of each satellite to the tens of thousands, and are designed for service lives of less than a year or two. The large number of cooperating satellites enables Planet Labs to have updated footage of virtually every point in the surface of the Earth almost weekly, whilst they can cope with the loss of one or multiple satellites because their cost is a fraction of that of a traditional system.

Another area where this trend is gaining traction is Defence. Current operational Unmanned Aerial Vehicles (UAVs), such as the Global Hawk, are very sophisticated and provide a lot of capability but the loss of such a key strategic asset could put an entire mission in jeopardy. In contrast, new multi-robot system models of operations are emerging as a complementary solution. In October 2016 the Strategic Capabilities Office of the US Department of Defense carried out a real flight test [2] where they launched, from fighter planes, one hundred small *Perdix* drones executing a cooperative mission. The coordination of the Perdix drone system is entirely based on decentralised communication between the drones themselves without a central entity. This makes the whole system resilient to the loss of several team members, and provides capabilities that simply would be unavailable with a single-asset solution.

Multi-robot systems have some fundamental strengths that are not possible with single-robot systems, let us mention a few of them: **increased flexibility**,

provided by a variety of payloads available in different team members; **simultaneous broad area coverage**, enabled by different robots in different spatial locations at the same time; **reliability and resilience**, inherent due to the large number of low-cost robots which can take over each other in case of fault; and the capability to **operate outside the communications range** of their base stations, as team members can also act as communication relays for each other.

However, for their successful operation, a key enabling technology is autonomous cooperation. Consider the Global Hawk UAV, a currently operational state-of-the-art system, it requires both a crew of at least four and continuous data links back to its command centre. This is because pilots, sensor operators, communications officers, etc. need to micromanage every aspect of its operation. This is clearly not scalable for a team of tens or hundreds of vehicles that are cooperating in the same mission. Therefore, a key step towards fully operational multi-robot systems is the transition from remote operators that micromanage the robots towards network operators who can inject tasks into a multi-robot network and supervise their execution. The effective autonomous assignment of the available resources is the key enabler of successful cooperation. The robots should be able to find an answer quickly, reliably, and effectively to the question: "given the resources available in the network and the tasks that ought to be carried out, what is the best allocation of these tasks among us?". This is known as the task allocation problem, and it is central to the coordination of maritime multi-robot systems [57], satellite constellations [95], ground robot teams [24], and multi-UAV missions [61]. There are several features of task allocation problem that is worthwhile remarking. We are dealing, primarily, with heterogeneous multi-robot systems, that is, the robots can have different mobility characteristics (fixed-wing vs rotary, underwater vs surface vessels, etc.). Robots that can carry different payloads or sensors (visual, radar, signal intelligence, etc.). Robots that have different information available to them, depending on what they have received from neighbours or sensed from the environment. Robots that are geographically distributed. Robots that are in challenging communication environments where they may be able to link with some of their neighbours, but not with their base, or the whole team. In summary, a multi-robot team can be very diverse, have different levels of information, and have different communication links. These characteristics are what make a multi-robot system very capable but, at the same time, they make the task allocation problem very difficult to solve. More formally, the general task allocation problem can be defined as follows:

Given a set of tasks $\mathcal{T}$, a set of agents $\mathcal{A}$, and a function for each agent $a \in \mathcal{A}$ specifying the utility of completing each subset of tasks $f_a : 2^{\mathcal{T}} \to \mathbb{R}^+$, find a non-overlapping allocation, $\mathcal{S}^* \in \mathcal{A}^{\mathcal{T}}$, that minimises/maximises a global objective function $\mathcal{J} : \mathcal{A}^{\mathcal{T}} \to \mathbb{R}^+$. (Adapted from the classical definition in [27])

The centralised solution of the task allocation problem involves having to communicate all the agents (i.e. robots) and environment data to a centralised entity. This may not be possible in some realistic scenarios because relying on a central entity removes resilience (by introducing single point of failure), or the bandwidth to communicate all the information to it may not be available. Increasingly, new hybrid control architectures are being proposed. Where a hi-

erarchical architecture for high level control (such as task creation) is combined with a decentralised decision making architecture for lower level control (task execution). Therefore, decentralised task allocation algorithms might become essential building blocks to enable these new architectural ideas. In light of this, we focus on developing algorithms that work on the scenario where the agents are not required to have access to a central planning entity, or a common understanding of the environment, and are only required to have access to their own individual utility functions, not that of their peers. This is known as the *decentralised* task allocation problem, and it is the focus of this thesis.

1.1.2 Task Allocation and Submodularity

The general task allocation problem is very hard to solve efficiently: it is both NP-Hard [35] and inapproximable within a constant factor [78]. This implies that it is very unlikely that efficient algorithms that, for the general case, find an optimal solution or a good approximate solution will ever be found. Faced with this bleak outlook, most of the decentralised algorithms presented to date have taken two qualitatively different approaches: either they target a trivial instance by assuming linearity of the utility function, enabling optimal algorithms [8, 19, 42, 66, 73, 105]; or they settle for algorithms without any guaranteed performance that, nevertheless, show good empirical results for the tested objective functions [25–27, 36, 49, 79, 104]. A great breakthrough was the introduction of a new and distinct approach: an algorithm that used non-trivial properties on the objective function to enable decentralised constant-factor approximation algorithms. In [19] Choi et al presented the Consensus-Based Bundle Auction (CBBA) Algorithm, which was the first decentralised algorithm that provided a solution guaranteed to be within a constant factor of the optimal. By assuming that the utility functions of the agents were monotone non-decreasing and submodular (i.e. exhibiting diminishing marginal returns), Choi et al were able to prove that their algorithm produces a solution that guarantees at least 50% of the value of the optimal one: it does not find an optimal solution but it finds a provably-good one. There are deep theoretical reasons why they chose submodularity, and they are intimately connected with the tractability of the task allocation problem. Indeed, much like convexity in continuous optimisation, submodularity seems to be the crucial non-trivial property that enables us to achieve 'good' solutions.

Submodularity is quite an intuitive notion. It simply requires that the marginal value that an agent obtains by executing an extra task diminishes as the tasks that ought to be carried out by that agent increases. It is quite natural, consider for example a surveillance mission: as a given agent is assigned more points to monitor, the time that it will be able to monitor an additional point decreases. Another example is sensor placement, as we deploy more sensors the coverage provided by adding an additional sensor keeps decreasing, until, eventually, the whole area of interest is covered, providing no extra coverage. We remark that there may be scenarios of practical interest that do not seem to have an obviously submodular model. However, due to the good tractability properties that it induces (as we shall see), it is a good model to aim for when designing utility functions. Indeed, something similar happens with convexity: many problems in control, estimation, signal processing, ma-

chine learning, etc. are not obviously convex; but scientists and engineers have devised useful models to "convexify" the problem, in order to enable fast algorithms that result in immensely useful applications. A concrete, and obvious, example is the Extended Kalman Filter (EKF), which imposes severe assumptions (linearity, non-correlation, gaussian noise, etc.) in order to find an optimal estimator. These assumptions are often violated in practice but, nevertheless, it still delivers such great performance that it has become the standard tool to use in its field. We think submodularity should be seen from that perspective: a useful model that, while it may not be 100% faithful to the application at hand, it can be leveraged to induce the desired behaviour in a multi-robot system. In the Machine Learning community there has been a great effort spearheading this idea: finding suitable submodular models to solve inherently discrete tasks, such as summarising documents, scene segmentation, or pattern discovery [10, 23, 45, 64, 72, 89, 94]. We think that this is an idea worth pursuing: task allocation is an inherently discrete problem (a task is either executed by an agent or not, it cannot be split), and so we should aim to build models that yield inherently good algorithms for discrete problems. That is, we think that "submodularising" interesting task allocation problems can yield immensely useful applications.

The task allocation problem can be formulated in terms of a more general problem: the optimisation of a set function subject to a matroid constraint. Understanding the tractability of this more general problem is key to appreciate the boundaries of what is possible to achieve with polynomial-time algorithms for the task allocation problem. Let us give a quick summary. The optimisation of a set function subject to a matroid constraint can take two forms: maximisation, and minimisation. In both cases, submodularity plays a key role defining the tractability boundaries. Indeed, unconstrained submodular minimisation is a tractable problem, and efficient algorithms have been developed [69]. However, somewhat counter-intuitively, the matroid-constrained problem is both NP-Hard and inapproximable [91]. On the other hand, when the problem takes the maximisation form the problem still remains NP-Hard [98], however, it becomes approximable if submodularity is assumed. In the late 1970s a seminal paper by Nemhauser, Wolsey, and Fisher [77] showed that the greedy algorithm achieves a $\frac{1}{2}$ approximation guarantee with monotone submodular functions. Sure enough, this fact lies at the core of why CBBA gives a constant factor approximation because, essentially, CBBA is a decentralised implementation of the greedy algorithm. In light of this, two questions arise: is $\frac{1}{2}$ the best we can do for monotone submodular functions?; and, what about non-monotone submodular functions?. Nemhauser et al [77] also proved that the best possible factor that one can achieve for matroid-constrained monotone submodular maximisation is $1 - \frac{1}{e}$, and in [15, 96] Vondrak et al provided an algorithm to achieve it. More recently, Feldman et al [32] presented a unifying algorithm: the *measured continuous greedy algorithm*. This algorithm obtains a $1 - \frac{1}{e}$ approximation for matroid-constrained monotone submodular maximisation and, more importantly, a $\frac{1}{e}$ factor for matroid-constrained non-monotone submodular maximisation. However, it is known that the $\frac{1}{e}$ factor is not optimal for the non-monotone case [30, 96]. Finding the optimal factor is an important open problem, currently all that is known is that it must be less than 0.478 [37]. To summarise: it is possible to devise task allocation algorithms that achieve a 63% (i.e., $1 - \frac{1}{e}$) approximation for the task allocation problem maximising

monotone submodular utilities, as well as a 37% (i.e., $\frac{1}{e}$) approximation for the non-monotone case. In contrast, it is not possible to find any polynomial-time constant approximation factor algorithm for the the task allocation problem minimising submodular utilities.

In view of this, we can see that there is a gap in the decentralised task allocation allocation literature between the algorithms that have been proposed, and what is possible to achieve from a theoretical perspective. In particular, the current state-of-the art in decentralised task allocation, CBBA [19], can only solve task allocation problems with monotone submodular utilities with a 50% approximation, while it provides no guarantees for non-monotone submodular utilities.

We believe that developing an algorithm that works with non-monotone submodular utilities is very important, because non-monotonicity is a feature that arises naturally in many practical scenarios. For example, in a multi-robot surveillance mission, if a robot is assigned too many targets to track it is possible that it ends up spending its time travelling between targets and not gathering enough useful information at the targets' locations. Therefore, adding tasks to a robot's assignment could, indeed, reduce the utility obtained. Another example is with a multi-robot team executing complicated manipulation functions, say in rough terrain, where completing each task is risky because the robot may suffer some difficulty, such as getting stuck, and thus fail the mission. In this situation, a single robot may well be the best suited to carry out all the required tasks, but a solution where an individual robot carries all the tasks would be undesirable because of the high risk of failure that it would involve. Monotone submodular functions are structurally ill-suited to model such scenarios, because, by definition, they do not contemplate a reduction in value due to an excessively large number of tasks. This ubiquitous situation cannot be modelled by monotone utility functions, and therefore, CBBA could perform arbitrarily poorly.

However, the continuous greedy algorithms that achieve the aforementioned approximation factors were devised, initially, as a theoretical tools to prove the existence of polynomial-time approximations, and were not geared towards efficiency. Essentially they solve a relaxation based on the multilinear extension, a continuous extension of submodular functions, and subsequently they round the solution. This resulted in impractically slow algorithms: Vondrak's algorithm [15] required $\Theta(n^8)$ value oracle calls, while Feldman's Measured Continuous Greedy [32] requires $O(n^6)$ assuming oracle access to the multilinear extension, which, in general, needs to be sampled, creating additional overhead, possibly in the order of $O(n^3)$ or $O(n^2)$. To remedy this, in [6] Badanidiyuru and Vondrak proposed an efficient $\left(1 - \frac{1}{e} - \epsilon\right)$-approximation algorithm that uses $O\left(\frac{nr}{\epsilon^4} \log^2 \frac{n}{\epsilon}\right)$ value oracle calls, for non-negative monotone submodular function maximisation subject to a matroid constraint. (Here n is the cardinality of the ground set and r is the matroid rank) They achieve this by using a Decreasing-Threshold procedure that enables them to reduce both the number of steps and the number of samples needed at each step of the continuous greedy process. Nevertheless, their algorithm does not apply to non-monotone submodular functions.

1.2 Aims and Objectives

In the last twenty years many great results in the Sciences and Engineering have been produced by following a *convexification* recipe: given an important problem, find a convex formulation, and apply efficient convex optimisation algorithms. We believe the analogous is possible in the Task Allocation domain: formulate important problems in terms of submodular utility models, and use guaranteed-approximation algorithms to solve them. Our vision is that the greatest leap in the Task Allocation field will come by *submodularising* utility models. This thesis aims to enable this vision by providing approximation algorithms for monotone and non-monotone submodular utility functions. In other words, in this thesis we bring decentralised task allocation approximation algorithms to the edge of what is currently known to be possible. That is, we develop the first decentralised task allocation algorithm that achieves a constant factor approximation with non-monotone submodular utilities, obtaining a $\frac{1}{e}$ ($\approx 37\%$) approximation guarantee. In addition, we improve the ratio for monotone submodular utilities up to the maximum that is possible to achieve: $1 - \frac{1}{e}$ ($\approx 63\%$). Finally, we present a preliminary framework that enables the decentralisation of a broad range of centralised algorithms.

Therefore, the objectives set up to achieve the aim of this study are as follows:

- Design a faster continuous greedy algorithm for matroid-constrained submodular maximisation, while keeping the same approximation factors.
- Perform theoretical analysis of our continuous greedy algorithm focusing on two aspects: approximation factor and computational complexity.
- Decentralise the aforementioned continuous greedy algorithm to solve the task allocation problem in a setting where agents can only access their own utility functions.
- Investigate properties of the proposed decentralised task allocation algorithm.
- Develop a realistic submodular model for a multi-UAV mission.
- Validate the task allocation algorithm developed using the submodular model developed.
- Present a preliminary framework to enable the use of centralised algorithms using polynomial-time communication.

1.3 Contributions and Layout

We start in **Chapter 2 where we present a Literature Survey** of the Task Allocation field. In the first part we present an a summary of the key features of the Task Allocation problem, and review key tractability results for the Task Allocation problem and for matroid-constrained submodular optimisation. To conclude, in the second part we present a review of decentralised Task Allocation algorithms.

In **Chapter 3 we present two improvements to make continuous greedy algorithms for general matroid-constrained submodular maximisation more practical**. Continuous greedy algorithms solve a relaxation of the problem by building the solution is small increments. The first improvement that we present is a new and smoother increment rule that enables us to use to use order of magnitude larger increments, reducing the number of steps required to solve the relaxation. The second improvement is to adapt the Decreasing-Threshold procedure of Badanidiyuru and Vondrak [6] to work with non-monotone submodular functions. In combination, these two modifications enable us to obtain achieve a $\left(\frac{1}{e} - 2\epsilon\right)$ approximation factor in the solution of the relaxation for matroid-constrained maximisation non-monotone submodular functions in $O(\frac{nr^2}{\epsilon^4}\left(\frac{\bar{d}+\underline{d}}{\bar{d}}\right)^2 \log^2(\frac{n}{\epsilon}))$ calls to the value oracle (where n is the cardinality of the ground set, r the rank of the matroid, and $\bar{d}$ and $\underline{d}$ bounds on the absolute values of the marginal values of the objective function).

Then, in **Chapter 4 we present the first decentralised algorithm with constant-factor approximation guarantees for submodular task allocation**. The algorithms that we presented in chapter 3 are developed for maximising a general non-negative submodular function subject to a matroid constraint. In this chapter, we take these algorithms and adapt them to solve the Submodular Task Allocation problem, defined as follows:

Given a set of tasks $\mathcal{T}$, a set of agents $\mathcal{A}$, and a non-negative submodular function for each agent $a \in \mathcal{A}$ specifying the utility of completing each subset of tasks $f_a : 2^{\mathcal{T}} \to \mathbb{R}^+$, find a non-overlapping allocation, $\mathcal{S}^* \in \mathcal{A}^{\mathcal{T}}$, that maximises a global objective function $\mathcal{F} : \mathcal{A}^{\mathcal{T}} \to \mathbb{R}^+$ defined as $\mathcal{F}(S) = \sum_{a \in \mathcal{A}} f_a(S_a)$. (adapted from [27])

Our algorithms solve this problem in the decentralised setting, that is, when each agent only has access to its own utility function and does not have any knowledge of the functions corresponding to other agents. In that sense, we refer to the utility functions of each agent as being *local* or *private*. Our algorithms achieve an approximation factor of $1 - \frac{1}{e} - 4\epsilon$ (≈ 63 %) for monotone submodular utilities, and a factor of $\left(\frac{1}{e} - 3\epsilon\right)$ for non-monotone submodular functions. This is the first decentralised algorithm that achieves a constant-factor approximation for task allocation with non-monotone submodular utilities, and it improves the current state-of-the-art for monotone utilities, CBBA [19], which achieves a $\frac{1}{2}$ (50%) factor. With this, we lay the theoretical bedrock upon which researchers and practitioners can build general non-negative submodular utility models for task allocation in a decentralised application.

In **Chapter 5 we present a submodular task allocation model for a multi-UAV surveillance mission**. Here, we lay out a model that features the allocation of heterogeneous surveillance tasks to a heterogeneous multi-UAV team under risk of enemy detection. We develop the model and present proofs to show that it is non-monotone submodular. Then, we run numerical experiments to study the effect of different parameters of our algorithm and compare its performance against the state-of-the-art. These experiments show that our algorithm's performance is superior to that of the current state-of-the-art CBBA [19], which is unsurprising because CBBA offers no guarantees for non-monotone submodular functions.

Finally, we take a completely different approach, the key idea is to trade constant-factor approximation guarantees in exchange for flexibility. There has been a huge amount of work in the Operations Research literature that enables the solution of NP-Hard problems very efficiently. For example, Vehicle Routing Problems with hundreds of tasks are solved daily by delivery businesses [93], even when the simplest instances (such as the Traveling Salesman Provlem) are NP-Hard. Recognising this, **in Chapter 6 we present a framework based on combinatorial auctions that can transfer almost any centralised solution method to the decentralised Task Allocation domain**. In other words, this framework provides a way to transfer successful methods to solve NP-Hard problems such as Metaheuristics, Mixed-Integer Programming, Constraint Programming, etc. to the decentralised setting. To illustrate our framework, we present preliminary results of numerical experiments with a multi-robot routing application using the commercial MIP Solver Gurobi [39].

1.4 Publication List

This thesis contains partially or in full material from the following publications by the author:

- *UAV Swarms: Decision-Making Paradigms.* HS Shin and P Segui-Gasco. Chapter in Encyclopaedia of Aerospace Engineering. John Wiley & Sons 2014.
- *A combinatorial auction framework for decentralised task allocation.* P Segui-Gasco, HS Shin, A Tsourdos, and VJ Segui. Wi-UAV: Globecom 2014 workshop - Wireless networking and control for unmanned autonomous vehicles; pages 1445-1450.
- *Decentralised Submodular Multi-Robot Task Allocation.* P Segui-Gasco, HS Shin, A Tsourdos, and VJ Segu. In 2015 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS); pages 2829-2834.
- *Fast Non-Monotone Submodular Maximisation Subject to a Matroid Constraint.* P Segui-Gasco and HS Shin. arXiv preprint arXiv:1703.06053.

Chapter 2

Literature Survey

In this chapter we provide a formal definition of the task allocation problem, followed by a review on its tractability, and a survey of the different decentralised methods that have been proposed to solve it.

2.1 The Task Allocation Problem

2.1.1 Problem Definition

Let us start by providing a formal definition of the general task allocation problem:

Given a set of tasks $\mathcal{T}$, a set of agents $\mathcal{A}$, and a set function for each agent $a \in \mathcal{A}$ specifying the utility of completing each subset of tasks $f_a : 2^{\mathcal{T}} \to \mathbb{R}^+$, find a non-overlapping allocation, $\mathcal{S}^* \in \mathcal{A}^{\mathcal{T}}$, that optimises a global objective function $\mathcal{J} : \mathcal{A}^{\mathcal{T}} \to \mathbb{R}^+$. (Adapted from [27])

Taxonomy

In recent years there has been a wealth of literature published studying different but closely related applications. The issue is that each application looks at its own particularisation of the problem with little or no reference to a more general framework or its relations with other works that might be reduced to the same abstract problem. To resolve this situation, a very important contribution to the field was made in the form a domain independent taxonomy to classify, interpret, and abstract the different versions of task allocation problems. In 2004 Gerkey and Mataric published what is now the standard taxonomy of the field [35], which provided a unifying theory to the task allocation family of problems, mapping instances of the task allocation problem to corresponding combinatorial optimisation problems. This taxonomy proposes three axes to characterise an instance of the task allocation problem:

- **Single-Task (ST) vs Multi-Task (MT)**.
  Distinguishing whether the robots are able to execute a single or multiple tasks at the same time.

- **Single-Robot (SR) vs Multi-Robot (MR)**.
  Distinguishing whether the tasks require one or multiple robots to be executed.

- **Instantaneous Assignment (IA) vs Time-Extended Assignment (IA)**.
  Distinguishing whether the robots construct a plan to be executed imminently or they can construct a more elaborate plan to be executed over a given horizon of time.

A particular instance of the task allocation problem is, therefore, classified by the triple ({ST or MT},{SR or MR},{IA or TA}), for a detailed explanation of each instance the reader is referred to [35]. This taxonomy by Gerkey and Mataric [35] provided a common reference frame to describe task allocation problems, however, there was a fundamental limitation: it did not explicitly cover dependencies between the tasks. Recently this gap was filled by the work of Korsah, Stentz and Dias [52]. Based on the taxonomy by Gerkey and Mataric, Korsah et al [52] provided the theoretical framework to extend the taxonomy to cover situations with task dependencies such as related utilities and task-coupling constraints. This provided a mapping of instances of the task allocation problem with dependencies to well studied combinatorial optimisation problems. The types of dependencies considered by this taxonomy are:

- **No Dependencies (ND).** Simplest case. Occurs when all the tasks are fully decoupled and, hence, the agent's utilities of the tasks are independent.

- **In-Schedule Dependencies (ID).** The assessments of an agent depend on what other tasks are being executed by that same agent. These dependencies could be in the utility function or constrains within the tasks schedule for each agent. Consequently its valuations are independent of the allocations of other agents.

- **Cross-Schedule Dependencies (XD).** Occurs when the tasks valuations depend not only of the executing agent's schedule but also on the other agents scheduled tasks. This can happen, for example, when multiple vehicles are needed to execute a given set of tasks, when temporal or precedence constraint are imposed in the tasks etc. Nevertheless these dependencies are simple, in the sense that they are known to the agents before the actual allocation.

- **Complex Dependencies (CD).** Occurs in the same case as cross-schedule dependencies but with the added compilation that these dependencies do not have a simple structure. They have a complex structure in the sense that the tasks that are being allocated have multiple decompositions into subtasks that are coupled on the allocation. Hence the dependencies can only be resolved simultaneously with the allocation, this coupling of the task decomposition problem and the task allocation problems create a more complicated set of dependencies than Cross-Schedule.

With this classification of the dependencies the problem instances are defined by combining a dependency type with an instance of the triple ({ST or

MT},{SR or MR},{IA or TA}). This expands the type of situations that can be modelled significantly, for a detailed discussion of each of the cases the reader is referred to [52]. The most studied cases involve either no dependencies or in-schedule dependencies, with some works devoted to cross-schedule and complex dependencies.

2.1.2 Tractability

General Formulation

To explain the tractability of the task allocation we need to cast it in terms of a general and well studied combinatorial optimisation problem: the optimisation of a set function subject to a matroid constraint. The task allocation problem optimises a set function subject to the constraint that no two agents can be allocated the same task, this constraint can be reformulated as a partition matroid. Matroids are an incredibly powerful abstraction of independence. They capture seemingly disconnected notions such as linear independence, forests in graphs, or traversals, among many others. They provide a flexible framework to characterise a variety of relevant constraints, such as: partitions, schedules, cardinality, or even rigidity. Of interest to us is a particular class of matroid, known as a partition matroid, which captures the non-overlapping allocation constraint in a task allocation problem. Let us state a formal definition of a matroid [85]:
A pair $(E, \mathcal{I})$ is called a matroid if E is a finite set and $\mathcal{I}$ is a nonempty collection of subsets of E satisfying:

- $\emptyset \in \mathcal{I}$*;*
- *if $A \in \mathcal{I}$ and $B \subseteq A$, then $B \in \mathcal{I}$; and*
- *if $A, B \in \mathcal{I}$ and $|A| < |B|$, then $A + z \in \mathcal{I}$ for some $z \in B \setminus A$.*

Let us now define a partition matroid, which captures the task allocation problem. A partition matroid is defined as follows: *Given the set E, and a partition of E into l disjoint sets $E = E_1 \cup E_2 \cup ... \cup E_l$, the set of independent sets $\mathcal{I}$ is defined as : $\mathcal{I} \triangleq \{S \subseteq E | 1 \geq |X \cap E_i|, \forall i = 1, ..., l\}$.* One can easily verify that this definition indeed satisfies the conditions of a matroid definition above. Let us now show how can it capture the non-overlapping allocation constraint. First we define the ground set $E = \mathcal{A} \times \mathcal{T}$, that is we have one 'copy' of each task for each agent, i.e. $(a, j) \in E$ for all $a \in \mathcal{A}$ and $j \in \mathcal{T}$. Then, we define the partition as follows: for each task $j \in \mathcal{T}$ we define a set $A_j = \{(a, i) \in E | i = j\}$ that contains all task-agent pairs that contain the task j, that is, $A_j = \{(a, j) | \forall a \in \mathcal{A}\}$. Therefore, we have disjoint sets that satisfy $E = \mathcal{A}_1 \cup \mathcal{A}_2 \cup ... \cup \mathcal{A}_l$. Then, the set of feasible allocations (i.e. independent sets) is $\mathcal{I} \triangleq \{S \subseteq E | 1 \geq |S \cap A_j|, \forall j \in \mathcal{T}\}$, that is an allocation S is feasible if it at most contains one task-agent pair per task. Now that we have defined the non-overlapping allocation constraint as a matroid, we can readily formulate the task allocation problem as:

$$\underset{S \in \mathcal{I}}{\text{optimise}}\ \mathcal{J}(S). \tag{2.1}$$

This is equivalent to the definition that we gave in the previous section, because $\mathcal{J} : \mathcal{A}^{\mathcal{T}} \to \mathbb{R}^+$ is the same function, and the constraint that $S \in \mathcal{I}$ simply states

that the allocation must be non-overlapping, i.e. it must be an independent set in the partition matroid defined above. Let us now look at the tractability of the task allocation problem.

The general task allocation problem is NP-Hard and inapproximable [35]. However, there are subclasses of the objective function for which there exist polynomial-time algorithms that provide optimal solutions or constant-factor approximation solutions. The first simplification that one can think of is to assume that the objective function $\mathcal{J}$ is the sum of the objectives of the utility functions of each of the agents, that is: $\mathcal{J}(S) = \sum_{a \in \mathcal{A}} f_a(S_a)$, where S_a is the set of tasks allocated to agent a. This, however, does not take us very far because the problem becomes an instance of the Set Packing Problem which happens to be both NP-Hard and impossible to approximate within a constant factor [80]. We can go one step further, and assume that the agents' set functions are linear. In this case the problem becomes the optimisation of a modular (i.e. linear) function subject to a matroid constraint, which, famously, can be solved optimally with a simple greedy algorithm [85]. However, a linear sum of linear (i.e. modular) utility functions is quite a limiting assumption. Indeed, in the taxonomy above, it only allows for ST-SR problems. To find more interesting objective functions we need to introduce the notion of submodularity.

Submodularity

A submodular function is simply a set function that exhibits diminishing marginal returns. More formally, a set function $f : 2^E \rightarrow \mathbb{R}$ is submodular if for all $Y, X \subseteq E$ satisfying $X \subseteq Y$ and $x \in E \setminus Y$, then

$$f(X + x) - f(X) \geq f(Y + x) - f(Y). \tag{2.2}$$

An intuitive way of thinking about submodular functions is to think of them as somewhat discrete analogous to convex functions. Convexity plays a central role in continuous optimisation. In the words of R.T. Rockafellar [84] -one of the great mathematicians in the field-: *"In fact, the great watershed in optimization isn't between linearity and nonlinearity, but convexity and nonconvexity."* Borwein and Vanderwerff [11] put it in other words: *"In a computational setting, since the interior-point revolution in linear optimization, it is now more or less agreed that 'convex'='easy' and 'nonconvex'='hard' -both theoretically and computationally."*. We believe that convexity and submdodularity are analogous in the following sense: the presence or absence of submodularity appears to be the key nontrivial property that determines whether one can find 'good' algorithms for the optimisation of a set function. This analogy should not be taken literally, indeed, diminishing returns are, in some sense, the discrete analogous of concavity too. Therefore, we think is best to view submodularity as a frontier that delineates the qualitative complexity in combinatorial optimisation problems.

There are qualitative differences between maximising and minimising a submodular function. Let us look at both of them. The minimisation of a submodular function without constraints is a tractable problem [69], and efficient algorithms have been devised for it. However, when one introduces constraints the problem becomes not only NP-Hard, but also inapproximable. In [91] Svitkina and Fleischer show that cardinality-constrained submodular minimisation is not approximable within a polynomial factor of $\Omega(n)$. Furthermore, the minimisation of a submodular function subject to a cardinality constraint can be

reduced to a more general problem: the minimisation of a submodular function subject to a matroid constraint, which consequently, is also inapproximable. Unfortunately for us, this implies that the approximability of task allocation problem is not tractable.

In contrast, submodular maximisation problems are NP-Hard to solve exactly [98]. There are, however, very practical constant-factor approximation algorithms. To explain them we need to introduce the concept of *monotonicity* of a set function: a set function is said to be *monotone* if the value of a set does not decrease as more elements are added to it, i.e. $f(A) \leq f(B)$ if $A \subseteq B$. In the monotone case, a classic result by Nemhauser, Wolsey, and Fisher [77] shows that the Greedy Algorithm yields a $1 - \frac{1}{e}$ constant-factor approximation for a cardinality constraint, and a $\frac{1}{2}$ factor for a matroid constraint. More recently, a fruitful avenue of research was spurred by Vondrak [15, 96] by showing that solving a relaxation of the problem based on the multilinear extension using the Continuous Greedy Algorithm and then rounding the result yields good approximation algorithms. They present their result in the context of non-negative monotone submodular functions to yield a $(1 - \frac{1}{e})$-approximation. Shortly after, Feldman et al. [32] modified Vondrak's algorithm to develop the Measured Continuous Greedy Algorithm which supported both non-negative non-monotone and monotone submodular functions. Their algorithm was the first to achieve a $\frac{1}{e}$-approximation for general non-negative submodular functions subject to a matroid constraint. Both continuous greedy algorithms can find the aforementioned approximation bounds for solving relaxation problems subject to the more general constraint class of down-closed polytopes. Let us now look at what are the limits of approximation factors that can be achieved in polynomial time. For matroid constrained monotone submodular maximisation Feige [31] showed that improving over the $1 - \frac{1}{e}$ threshold is NP-Hard, and so the continuous greedy guarantees are tight. In the non-monotone case, [37] showed that no polynomial time algorithm can achieve an approximation better than 0.478. Closing the gap between $\frac{1}{e}$ and 0.478 remains an important open problem where recent advances have been made: Ene and Nguyen in [30] give a 0.372-approximation, while Feldman et al in [12] improve it to a 0.385-approximation. These improvements are relatively small, recall that $\frac{1}{e} \approx 0.368$, but show that there is room for future improvement. Both algorithms are based on the measured continuous greedy. All these results apply to the task allocation problem because, as we have seen, it can be reduced to the optimisation of a set function subject to a matroid constraint. The current state-of-the-art in decentralised task allocation only scratches the surface of these results. One of our contributions is to present decentralised task allocation approximation algorithms that match the $1 - \frac{1}{e}$ factor for the monotone case, and $\frac{1}{e}$ for the non-monotone case. Now we review the current literature in decentralised task allocation.

2.2 Decentralised Task Allocation Literature

Linear Utilities

The earliest decentralised task allocation algorithms were proposed as solutions to the Assignment Problem. This is a slightly different problem to the version

that we have presented above. In this instance we have n tasks to assign to n agents, that is: each agent must have a task and each task must have an agent. Therefore, there is no notion of an agent utility function, instead, there is a weight or value for each task and agent. This, in the taxonomy described above, is an SR-ST task allocation problem without dependencies. The assignment problem can be solved optimally in polynomial time, the classic approach is the Hungarian Algorithm [56]. (For the most detailed and current treatise on the centralised approaches to this problem, the reader is referred to the book by Burkard et al [14].) Due to the tractability of these problems, there have been a number of algorithms proposed that do guarantee optimal performance in a decentralised setting. The first distributed task allocation strategy was proposed by Bertsekas [8], where an auction algorithm was developed based on a shared memory model. However, the shared memory model required a topology of the networked system that is not always achievable in real scenarios. To address this issue Zavlanos et al [105] proposed an auction-based algorithm to handle a networked system in which agents interact with its neighbours, rather than having access to a shared database. More recently, [42] presented a decentralised version of the Hungarian method which improved the performance of the auction-based methods, such as those above. An extension of the assignment problem is the Generalised Assignment Problem, where the utilities are linear, and additionally each task consumes a 'resource' when executed by a robot, and each robot has a linear resource constraint which must be satisfied. The addition of the resource constraint makes the Generalised Assignment Problem NP-Hard [62], but admits constant-factor approximation algorithms [86]. Indeed, Luo et al in [67] proposed the first decentralised task allocation algorithm with constant factor approximation guarantees by leveraging decentralised auction which, in essence, implements a greedy algorithm.

When we remove the constraint on allocating a task for each agent, the problem becomes the optimisation of a modular (i.e. linear) function subject to a matroid constraint, which, as we have seen, is solved optimally using the greedy algorithm. Several works have implemented versions of the greedy algorithm in a decentralised setting. In [19] the Choi et al present the Consensus-Based Auction Algorithm CBAA, this algorithm uses the concept of maximum consensus to distribute a series of single task auctions across the network, which in effect, implement a distributed version of the greedy algorithm. Liu and Shell [66] present another approach to implement a decentralised greedy algorithm based on on local task swaps. While in [73] Moon et al present an application of a qualitatively similar algorithm for multi-UAV task allocation in a dynamic environment, including an account of its performance in real flights.

Submodular Utilities

As we have seen, submodularity is the key property that enables tractability. A very important breakthrough in the decentralised task allocation field was the Consensus-Based Bundle Auction Algorithm (CBBA), presented by Choi, Brunet, and How in [19] which, for the first time, used the notion of submodularity to provide a decentralised approximation algorithm. Their target instance is a maximisation with a monotone submodular function, a problem that, as we have seen in the previous section, the greedy algorithm can approximate to a factor of $\frac{1}{2}$ [77]. Their algorithm is based on a decentralised auction and has two

phases: bundle construction and conflict resolution (consensus). In the bundle construction phase, each agent creates a bundle by greedily adding tasks based on their marginal values, until there are no tasks left or its bids are inferior than the current highest bidder. Once each agent has built their own bundle of tasks, the conflict resolution (or consensus) phase starts. In this phase, agents exchange with each other the marginal values (bids) that they have for each task and the agent with the highest marginal value for each task is assigned to it, and the outbid agents drop their subsequent bids. CBBA has spurred a lot of interest, as of mid-2017 it has more than 250 citations, and is already cited in industrial patents such as [16] for practical use in the coordination of heterogeneous vehicle missions. This has instigated a significant amount of work to extend its capabilities further, we discuss some of the extensions in the following paragraph. However, almost all these extensions surrender the approximation guarantees. Nevertheless, we remark that CBBA targets a *monotone* submodular function, and that it could perform arbitrarily poorly with a *non-monotone* submodular function, and so far there is no decentralised constant-factor, approximation task-allocation algorithm that exploits the favourable tractability of non-monotone submodular functions.

As we shall see in the following section, the greedy algorithm has been extensively used for general utility functions despite its lack of guarantees. This is because empirically, the greedy algorithm behaves well with many practical objective functions. However, CBBA can have convergence problems when the functions are not submodular. To address this, Johnson et al [47, 48] present a task scoring scheme that uses warping functions so that the bids from non-submodular utilities appear 'as if they were submodular (sic)' in the consensus phase while they are handled as non-submodular in the agent's own domain, consequently allowing for improved synergies within the bundles, and improving convergence. This lifts the requirement for convergence of the submodularity condition but, albeit, surrendering all the performance guarantees. Recently there has been a significant amount of work trying to find submodular surrogates, that is: submodular functions that are used to drive the decisions of the algorithms to optimise non-submodular functions. A good example is [18], where submodular surrogates are used to model the value of information. We believe these two ideas to be complementary, and in light of the good behaviour of the greedy algorithm in practice, believe that a great avenue for future research lies in the use of submodular models to optimise non-submodular utilities.

Several authors have extended CBBA so as to perform better in dynamic environments. In [9] the authors adapt CBBA for a realistic scenario with obstacles and measurement noise. In [81] the authors introduce tasks with time windows by using a exponentially decaying reward function, handling changing communication networks, and fuel cost reward awareness. Ponda et al [83] propose a framework to handle stochastic environments through chance-constrained reward functions. Johnson et al [46] propose a new set of consensus rules for CBBA so that it allows local agreement within asynchronous networks. Casbeer et al [70] propose another approach to overcome these problems in dynamic environments by the introduction of local interaction rules to handle "pop up" tasks within local agents, speeding up the convergence. In another work, Das et al. [22] exploit the same principle to speed up convergence, by the accomplishment of two phases, first: the execution of bids within neighbour robots in parallel; and second: the performance of a consensus operation to resolve the

conflicts. This accelerates the convergence and allows to parallelise allocation and execution. In [20] a CBBA-based mechanism to allocate tasks involving two agents is presented. Later, the same group, Choi et al [102], introduced an extension to handle the following task dependencies: unilateral dependency, mutual dependency, mutual exclusion, and timing constraints; all involving possibly more than two agents. With the purpose of adequate human supervision of large UAV autonomous networks, Casbeer et al [5] extended CBBA with the notion of teams, each team allocates tasks independently using CBBA and then, through an outer loop, teams exchange unassigned tasks. In another work Hunt et al [41] introduced group-dependent tasks by modifying the score functions and the conflict resolution strategy of CBBA at the expense of higher communication overhead. In summary one can say that these CBBA extensions mostly do one of two things: extending the applicability of CBBA to more difficult environments (communication, dynamic, etc); or enabling CBBA to converge when solving a more general or more constrained problem, at the expense, however, of the approximation guarantees.

General Utilities

Now we focus on works that take a qualitatively different approach: rather than relying on assumptions on the objective function -like, e.g. CBBA with submodularity, to obtain provably-good solutions- these works rely on the decentralisation of metaheuristic approaches that show good performance in practice. This is perhaps the area of work that has produced the most publications in decentralised task allocation, and the field is very diverse, so here we review only a selection of them. Given the good theoretical guarantees that we have seen that the greedy algorithm provides, it is no surprise that it is at the centre of many heuristic approaches even if the guarantee-enabling assumptions are not satisfied. Indeed, in the early 2000s there was a push for mechanisms that used market-based mechanisms based on a common approach: find an initial allocation using some sort of greedy heuristic, and then improve the solution via dynamic re-allocations using different methods. Among these, the most prominent are: Alliance [79], Traderbots [25, 26], Murdoch [36], and Hoplites [49]. Which had huge success and laid the foundation for many of the algorithms that were proposed later.

Another interesting idea is to take a given objective function and to decentralise successful centralised algorithms. Routing is one example, in [58] the authors use a approximation algorithms for the Traveling Salesman Problem based on Minimum Spanning Forests to provide decentralised routing for a team of robots with constant-factor approximation guarantees. And, in [75] this approach was extended to support limited communication capability. However, these are approaches, by their very nature, are ad-hoc for the problem. And we are not aware of any general framework that allows the decentralisation of a wide variety of centralised algorithms.

Finally, we would like to mention another compelling approach presented recently by Zhang et al in [107] and [106]. They proposed a series of Stohastic Clustering Auction algorithms (SCA) that are conceptually similar to the classic Simulated Annealing metaheuristic, because allow the designer to choose the rate at which the stochastic exploration of the solution space takes place. As with Simulated Annealing, a global optimal solution can be obtained if the

cooling rate (in SA terms), or the rate of the proportion of stochastic steps in the search is reduced, is slow enough. However, we need to keep in mind that the problem is NP-Hard, and hence if global optimal results are desired, the cooling rate will have to be very slow, and an exponential convergence time should be expected. Nevertheless, the notion of what a satisfactory solution is depends on each specific problem and, consequently, allowing direct control on the speed-optimality trade off, empowers the designer with the tools to adjust the convergence speed to his/her specific notion of what a satisfactory solution is in each situation.

The main conclusion that we can extract for the general problem is that there are a good number of decentralised algorithms but, in essence, these implement rather simple heuristics. There are, however, centralised tools in the Operations Research literature that solve instances of practical interest very well. For example, Vehicle Routing problems with hundreds of tasks are solved daily by delivery businesses [93]. This is because there are a host of techniques, e.g. Metaheuristics, Mixed Integer Programming, or Constraint Programming, that over many decades of algorithmic research and engineering have been adapted and tailored for specific problems, reaching the point where they can solve large instances efficiently. There is not, nevertheless, a framework that enables the use of these centralised methods directly in the decentralised task allocation domain.

2.3 Discussion

The main conclusion that we can extract from the tractability survey and decentralised task allocation survey is that there is a gap between what is theoretically possible, and what has been carried over to decentralised task allocation domain. In different forms and shapes, the backbone of most decentralised task allocation methods is the greedy algorithm. This is a direct consequence of its good theoretical guarantees: it is optimal with linear objective functions, and gives a $\frac{1}{2}$ approximation for the maximisation monotone submodular functions. However, these theoretical results on the greedy algorithm were first presented in the late 1970s, and the discrete optimisation field has progressed a lot since then. For us, there are two key relevant developments that motivate the contributions of this thesis. The first, and most important, is that there are polynomial-time algorithms to approximate the maximisation of a *non-monotone* submodular function subject to a matroid constraint within a factor (i.e. $\frac{1}{e}$). The second, is that the *monotone* case approximation factor can be improved from $\frac{1}{2}$ to $1 - \frac{1}{e}$. Therefore, in this thesis we aim to bridge this gap and bring the decentralised task allocation state of the art to the edge of what is known to be possible.

There is, however, a less positive corollary of our review of the tractability of the submodular task allocation problem: the minimisation version of a submodular task allocation problem is intractable and, worse, inapproximable.

Finally, we would like to point out another conclusion from our review of the decentralised task allocation literature: there is a myriad of methods that are tailored to specific instances that work well. But there is no general framework that given a good centralised algorithm that works well in practice can translate it into a decentralised algorithm.

Chapter 3

Continuous Greedy Algorithms for Submodular Maximization

3.1 Introduction

Our aim in this dissertation is obtaining useful ways to solve Task Allocation problems. We have seen that in general we cannot find the optimal solution efficiently, but that does not mean that we cannot find an approximate solution that is provably good. In other words, we do not know how to find the optimal solution, but we do know how to find something that is not far off. This kind of algorithms are known as constant-factor approximation algorithms because they produce a solution S such that, for a constant-factor $\alpha \leq 1$, $f(S) \geq \alpha f(OPT)$, where f is the objective function. The price to pay for this guarantees, in our case, is that we have to restrict the objective functions to be non-negative submodular. On the other hand, we can abstract the constraint that no task can be allocated to more than one agent to a more general one: a matroid. That is, the algorithms that we present in this chapter work not only for the Submodular Task Allocation problem, but rather for a more general problem class: the maximisation of a general non-negative submodular function subject to a matroid constraint. Moreover, they provide the theory underpinning the decentralised algorithms in the following chapter. Let us now explain more about submodularity and matroids.

Submodular maximisation problems have drawn a lot of attention recently [54]. This interest is due to a good confluence of theoretical results and practical applicability. Intuitively, a submodular set function is said to be so because it exhibits diminishing marginals returns, i.e.: the marginal value that an element adds to a set decreases as the size of the set increases. This simple property arises naturally in many applications and is what enables us to obtain constant-factor approximation algorithms. It has been used in a variety of application domains, to name but a few: markets [29, 60], influence in networks [51], document summarisation [23, 64, 94], and sensor placement [55, 63].

Matroids are an incredibly powerful abstraction of independence. They

capture seemingly disconnected notions such as linear independence, forests in graphs, or traversals, among many others. Interestingly, with linear sum objectives (modular functions), they are inextricably linked with the greedy algorithm. If the greedy algorithm is optimal, then there is an implicit matroid; if there is a matroid, then the greedy algorithm is optimal [85]. They provide a flexible framework to characterise a variety of relevant constraints, such as: partitions, schedules, cardinality, or even rigidity. Indeed, a particular class of matroid, known as partition matroid, captures the constraints in a Task Allocation problem that no one task is allocated to more than one agent.

Importantly, the combination of a submodular function and a matroid constraint, not only captures the Submodular Task Allocation problem (also known as Submodular Welfare), it actually plays a unifying role for many well-known combinatorial optimisation problems, such as: Max k-Cover, Max Generalised Assignment, Max Facility Location, and Constrained Max Cut (e.g. Max Bisection) among many others.

However, maximising a submodular function subject to a matroid constraint is NP-Hard. Hence much of the research effort has focused on obtaining good approximation algorithms. A classic result by Nemhauser, Wolsey, and Fisher [77] shows that the Greedy Algorithm is a $\frac{1}{2}$-approximation for non-negative monotone submodular functions. More recently, a fruitful avenue of research was spurred by Vondrak [15, 96] by showing that solving a relaxation of the problem based on the multilinear extension using the Continuous Greedy Algorithm and then rounding the result yields good approximation algorithms. They present their result in the context of non-negative monotone submodular functions to yield a $(1 - \frac{1}{e})$-approximation. Shortly after, Feldman et al. [32] modified Vondrak's algorithm to develop the Measured Continuous Greedy Algorithm which supported both non-negative non-monotone and monotone submodular functions. Their algorithm was the first to achieve a $\frac{1}{e}$-approximation for general non-negative submodular functions subject to a matroid constraint. Both continuous greedy algorithms can find the aforementioned approximation bounds for solving relaxation problems subject to the more general constraint class of down-closed polytopes. An important breakthrough in this field are Contention Resolution Schemes, a rounding framework proposed in [99], because they provide a paradigm for developing rounding schemes for a combination of useful constraints including matroids and knapsacks. Thus the combination of continuous greedy relaxations and Contention Resolution schemes enabled approximation algorithms for many important submodular maximisation problems.

However, continuous greedy algorithms were devised initially as a tool to show the existence of polynomial-time approximations, and thus were not geared towards efficiency. This resulted in impractically slow algorithms: Vondrak's algorithm required $\Theta(n^8)$ [15] value oracle calls, while Feldman's Measured Continuous Greedy requires $O(n^6)$ [32] assuming oracle access to the multilinear extension, which needs to be sampled in general, creating additional overhead, possibly more than around $O(n^3)$ or $O(n^2)$.

To remedy this, in [6] Badanidiyuru and Vondrak proposed an efficient $\left(1 - \frac{1}{e} - \epsilon\right)$-approximation algorithm that uses $O\left(\frac{nr}{\epsilon^4} \log^2 \frac{n}{\epsilon}\right)$ value oracle calls, for non-negative monotone submodular function maximisation subject to a matroid constraint. They achieve this by using a Decreasing-Threshold procedure that enables them to reduce both the number of steps and the number of samples

needed at each step of the continuous greedy process.

In the inapproximability front, for matroid constrained non-negative monotone submodular maximisation Feige [31] showed that improving over the $1-\frac{1}{e}$ threshold is NP-Hard, and so the continuous greedy guarantees are tight. In the non-monotone case, [37] showed that no polynomial time algorithm can achieve an approximation better than 0.478. Closing the gap between $\frac{1}{e}$ and 0.478 remains an important open problem where recent advances have been made: Ene and Nguyen in [30] give a 0.372-approximation, while Feldman et al in [12] improve it to a 0.385-approximation. This improvements are relatively small, recall that $\frac{1}{e} \approx 0.368$, but show that there is room for future improvement. Both algorithms are based on the measured continuous greedy, and would therefore benefit from the techniques presented here.

Contribution

In this chapter we present a smoother version of the measured continuous greedy algorithm of [32] with the same approximation ratio, i.e. $\frac{1}{e}$ for non-monotone and $1-\frac{1}{e}$ for monotone, while using orders of magnitude fewer function evaluations than the current state of the art method described in [32]. Then, we take this algorithm and accelerate it even further to create the first practical algorithm for the maximisation of a general non-negative submodular function subject to a matroid constraint.

In section 3.2 we present a smoother version of the measured continuous greedy is based on a more continuous-like integration rule than that of [32], which allows us to use an orders of magnitude larger step-size. This makes the proposed method faster because the number of value oracle calls to f scales with the number of steps, hence the bigger the step-size δ, the fewer function evaluations that need to be performed. The critical observation for developing the proposed method is that, around a small region, the subset $S_{\mathbf{v}^*}$ (i.e., the subset that maximises the marginal value) does not change. Hence, if $S_{\mathbf{v}^*}$ is given, we can find an analytical solution to the target differential equation to use as the update step of the integration method. Therefore, the error of the proposed scheme depends on whether δ is small enough to capture the changes in $S_{\mathbf{v}^*}$. This, in practice, allows us to use orders of magnitude larger δ to obtain the same accuracy as the Euler-like method of Feldman [32]. An improvement that can be consistently observed for a variety of different submodular functions, as we show in the numerical experiments section. Indeed, in contrast with the algorithm in [32] which required $O(n^6)$ in the multilinear oracle i.e. $O(n^7)$ or $O(n^8)$ in the value oracle, our smooth measured continuous greedy requires $O\left(\frac{r^4}{\epsilon^3} n \log(n) \left(\frac{\bar{d}+\underline{d}}{\bar{d}}\right)^3\right)$ value oracle calls, and $O\left(\frac{r^2 n}{\epsilon}\left(\frac{\bar{d}+\underline{d}}{\bar{d}}\right)\right)$ independence oracle calls. That is we obtain a speed-up of at least $O(n^2)$. Note that $\underline{d}, \bar{d} \in \mathbb{R}^+$ are bounds on the absolute value of the minimum and maximum marginal values that the function f can take, i.e. $-\underline{d} \leq f(S+i) - f(S) \leq \bar{d}$, for all $i \in E$ and $S \subseteq E$.

But this can still be improved, in section 3.3 we present a $\frac{1}{e}-\epsilon$-approximation algorithm for the non-monotone case that reduces the number of value oracle calls by $O(r^2)$. We achieve this acceleration by combining the decreasing-threshold procedure of [6] with our smoother version of the measured continuous greedy. This enables us to obtain an algorithm that requires just

$O\left(\frac{nr^2}{\epsilon^4}\log^2(\frac{n}{\epsilon})\left(\frac{\bar{d}+d}{\bar{d}}\right)^2\right)$ value oracle calls. To our knowledge, this is the first to achieve a practical efficiency for matroid-constrained general non-negative submodular function maximisation. Our algorithm is slower than the one proposed by Badanidiyuru and Vondrak [6] by $O\left(\left(\frac{\bar{d}+d}{\bar{d}}\right)^2\right)$, but their algorithm only works for monotone submodular functions. Here we trade-off some computational time in exchange of approximation guarantees for non-monotone submodular functions.

3.1.1 Preliminaries and the Continuous Greedy Process

Before we proceed to explain our algorithm we need to lay out some preliminary concepts that will be used in this chapter and throughout the thesis. First we present the definitions of a submodular function and a matroid in order to define formally the our target problem and model of computation. Then we describe in detail the relaxation that continuous greedy algorithms use. This involves the definition of the matroid polytope and of the multilinear extension. Later we define in more detail important aspects of the multilinear extension, which is central to continuous greedy algorithms. Subsequently we define the notions of marginal value and maximum marginal improvement which allow us to present formally the continuous greedy process.

Submodularity A function $f : 2^E \to \mathbb{R}^+$ on a set E is said to be submodular if:

$$f(A) + f(B) \geq f(A \cup B) + f(A \cap B), \text{ for all } A, B \subseteq E. \tag{3.1}$$

A more intuitive, but equivalent, definition can be formulated in terms of the marginal value added by an element: given $Y, X \subseteq E$ satisfying $X \subseteq Y$ and $x \in E \setminus Y$, then $f(X + x) - f(X) \geq f(Y + x) - f(Y)$. Herein we overload the symbol $+$ $(-)$ to use it as shorthand notation for the addition (subtraction) of an element to a set, i.e. $S + i = S \cup \{i\}$ $(S - i = S \setminus \{i\})$. We also use the common subscript notation to denote the discrete derivative a function, that is $f_S(x) \triangleq f(S + x) - f(S)$, this is also known as the marginal value.

Matroids A matroid can be defined as follows [85]:
A pair $(E, \mathcal{I})$ is called a matroid if E is a finite set and $\mathcal{I}$ is a nonempty collection of subsets of E satisfying:

- $\emptyset \in \mathcal{I}$;
- *if $A \in \mathcal{I}$ and $B \subseteq A$, then $B \in \mathcal{I}$; and*
- *if $A, B \in \mathcal{I}$ and $|A| < |B|$, then $A + z \in \mathcal{I}$ for some $z \in B \setminus A$.*

A matroid base $B \subseteq E$ is a maximally independent set $B \in \mathcal{I}$, that becomes dependent by adding an additional element $e \in E \setminus B$, i.e. $B + e \notin \mathcal{I}$. A key property of matroids is that we can exchange elements between bases, this is formalised as follows:

Lemma 3.1.1. *(Corollary 39.12A in [85]) Let $\mathcal{M} = (N, \mathcal{I})$ be a matroid, and $B_1, B_2 \in \mathcal{B}$ be two bases. Then there is a bijection $\phi : B_1 \to B_2$ such that for every $b \in B_1$ we have $B_1 - b + \phi(b) \in \mathcal{B}$.*

Problem Definition Now we can formally define our target problem:
Given a ground set E, a matroid defined upon it $\mathcal{M} = (E, \mathcal{I})$, and a general non-negative submodular function $f : 2^E \to \mathbb{R}^+$, find:

$$\max_{S \in \mathcal{I}} f(S). \tag{3.2}$$

Model of Computation As it is common in the submodular maximisation literature, here we assume that f and $\mathcal{I}$ are given as oracles, and we quantify the complexity of our algorithms in terms of number of calls to a *value oracle* and an *independence oracle* respectively.

Now we can proceed to describe in detail the relaxation that continuous greedy algorithms use. This has two parts: the domain, which is the matroid polytope; and the function used to evaluate fractional solutions, the multilinear extension. We describe both in detail and present relevant facts and lemmas that will be used later on.

Relaxation A common approach to solve combinatorial optimisation problems is by using relaxation and rounding. This involves solving a continuous problem, in which we allow fractional solutions, and a rounding procedure which transforms the fractional output back to a discrete solution. Now let us explain the relaxation that is used in continuous greedy algorithms. There are two key elements needed to define the relaxation: the domain that contains fractional solutions, and a function to evaluate fractional solutions. In our problem, the most natural representation for the domain is the matroid polytope. And the function that we use to evaluate fractional solutions is the multilinear extension. Both the matroid polytope $P(\mathcal{M})$ and the multilinear extension F have some nice properties worth noting let us present them now.

Matroid Polytope The matroid polytope $P(\mathcal{M})$ is the convex hull of the incidence vectors of the independent sets in the matroid, i.e.: $P(\mathcal{M}) = \text{Conv}(\{\mathbf{1}_S : \forall S \in \mathcal{I}\})$ [85]. Where $\mathbf{1}_S$ denotes the incidence vector of a set $S \subseteq E$, which contains a 1 for each element in S and 0 elsewhere, thus $\mathbf{1}_S \in [0,1]^E$. $P(\mathcal{M})$ is down-closed, which means that, given $\mathbf{v} \in P(\mathcal{M})$, if we reduce any coordinate of $\mathbf{v}$, $\mathbf{v}$ still remains inside $P(\mathcal{M})$[85]. Another advantage is that, in general, given a matroid polytope, linear programs on it with arbitrary real objective vectors have integral solutions [85]. Furthermore, they can be obtained using a simple greedy algorithm [85] provided independence oracle access to the matroid. This implies that, given any objective vector $\mathbf{c} \in \mathbb{R}^E$, the linear programming problem:

$$\max_{\mathbf{v} \in P(\mathcal{M})} \quad \mathbf{c}^\intercal \mathbf{v} \tag{3.3}$$

can be solved using a simple polynomial-time greedy algorithm. Fortunately, this means that we can solve quickly such linear programs even when $P(\mathcal{M})$ is defined by an exponential number of linear constraints, as it is sometimes the case.

The Multilinear Extension Now let us examine the properties of the multilinear extension. Given a set E and a submodular set function $f : 2^E \to \mathbb{R}^+$

defined on it, let $\mathbf{y} \in [0,1]^E$ represent a fractional solution, and let $R(\mathbf{y}) \subseteq E$ be a random set containing each element $i \in E$ with probability y_i. Then, the multilinear extension of f is:

$$F(\mathbf{y}) \triangleq \mathbb{E}[f(R(\mathbf{y}))] = \sum_{\mathcal{S} \subseteq E} f(\mathcal{S}) \prod_{i \in \mathcal{S}} y_i \prod_{j \notin \mathcal{S}} (1 - y_j). \tag{3.4}$$

The multilinear extension holds the following useful properties (mostly from [15]):

- F has second partial derivatives everywhere in its definition unit hypercube, i.e. $F \in C_2([0,1]^E)$
- F is multilinear, i.e., $\frac{\partial^2 F}{\partial y_i^2} = 0$,
  hence, $\frac{\partial F}{\partial y_i} = \frac{F(\mathbf{y}+\mathbf{1}_i \Delta y) - F(\mathbf{y})}{\Delta y}, \forall i \in E, \forall \Delta y \in (0, 1 - y_i]$.
- F has negative second partial derivatives $\frac{\partial^2 F}{\partial y_i \partial y_j} \leq 0$.
- $\frac{\partial F(\mathbf{y})}{\partial y_i} = \mathbb{E}[f(R(\mathbf{y}) + i) - f(R(\mathbf{y}) - i)] \leq f(i) \leq \bar{d}$. Due to submodularity.
- If f is monotone: $\frac{\partial F}{\partial y_i} \geq 0, \forall i \in E$.
- If f is non-monotone: $\frac{\partial F}{\partial y_i} = \mathbb{E}[f(R(\mathbf{y}) + i) - f(R(\mathbf{y}) - i)] \geq -\underline{d}, \forall i \in E$.
- If f is monotone: $\frac{\partial^2 F}{\partial y_i \partial y_j} =$
  $\mathbb{E}[f(R(\mathbf{y}) + i + j) - f(R(\mathbf{y}) - i + j) - (f(R(\mathbf{y}) + i - j) - f(R(\mathbf{y}) - i - j))] \geq -\bar{d}$.
- If f is non-monotone: $\frac{\partial^2 F}{\partial y_i \partial y_j} =$
  $\mathbb{E}[f(R(\mathbf{y}) + i + j) - f(R(\mathbf{y}) - i + j) - (f(R(\mathbf{y}) + i - j) - f(R(\mathbf{y}) - i - j))] \geq -(\bar{d} + \underline{d})$.
- If f is normalised, then we have that $F(\mathbf{0}) = 0$.
- F can be lower bounded in terms of f (from [32]):

  Lemma 3.1.2. *Let $f : 2^E \to \mathbb{R}^+$ be a submodular function; let $F : [0,1]^E \to \mathbb{R}^+$ be the multilinear extension of f; and let $\mathbf{y} \in [0,1]^E$ be a vector whose components are bounded by a, i.e. $y_i \leq a, \forall i \in E$. Then, for every $S \subseteq E$, we have that*

  $$F(\mathbf{y} \vee \mathbf{1}_S) \geq (1 - a) f(S). \tag{3.5}$$

Where $\underline{d}, \bar{d} \in \mathbb{R}^+$ are bounds on the absolute value of the minimum and maximum marginal values that the function f can take, i.e. $-\underline{d} \leq f_S(i) \leq \bar{d}$, for all $i \in E$ and $S \subseteq E$. Also, note that by $\mathbf{a} \vee \mathbf{b}$ we denote the vector $\mathbf{c}$ with coordinates $c_i = \max(a_i, b_i)$, i.e. the coordinate-wise maximum. Similarly, $\wedge$ denotes the coordinate-wise minimum.

A drawback of this approach is that, in general, given an arbitrary non-negative submodular function there is no closed form of multilinear extension that enables us to evaluate it efficiently. The usual way to deal with this is to sample it, and so we need the following concentration inequality to quantify the sampling error:

Lemma 3.1.3. *(Hoeffding Bound, Theorem 2 in [40]) Let $X_1, ..., X_m$ be independent random variables such that for each i, $a \leq X_i \leq b$, with $a, b \in \mathbb{R}$. Let $\tilde{X} = \frac{1}{m}\sum_{i=1}^{m} X_i$. Then*

$$\Pr[|\tilde{X} - \mathbb{E}(X)| > t] \leq 2e^{-\frac{2t^2}{(b-a)^2}m}.$$

At this point we are ready to introduce the notion of *marginal value of the multilinear extension*, which is critical for the definition of the continuous greedy process. First, we define it more formally and provide bounds on the error that we incur when we approximate it by sampling the multilinear extension. Then, we define the notion of *maximal marginal improvement* which is the direction that the integration of the continuous greedy process follows. This is a central notion because we can establish a lower bound on the maximum marginal improvement in terms of the optimal solution. We will use these lower bounds later to prove the approximation factors.

Marginal Value A key magnitude used in the Continuous Greedy Algorithm is the marginal value of an element $i \in E$ given a fractional solution $\mathbf{y} \in P(\mathcal{M})$, we refer to it by $\Delta F_i(\mathbf{y})$, defined as follows:

$$\Delta F_i(\mathbf{y}) = F(\mathbf{y} \vee \mathbf{1}_i) - F(\mathbf{y}). \tag{3.6}$$

In other words, it is the value that would be created by adding the full element i to the fractional solution $\mathbf{y}$. Due to the multilinearity of the extension, the marginal value is very closely related to the derivative of the multilinear extension:

$$\frac{\partial F}{\partial y_i} = \frac{F(\mathbf{y} \vee \mathbf{1}_i) - F(\mathbf{y})}{1 - y_i} = \frac{\Delta F_i(\mathbf{y})}{1 - y_i}, \quad \forall i \in E \tag{3.7}$$

or, using vector notation:

$$\mathbf{\Delta F}(\mathbf{y}) = \nabla F \odot (\mathbf{1} - \mathbf{y}), \tag{3.8}$$

where $\odot$ represents element by element multiplication.

In our algorithm we need to estimate the marginal values of each element by sampling, let us bound the error introduced:

Corollary 3.1.4. *Given a non-negative submodular function $f : 2^E \to \mathbb{R}^+$, and a point $\mathbf{y} \in [0,1]^E$; Let $\underline{d}, \bar{d} \in \mathbb{R}^+$ be upper and lower bounds on the marginal values of f, such that $-\underline{d} \leq f_S(j) \leq \bar{d}$ for all $S \subseteq E$ and $j \in E$; let $R_1, R_2, ..., R_m$ be iid samples drawn from $R(\mathbf{y})$, let $w_j(\mathbf{y}) = \frac{1}{m}\sum_{i=1}^{m} f_{R_i}(j)$; and let $f(OPT) = \max_{S \in \mathcal{I}} f(S)$. Then,*

$$\Pr\left(|w_j(\mathbf{y}) - \Delta F_j(\mathbf{y})| \geq \beta\left(\bar{d} + \underline{d}\right)\right) \leq 2e^{-2m\beta^2}.$$

Proof. Immediate application from the Hoeffding bound in Lemma 3.1.3. □

Maximal marginal improvement: $\mu^*(\mathbf{y})$, $\mathbf{v}^*(\mathbf{y})$, and $S_{\mathbf{v}^*(\mathbf{y})}$ Given a fractional solution $\mathbf{y} \in P(\mathcal{M})$, a key part of the continuous greedy process is to find the elements that provide the maximum marginal improvement $\mu^*(\mathbf{y})$. This can be formalised by the following linear program:

$$\mu^*(\mathbf{y}) = \max_{\mathbf{v} \in P(\mathcal{M})} \boldsymbol{\Delta}\mathbf{F}(\mathbf{y})^\intercal \mathbf{v}. \tag{3.9}$$

We will refer to the solution of this linear program by $\mathbf{v}^*(\mathbf{y})$. Note that by definition $\mathbf{v}^*(\mathbf{y}) \in P(\mathcal{M})$, moreover, given that $P(\mathcal{M})$ is a matroid polytope, $\mathbf{v}^*(\mathbf{y})$ is integral, which implies that its coordinates will be either 0 or 1. This allows for $\mathbf{v}^*(\mathbf{y})$ to be casted as the set of elements whose coordinate is 1, we denote such set by $S_{\mathbf{v}^*(\mathbf{y})}$, observe that, naturally, $S_{\mathbf{v}^*(\mathbf{y})} \subseteq E$ and $\mathbf{1}_{S_{\mathbf{v}^*(\mathbf{y})}} = \mathbf{v}^*(\mathbf{y})$. Note that due to the structure of $P(\mathcal{M})$, $\mathbf{v}^*(\mathbf{y})$ will only contain r elements with value 1, where $r = \max_{S \in \mathcal{I}} |S|$ is the rank of the matroid $\mathcal{M}$, thus $|S_{\mathbf{v}^*(\mathbf{y})}| \leq r$ [85]. Intuitively, $S_{\mathbf{v}^*(\mathbf{y})}$ is the the set of elements that yield the most marginal value given the current $\mathbf{y}$.

Crucially, we can relate $\mu^*(\mathbf{y})$ with $f(OPT)$ using the following theorem and its corollaries, which, ultimately, is why the process finds a bounded approximation.

Theorem 3.1.5. *Let $OPT = argmax_{S \in E} f(S)$ be the optimal solution. Then, for any point $\mathbf{y} \in P(\mathcal{M})$,*

$$\mu^*(\mathbf{y}) \geq F(\mathbf{y} \vee \mathbf{1}_{OPT}) - F(\mathbf{y}) \tag{3.10}$$

Proof. By definition, the solution represented by $OPT = \text{argmax}_{S \in \mathcal{I}} f(S)$ is feasible, hence, $\mathbf{1}_{\text{OPT}} \in P(\mathcal{M})$. Then, by definition of $\mu^*(\mathbf{y})$, we have that $\mu^*(\mathbf{y}) \geq \sum_{i \in \text{OPT}} \Delta F_i(\mathbf{y})$. Using the definition of $\Delta F_i(\mathbf{y})$, $\sum_{i \in \text{OPT}} \Delta F_i(\mathbf{y}) = \sum_{i \in \text{OPT}} F(\mathbf{y} \vee \mathbf{1}_i) - F(\mathbf{y})$. Now, from the submodularity condition, for any element $i \in E$ and any subset of elements $S \subseteq E$, we have that $\sum_{i \in S}(F(\mathbf{y} \vee \mathbf{1}_i) - F(\mathbf{y})) \geq F(\mathbf{y} \vee \mathbf{1}_S) - F(\mathbf{y})$. Hence, $\sum_{i \in \text{OPT}} F(\mathbf{y} \vee \mathbf{1}_i) - F(\mathbf{y}) \geq F(\mathbf{y} \vee \mathbf{1}_{\text{OPT}}) - F(\mathbf{y})$. This yields immediately that $\mu^*(\mathbf{y}) \geq F(\mathbf{y} \vee \mathbf{1}_{\text{OPT}}) - F(\mathbf{y})$. □

Now we can use lemma 3.1.2, to bound $\mu^*(\mathbf{y})$ in terms of $f(OPT)$, i.e., the optimal solution of the discrete problem.

Corollary 3.1.6. *Let f be a non-negative subomdular function and F be its multilinear extension. And let $\bar{y}$ be an upper bound on the coordinates of $\mathbf{y}$, i.e. $y_i \leq \bar{y}, \forall i \in E$. Then, for any point $\mathbf{y} \in P(\mathcal{M})$,*

$$\mu^*(\mathbf{y}) \geq (1 - \bar{y}) f(OPT) - F(\mathbf{y}) \tag{3.11}$$

Proof. From theorem 3.1.5 we have that $\mu^*(\mathbf{y}) \geq F(\mathbf{y} \vee \mathbf{1}_{\text{OPT}}) - F(\mathbf{y})$. Now, using lemma 3.1.2, it follows that $\sum_{i \in E} F(\mathbf{y} \vee \mathbf{1}_{\text{OPT}}) - F(\mathbf{y}) \geq \sum_{i \in E}(1 - \bar{y}) f(\text{OPT}) - F(\mathbf{y})$. Hence, we can rearrange to yield $\mu^*(\mathbf{y}) \geq (1 - \bar{y}) f(\text{OPT}) - F(\mathbf{y})$. □

Furthermore, if f is monotone, we can establish a tighter bound:

Corollary 3.1.7. *Let f be a non-negative monotone subomdular function and F be its multilinear extension. Then, for any point $\mathbf{y} \in P(\mathcal{M})$,*

$$\mu^*(\mathbf{y}) \geq f(OPT) - F(\mathbf{y}) \tag{3.12}$$

Proof. From the monotonicity condition and the definition of the multilinear extension we have: $F(\mathbf{y} \vee \mathbf{1}_{\mathrm{OPT}}) \geq F(\mathbf{1}_{\mathrm{OPT}}) = f(OPT)$. Thus from Theorem 3.1.5 the result is immediate.

□

Now we have all the tools to present the *continuous greedy process*, and to explain why it yields a good solution to the relaxation of our problem.

The Continuous Greedy Process The objective is to find a good solution to the relaxation problem $\max_{\mathbf{y} \in P(\mathcal{M})} F(\mathbf{y})$ that can be rounded. The main issue is that F is neither convex nor concave, and an efficient algorithm to solve it does not exist. However, the Continuous Greedy Process can find a point that is within a constant factor of the optimal. We are now equipped with all the concepts needed to describe the Continuous Greedy Process. Initially, the process was presented by Jan Vondrak [15, 96] in the following form: consider a particle starting at $t = 0$ at the point $\mathbf{y}(0) = \mathbf{0}$, and integrate the process

$$\frac{d\mathbf{y}}{dt} = \mathbf{v}^*(\mathbf{y}(t)) \tag{3.13}$$

until t = 1. This yields a point $\mathbf{y}^* = \mathbf{y}(1)$, such that $F(\mathbf{y}^*) \geq (1 - \frac{1}{e}) f(OPT)$. This process works only when f is monotone. In light of this, Feldman [32] proposed an algorithm that integrated a modified version that could find a constant-factor approximation for both the monotone and non-monotone case. This was called the Measured Continuous Greedy Process, and it is the one we will focus on for the remainder of the chapter. It can be defined as follows: consider a particle starting at $t = 0$ at point $\mathbf{y}(0) = \mathbf{0}$, and integrate the process

$$\frac{d\mathbf{y}}{dt} = (\mathbf{1} - \mathbf{y}(t)) \odot \mathbf{v}^*(\mathbf{y}(t)), \tag{3.14}$$

where $\odot$ represents element by element multiplication, until t = 1. This gives us a point $\mathbf{y}^* = \mathbf{y}(1)$, such that $F(\mathbf{y}^*) \geq (1 - \frac{1}{e}) f(OPT)$ when f is monotone, but also, crucially, it yields a point such that $F(\mathbf{y}^*) \geq \frac{1}{e} f(OPT)$ when f is non-monotone.

To understand why it yields these approximations, first, keep in mind that f is normalised, i.e. $f(\emptyset) = 0$, hence $F(\mathbf{0}) = 0$. Second, let us look at the derivative of $F(\mathbf{y}(t))$ with respect to t. Using the chain rule we have

$$\frac{dF}{dt} = \nabla F \cdot \frac{d\mathbf{y}}{dt} \tag{3.15}$$

which, noting that $\nabla F \cdot \frac{d\mathbf{y}}{dt} = \nabla F \cdot ((\mathbf{1} - \mathbf{y}) \odot \mathbf{v}^*(\mathbf{y})) = \mathbf{\Delta F}(\mathbf{y}) \cdot \mathbf{v}^*(\mathbf{y}) = \mu^*(\mathbf{y})$ due to the multilinearity of F, implies

$$\frac{dF}{dt} = \mu^*(\mathbf{y}(t)). \tag{3.16}$$

This rate of increase is the reason why it yields the approximation guarantees, let us study first the monotone case.

In the monotone case, from Corollary 3.1.7, we have that $\frac{dF}{dt} = \mu^*(\mathbf{y}(t)) \geq f(\text{OPT}) - F(\mathbf{y}(t))$. By making the inequality tight, we can define a function $\eta(t)$ which satisfies $\frac{d\eta}{dt} = f(\text{OPT}) - \eta(t)$, that lower bounds F, i.e. $F(\mathbf{y}(t)) \geq \eta(t)$. Solving it, with $\eta(0) = 0$, yields $\eta(t) = (1 - e^{-t})f(OPT)$. Hence, $F(\mathbf{y}(1)) \geq \eta(1) = (1 - \frac{1}{e})f(OPT)$.

Let us now investigate the non-monotone case. From Corollary 3.1.6, case we have that $\frac{dF}{dt} = \mu^*(\mathbf{y}(t)) \geq (1 - \bar{y}(t))f(\text{OPT}) - F(\mathbf{y}(t))$. First, we establish the upper bound on the coordinates of $\mathbf{y}(t)$, i.e. $\bar{y}(t)$. Since $P(\mathcal{M})$ is a matroid polytope any $\mathbf{v}^*(\mathbf{y}(t))$ coordinate can take at most value 1, the equation $\frac{d\bar{y}}{dt} = 1 - \bar{y}(t)$ gives such upper bound, and by integrating with $\bar{y}(0) = 0$, we get $\bar{y}(t) = 1 - e^{-t}$. Now, we can write $\frac{dF}{dt} \geq e^{-t}f(\text{OPT}) - F(\mathbf{y}(t))$. Again, as in the monotone analysis, we can establish a function $\zeta(t)$ as a lower bound of F by making the inequality tight: $\frac{d\zeta}{dt} = e^{-t}f(\text{OPT}) - \zeta(t)$, which integrating, with $\zeta(0) = 0$, yields $\zeta(t) = f(OPT)e^{-t}t$. Hence, $F(\mathbf{y}(1)) \geq \zeta(1) = \frac{1}{e}f(OPT)$.

Rounding This is the step that takes a fractional solution to the relaxation problem and produces a discrete solution while keeping the approximation ratios. In this work we shall not dwell on the rounding step. Suffice it to say that there exist algorithms, such as Contention Resolution Schemes [17], Pipage-Rounding [4, 15, 98], or Swap-Rounding [99], that given a general non-negative submodular function, and a point $\mathbf{y} \in P(\mathcal{M})$, can find a set $S \in \mathcal{I}$, such that $f(S) \geq F(\mathbf{y})$. The most efficient technique for a general matroid constraint is Swap-Rounding, and its results are used in [6]. However, the swap-rounding results in [99] hinge around Chernoff-like concentration bounds for monotone submodular functions, but in light of the concentration bounds for non-monotone submodular functions in [97], we believe that this technique can be adapted to work with non-monotone functions. In the case of the Task Allocation problem, we have a partition matroid which can be rounded very efficiently ($\text{O}(E)$, better than Swap-Rounding) for both monotone and non-monotone cases via a simple randomised rounding procedure, see [15].

3.2 Smooth Measured Continuous Greedy

After all the necessary preliminaries, we can now explain our first improvement to the measured continuous greedy algorithm. First, we motivate it by describing the key observation that underpins its rationale. And then, we proceed to describe and analyse it in detail.

3.2.1 Algorithm Definition

Key Observation

In essence, Vondrak's [15] and Feldman's [32] algorithms are a simple Euler integration of Equations 3.13 and 3.14 respectively. It is well known that the Euler method is not the most computationally efficient way to integrate a differential equation. To investigate which method to use instead, let us look at the integral that needs to accrue for the increment of $\mathbf{y}$ in a single step:

$$\mathbf{y}(t+\delta) - \mathbf{y}(t) = \int_0^{\delta} (\mathbf{1} - \mathbf{y}(t+\tau)) \odot \mathbf{v}^*(\mathbf{y}(t+\tau))d\tau. \tag{3.17}$$

The classic approach to integrate the above equation is by using a formula of the kind $\mathbf{y}(t+\delta) = \mathbf{y}(t) + \mathbf{q}\delta$, where $\mathbf{q}$ is an estimate of the average gradient (i.e., the integrand above) along the interval. There are two common methods used to estimate $\mathbf{q}$. The first option is to use integration strategies that sample intermediate points along the interval between t and $t+\delta$, such as the classic Runge-Kutta methods, to find an estimate of $\mathbf{q}$. However, for Equation 3.14 these methods would not be efficient because $\mathbf{v}^*$ is very expensive to compute. This is because to compute $\mathbf{v}^*$ we need to estimate ΔF_i for each element i, and this can only be computed in practice by sampling, which is time consuming. The second option would be to use linear multistep methods that use previous points in the integration to refine the estimate of the gradient $\mathbf{q}$, such as Adams methods or Backward Differentiation Formula (BDF) methods. This, however, would be inadequate for Equation 3.14 because the coordinates of $\mathbf{v}^*$ are either 0 or 1, thus, with this discontinuity, knowledge of the previous steps would not inform the next value.

What we propose, instead, is to use the intrinsic characteristics of the terms in the integrand to find an increment rule that remains accurate even with large steps. The integrand has two terms that vary with τ: $\mathbf{y}$ and $\mathbf{v}^*$. The key to our algorithm is the observation that, in practice, $\mathbf{v}^*$ changes only with relatively large stepsizes, while it remains constant around a small region. This is because it is the solution to a linear program and its solution can still be valid even when the objective weights $\Delta\mathbf{F}$ change slightly. Now, if $\mathbf{v}^*$ is assumed to remain constant during the integration step, an exact solution to the integral can be found. Such an increment rule is given by the solution of the differential equation:

$$\frac{d\mathbf{y}}{d\tau} = (\mathbf{1} - \mathbf{y}(t+\tau)) \odot \mathbf{v}^*(\mathbf{y}(t)), \tag{3.18}$$

where t is a constant, hence $\mathbf{v}^*(\mathbf{y}(t))$ is a constant, and the solution of the equation can be readily obtained as:

$$y_i(t+\tau) = 1 + e^{-v_i^*(\mathbf{y}(t))\tau}(y_i(t) - 1) \tag{3.19}$$

which we use as the update step. The correctness of this approach depends on whether δ is small enough for $\mathbf{v}^*(\mathbf{y}(t)) = \mathbf{v}^*(\mathbf{y}(t+\delta))$ to be true. This, as we shall see in the numerical experiments section, is extremely effective in practice because it obtains the same solution that [32], but with orders of magnitude larger step sizes, and thus, orders of magnitude less evaluations of the costly magnitude $\mathbf{v}^*(\mathbf{y}(t))$.

Algorithm

Let us now define our main result in Algorithm 1. Its inputs are: the ground set E; a facility to evaluate the multilinear extension of the utility function $F : [0,1]^E \to \mathbb{R}^+$; and finally, the stepsize $\delta \ll 1$, which is a small scalar that defines the size of the step taken in each iteration. Note that since the multilinear extension cannot be evaluated exactly, it is evaluated by taking the expected

value of a given number of samples. Also, δ must be an exact divisor of 1. The main loop begins after the initialisation of the relaxation $\mathbf{y}$ to 0. The main iteration evaluates $F(\mathbf{y}(t) \vee \mathbf{1}_i) - F(\mathbf{y}(t))$, i.e. $\Delta F_i(y(t))$, which is the marginal value that the element $i \in E$ would add given the current fractional solution $\mathbf{y}$. We do this averaging $O(\frac{\log(|E|)}{\delta^2 r^2})$ samples of $f_{R\mathbf{y}(t)}(i)$. (This is to achieve an additive error $\delta(\bar{d} + \underline{d})r^2$). Next, the algorithm finds the elements that achieve the most marginal value, $\mathbf{v}^*$. This linear program can be solved using a simple greedy algorithm because $P(\mathcal{M})$ is a matroid polytope. The last step in the iteration is to increment the coordinates of the elements that were found to yield the largest marginal values, $\mathbf{v}^*$, using the rule $y_i(t+\tau) = 1+e^{-v_i^*(\mathbf{y}(t))\tau}(y_i(t)-1)$ as we have previously explained. The algorithm keeps iterating until t reaches 1, and return $\mathbf{y}(1)$ which satisfies: $F(\mathbf{y}(1)) \geq (1 - \frac{1}{e})f(OPT)$ for a monotone f; and $F(\mathbf{y}(1)) \geq \frac{1}{e}f(OPT)$ for a non-monotone f.

Algorithm 1: Smooth Measured Continuous Greedy Algorithm

Input : $f : 2^E \to \mathbb{R}^+$, $\epsilon \in [0, 1]$, $\mathcal{I} \subseteq 2^E$.
Output: A point $\mathbf{y} \in P(\mathcal{M})$

$\mathbf{y}(0) \leftarrow \mathbf{0}$ `// initialisation`

for $t = \{0, \delta, 2\delta, 3\delta, \ldots, 1 - \delta\}$ **do**
  for $i \in E$ **do**
    $\tilde{\Delta F}_i(t) \leftarrow F(\mathbf{y}(t) \vee \mathbf{1}_i) - F(\mathbf{y}(t))$
    `// Sampling up to an additive error of` $\delta(\bar{d} + \underline{d})r^2$ `using Corollary 3.1.4.`
  $\tilde{\mathbf{v}}^*(\mathbf{y}(t)) \leftarrow \text{argmax}_{\mathbf{v} \in P(\mathcal{M})} \tilde{\mathbf{\Delta F}}(\mathbf{y}(t))^T \mathbf{v}$
  for $i \in E$ **do**
    $y_i(t + \delta) \leftarrow 1 + e^{-\tilde{v}_i^*(\mathbf{y}(t))\delta}(y_i(t) - 1)$

Return: $\mathbf{y}(1)$

3.2.2 Analysis

In this section we carry out the analysis that proves that our algorithm yields the stated guarantees. First, we show that the algorithm produces a feasible point, i.e., a point inside the matroid polytope $P(\mathcal{M})$. Then, we prove the approximation ratio for the non-monotone and the monotone cases. Finally we study the complexity of the algorithms.

Feasibility

First, we need to show that our algorithm produces a feasible point, i.e. $\mathbf{y}(1) \in P(\mathcal{M})$.

Theorem 3.2.1. *The proposed algorithm produces a feasible fractional solution, i.e.,* $\mathbf{y}(1) \in P(\mathcal{M})$.

Proof. We follow the approach used by [32]. We first define a vector $\mathbf{x}$ that coordinate wise upper-bounds $\mathbf{y}(1)$. Then, given that $P(\mathcal{M})$ is down-monotone, we only need to show that $\mathbf{x}$ is in $P(\mathcal{M})$ to show that $\mathbf{y}(1) \in P(\mathcal{M})$. Consider

the vector $\mathbf{x} = \delta \sum_{l=0}^{\frac{1}{\delta}-1} \tilde{\mathbf{v}}^*(\mathbf{y}(l\delta))$. This is a coordinate-wise upper bound of $\mathbf{y}(1)$ because when $i \in S_{\tilde{\mathbf{v}}^*}$, we have that $y_i(t+\delta) - y_i(t) = 1 + e^{-\delta}(y_i(t) - 1) - y_i(t) = (1 - e^{-\delta})(1 - y_i(t)) \leq 1 - e^{-\delta} \leq \delta$, for all $\delta \in [0, 1]$; and when $i \notin S_{\tilde{\mathbf{v}}^*}$ $y_i(t+\delta) - y_i(t) = 0$. We now show that $\mathbf{x}$ is in $P(\mathcal{M})$. First, note that by definition $\tilde{\mathbf{v}}^* \in P(\mathcal{M})$. Then, observe that $\mathbf{x}/\delta$ is the sum of $\frac{1}{\delta}$ points in $P(\mathcal{M})$, thus $(\mathbf{x}/\delta)/(1/\delta) = \mathbf{x}$ is a convex combination of points in $P(\mathcal{M})$, hence $\mathbf{x} \in P(\mathcal{M})$, and consequently $\mathbf{y}(1) \in P(\mathcal{M})$. □

In fact, it is possible to find a point that still lies in $P(\mathcal{M})$ even with a $t > 1$, specifically stopping at a value depending on a magnitude called the density of the matroid. This yields tighter approximation bounds that match those found for particular matroids, such as partition matroids. Asymptotically, however, these bounds are the same than those presented here. Therefore, in the aim of simplicity, the analysis is carried out with a stopping time of 1. The interested reader is referred to [32].

Now let us establish a bound to the coordinates of $\mathbf{y}(t)$ that will become useful.

Lemma 3.2.2. *At time $0 \leq t \leq 1$, we have that:*

$$y_i(t) \leq 1 - e^{-t}, \quad \forall i \in E. \tag{3.20}$$

Proof. Consider the recurrence $g(n+1) = 1 + e^{-\delta}(g(n) - 1)$, with $g(0) = 0$. This recurrence has the following solution: $g(n) = 1 - e^{-n\delta}$. Now, in our algorithm t is incremented linearly, so the number of iteration, n, and t are related by $t = n\delta$. At t, all the coordinates of $\mathbf{y}$ follow $y(t+\delta) = 1 + e^{-\delta\tilde{v}^*}(y(t) - 1)$, thus, they stay constant for $\tilde{v}^* = 0$, or increase for $\tilde{v}^* = 1$. Hence, the recurrence g is an upper bound because it corresponds to incrementing in each and every iteration. Consequently, at t, $y(t) \leq g(\frac{t}{\delta}) = g(n) = 1 - e^{-t}$. □

Approximation Ratio: The Non-Monotone Case

Now, we shall carry out the derivation of the approximation bounds. In our analysis we first develop bounds in the derivatives of the multilinear extension and the marginal values with respect to the increment taken. Then, we use these to bound the improvement achieved in a given iteration. Once we know the improvement in a given iteration, we can pose a recurrence relation whose solution yields the approximation ratios.

Let us now find a bound for the derivative of the marginal values.

Lemma 3.2.3. *Let $\underline{d}, \bar{d} \in \mathbb{R}^+$ be bounds on the absolute value of the minimum and maximum marginal values that the function f can take, i.e. $-\underline{d} \leq f_S(i) \leq \bar{d}$, for all $i \in E$ and $S \subseteq E$. Then, for $0 \leq t \leq 1$ the following condition holds:*

$$\frac{d}{d\tau} \sum_{i \in S_{\tilde{\mathbf{v}}^*(\mathbf{y}(t))}} \Delta F_i(\mathbf{y}(t+\tau)) \geq -r^2(\underline{d} + \bar{d}) \tag{3.21}$$

where r is the matroid rank.

Proof. First we bound the derivative of the marginal value of a single element:

$$
\begin{aligned}
&\frac{d}{d\tau}\Delta F_i(\mathbf{y}(t+\tau)) = \\
&= \frac{d}{d\tau}\left(\frac{\partial F(\mathbf{y}(t+\tau))}{\partial y_i}(1-y_i(t+\tau))\right) \\
&= \frac{d}{d\tau}\left(\frac{\partial F(\mathbf{y}(t+\tau))}{\partial y_i}\right)(1-y_i(t+\tau)) + \left(\frac{\partial F(\mathbf{y}(t+\tau))}{\partial y_i}\right)\frac{d}{d\tau}(1-y_i(t+\tau)) \\
&= (1-y_i(t+\tau))\sum_{k\in S_{\tilde{\mathbf{v}}^*(\mathbf{y}(t))}}\left(\frac{\partial^2 F(\mathbf{y}(t+\tau))}{\partial y_i \partial y_k}\frac{dy_k(t+\tau)}{d\tau}\right) - \left(\frac{\partial F(\mathbf{y}(t+\tau))}{\partial y_i}\right)(1-y_i(t+\tau)) \\
&= (1-y_i(t+\tau))\sum_{k\in S_{\tilde{\mathbf{v}}^*(\mathbf{y}(t))}}\left(\frac{\partial^2 F(\mathbf{y}(t+\tau))}{\partial y_i \partial y_k}(1-y_k(t+\tau))\right) - \Delta F_i(\mathbf{y}(t+\tau)) \\
&= (1-y_i(t+\tau))\sum_{k\in S_{\tilde{\mathbf{v}}^*(\mathbf{y}(t))}\setminus i}\left(\frac{\partial^2 F(\mathbf{y}(t+\tau))}{\partial y_i \partial y_k}(1-y_k(t+\tau))\right) - \Delta F_i(\mathbf{y}(t+\tau)) \\
&\geq (1-y_i(t+\tau))\sum_{k\in S_{\tilde{\mathbf{v}}^*(\mathbf{y}(t))}\setminus i}\left(-(\bar{d}+\underline{d})(1-y_k(t+\tau))\right) - \underline{d} \\
&\geq (r-1)(-(\bar{d}+\underline{d})) - \underline{d} \\
&\geq -r(\bar{d}+\underline{d}).
\end{aligned}
$$

Now we can bound the derivative of the sum of marginal values of all selected elements:

$$
\frac{d}{d\tau}\sum_{i\in S_{\tilde{\mathbf{v}}^*(\mathbf{y}(t))}}\Delta F_i(\mathbf{y}(t+\tau)) \geq -r^2(\bar{d}+\underline{d}).
$$

□

At this point, the bounds derived earlier can be combined to relate the slope of the value of our solution to the increment made.

Lemma 3.2.4. *At time $0 \leq t \leq 1$, and with $0 \leq \tau \leq \delta$, we have that:*

$$
\frac{dF(\mathbf{y}(t+\tau))}{d\tau} \geq \mu^*(\mathbf{y}(t)) - 2r^2(\bar{d}+\underline{d})\delta \tag{3.22}
$$

Proof. We have that:

$$
\begin{aligned}
\frac{dF(\mathbf{y}(t+\tau))}{d\tau} &= \sum_{i\in S_{\tilde{\mathbf{v}}^*(\mathbf{y}(t))}}\left(\frac{\partial F(\mathbf{y}(t+\tau))}{\partial y_i}\frac{dy_i(t+\tau)}{d\tau}\right) \\
&= \sum_{i\in S_{\tilde{\mathbf{v}}^*(\mathbf{y}(t))}}\left(\frac{\partial F(\mathbf{y}(t+\tau))}{\partial y_i}(1-y_i(t+\tau))\right) \\
&= \sum_{i\in S_{\tilde{\mathbf{v}}^*(\mathbf{y}(t))}}\Delta F_i(\mathbf{y}(t+\tau)).
\end{aligned}
$$

The first equality comes from applying the chain rule, noting that the only element that change are those whose component in $\mathbf{v}^*$; the second is by the update rule derivative, $\frac{dy_i(t+\tau)}{d\tau} = (1 - y_i(t+\tau))$; and the third equality comes from the definition of the marginal value.

Now, due to the C2 continuity of the multilinear extension, ΔF_i will also have first order derivatives, so we can use Taylor's theorem to expand:

$$\sum_{i \in S_{\tilde{\mathbf{v}}^*(\mathbf{y}(t))}} \Delta F_i(\mathbf{y}(t+\tau)) = \sum_{i \in S_{\tilde{\mathbf{v}}^*(\mathbf{y}(t))}} \Delta F_i(\mathbf{y}(t)) + \int_0^\tau \frac{d}{ds} \sum_{i \in S_{\tilde{\mathbf{v}}^*(\mathbf{y}(t))}} \Delta F_i(\mathbf{y}(t+s)) ds. \tag{3.23}$$

And, with Lemma 3.2.3, we can lower bound the value of the integral:

$$\int_0^\tau \frac{d}{ds} \sum_{i \in S_{\tilde{\mathbf{v}}^*(\mathbf{y}(t))}} \Delta F_i(\mathbf{y}(t+s)) ds \geq -r^2(\underline{d} + \bar{d})\tau. \tag{3.24}$$

Now, in our algorithm we are assuming that we do not have access directly to the multilinear extension, this is the most general, and pessimistic, case (though for many interesting functions such a closed form does exist). Therefore, we need to sample it, and account for the error that the sampling introduces. That is, instead of having access to $\Delta F_i(\mathbf{y}(t))$ we have access to $\tilde{\Delta F}_i(\mathbf{y}(t))$ which is a sampled version. We need to subtract an upper bound on the error introduced by sampling using Corollary 3.1.4, that is: $\tilde{\Delta F}_i(\mathbf{y}(t)) \geq \Delta F_i(\mathbf{y}(t)) - \delta r(\bar{d} + \underline{d})$. And, therefore, we can bound the value of maximal set that our algorithm selects with respect to the optimal accounting for the degradation introduced by sampling :

$$\max_{\mathbf{v} \in P(\mathcal{M})} \tilde{\mathbf{\Delta F}}(\mathbf{y}(t))^T \mathbf{v} \geq \max_{\mathbf{v} \in P(\mathcal{M})} \left(\mathbf{\Delta F}(\mathbf{y}(t))^T \mathbf{v}\right) - \delta r^2(\bar{d}+\underline{d}) = \mu^*(\mathbf{y}(t)) - r^2(\bar{d}+\underline{d})\delta$$

hence:

$$\sum_{i \in S_{\tilde{\mathbf{v}}^*(\mathbf{y}(t))}} \Delta F_i(\mathbf{y}(t)) \geq \mu^*(\mathbf{y}(t)) - r^2(\bar{d} + \underline{d})\delta. \tag{3.25}$$

Now we can plug into Equation 3.23 the bounds in Equations 3.24 and 3.25 to yield:

$$\frac{dF(\mathbf{y}(t+\tau))}{d\tau} \geq \mu^*(\mathbf{y}(t)) - r^2(\bar{d} + \underline{d})\delta - r^2(\bar{d} + \underline{d})\tau. \tag{3.26}$$

Which, given that $0 \leq \tau \leq \delta$, yields the result:

$$\frac{dF(\mathbf{y}(t+\tau))}{d\tau} \geq \mu^*(\mathbf{y}(t)) - 2r^2(\bar{d} + \underline{d})\delta. \tag{3.27}$$

□

Given the slope with respect to the increment, the value gained between iterations can be bounded.

Corollary 3.2.5. *At time $0 \leq t \leq 1$, the following inequality holds:*

$$F(\mathbf{y}(t+\delta)) - F(\mathbf{y}(t)) \geq \delta \left(\mu^*(\mathbf{y}(t)) - 2r^2\delta(\bar{d} + \underline{d})\right) \tag{3.28}$$

Proof. Similarly as we did in the Corollary 3.2.4 above, C2 continuity allows us to write: $F(\mathbf{y}(t+\delta)) = F(\mathbf{y}(t)) + \int_0^\delta \frac{dF(\mathbf{y}(t+\tau))}{d\tau} d\tau$. And we can bound the integral term with corollary 3.2.4: $\int_0^\delta \frac{dF(\mathbf{y}(t+\tau))}{d\tau} d\tau \geq \delta \left(\mu^*(\mathbf{y}(t)) - 2r^2(\bar{d} + \underline{d})\delta\right)$. Finally, rearranging yields the result. □

This can further refined using the bounds on the value of $\mu^*(\mathbf{y}(t))$.

Corollary 3.2.6. *At time $0 \leq t \leq 1$, we have that:*

$$F(\mathbf{y}(t+\delta)) - F(\mathbf{y}(t)) \geq \delta(e^{-t} f(OPT) - F(\mathbf{y}(t)) - 2r^2(\bar{d} + \underline{d})\delta) \tag{3.29}$$

Proof. Combining Lemma 3.2.5, Corollary 3.1.6, and Lemma 3.2.2, it is trivial that the result holds. □

So far we have a bound on the improvement on F at every iteration provided by corollary 3.2.6. Now we use it to form a recurrence whose solution describes the evolution in the accumulated value of $F(\mathbf{y})$.

Lemma 3.2.7. *At time $0 \leq t \leq 1$, we have that:*

$$F(\mathbf{y}(t)) \geq \frac{\delta e^\delta (e^{-t} - (1-\delta)^{t/\delta})}{e^\delta(\delta - 1) + 1} f(OPT) - (1 - (1-\delta)^{t/\delta})\delta r^2 2(\bar{d} + \underline{d}) \tag{3.30}$$

Proof. Consider the recurrence $a(n+1) = k_1 a(n) + k_2 e^{-n\delta} - k_3$, with $a(0) = 0$. This recurrence has the following solution: $a(n) = \frac{e^\delta(k_1^n - e^{-\delta n})}{e^\delta k_1 - 1} k_2 - \frac{k_1^n - 1}{k_1 - 1} k_3$. Given Corollary 3.2.6, we can see that with $t = n\delta$, $k_1 = 1 - \delta$, $k_2 = \delta f(\text{OPT})$, and $k_3 = 2\delta^2 r^2(\bar{d} + \underline{d})$, this recurrence is equivalent to the increment made by our algorithm in every step. Then, substituting all these coefficients and simplifying yields: $F(\mathbf{y}(t)) \geq \frac{\delta e^\delta (e^{-t} - (1-\delta)^{t/\delta})}{e^\delta(\delta-1)+1} f(\text{OPT}) - (1 - (1-\delta)^{t/\delta})\delta r^2 2(\bar{d} + \underline{d})$. □

This can be simplified to:

Corollary 3.2.8. *With a non-monotone non-negative submodular function, Algorithm 1 returns a point $\mathbf{y}^*$ such that $\mathbf{y}^* \in P(\mathcal{M})$ and:*

$$F(\mathbf{y}^*) \geq \left(\frac{1}{e} - \epsilon\right) f(OPT) \tag{3.31}$$

Proof. From Lemma 3.2.7, at time $t = 1$, we have that $F(\mathbf{y}(1)) \geq \frac{\delta e^\delta (e^{-1} - (1-\delta)^{1/\delta})}{e^\delta(\delta-1)+1} f(\text{OPT}) - (1 - (1-\delta)^{1/\delta})\delta r^2 2(\bar{d} + \underline{d})$. The term with $f(\text{OPT})$ can be simplified by observing that for $0 \leq \delta \leq 1$ we have that $\frac{\delta e^\delta (e^{-1} - (1-\delta)^{1/\delta})}{e^\delta(\delta-1)+1} \geq \frac{1}{e}$. The second term can be also be simplified by observing that $(1 - (1-\delta)^{1/\delta}) \leq 1$. Now, we can set $\epsilon = 2r^2 \frac{\bar{d} + \underline{d}}{\bar{d}} \delta$, and since by definition $f(\text{OPT}) \geq \bar{d}$, we have that $\epsilon = 2r^2 \frac{\bar{d} + \underline{d}}{\bar{d}} \delta \geq 2r^2 \frac{\bar{d} + \underline{d}}{f(\text{OPT})} \delta$, which yields the result. Finally, from Theorem 3.2.1 we have that $\mathbf{y}^* \in P(\mathcal{M})$. □

Approximation Ratio: The Monotone Case

Let us now prove the tighter bounds achievable if the function f is assumed to be monotone. The first thing to note is that all the theorems and facts from the non-monotone case are still valid in the monotone case. The main difference lies in the lower bound that can be imposed on $\mu^*(\mathbf{y})$ using the monotonicity condition. This has implications for the value gained in each step, which is used to form a recurrence whose solution yields the approximation result.

First we use Theorem 3.2.5 with the bound for $\mu(\mathbf{y})$ for the monotone case to bound the improvement on a given step.

Corollary 3.2.9. *At $0 \leq t \leq 1$, with monotone objective functions, the following inequality holds:*

$$F(\mathbf{y}(t+\delta)) - F(\mathbf{y}(t)) \geq \delta(f(OPT) - F(\mathbf{y}(t)) - 2r^2\bar{d}\delta) \tag{3.32}$$

Proof. Inmediate from Theorem 3.2.5 and Corollary 3.1.7. Noting that for a monotone function $\underline{d} = 0$. □

Now the recurrence that bounds the value can be simplified:

Lemma 3.2.10. *At $0 \leq t \leq 1$, with monotone objective functions, we have that:*

$$F(\mathbf{y}(t)) \geq (1 - (1-\delta)^{t/\delta})(f(OPT) - 2\delta r^2\bar{d}) \tag{3.33}$$

Proof. Consider the recurrence $a(n+1) = k_1 a(n) + k_2$, with $a(0) = 0$. This recurrence has the following solution: $a(n) = \frac{k_2(k_1^n - 1)}{k_1 - 1}$. Given Corollary 3.2.9, we can see that with $t = n\delta$, $k_1 = 1-\delta$, $k_2 = \delta f(\mathrm{OPT}) - 2\delta^2 r^2 \bar{d}$, this recurrence is equivalent to the increment made by our algorithm in every step. Then, substituting all these coefficients and simplifying yields: $F(\mathbf{y}(t)) \geq (1 - (1 - \delta)^{t/\delta})(f(\mathrm{OPT}) - 2\delta r^2\bar{d})$. □

Now the recurrence that bounds the value can be solved:

Corollary 3.2.11. *With a monotone non-negative submodular function, Algorithm 1 returns a point $\mathbf{y}^*$ such that $\mathbf{y}^* \in P(\mathcal{M})$ and:*

$$F(\mathbf{y}^*) \geq \left(1 - \frac{1}{e} - \epsilon\right) f(OPT) \tag{3.34}$$

Proof. From Lemma 3.2.10, we have that at the end of the algorithm, i.e. $t = 1$, $F(\mathbf{y}(1)) \geq (1 - (1-\delta)^{1/\delta})(f(\mathrm{OPT}) - 2\delta r^2\bar{d})$. Now, note that $(1 - (1-\delta)^{1/\delta}) \geq 1 - \frac{1}{e}$. Thus we have: $F(\mathbf{y}(1)) \geq \left(1 - \frac{1}{e}\right)(f(\mathrm{OPT}) - 2\delta r^2\bar{d})$. Further, we have that $f(\mathrm{OPT}) \geq \bar{d}$, therefore: $F(\mathbf{y}(1)) \geq \left(1 - \frac{1}{e} - 2\delta r^2\right) f(\mathrm{OPT})$. Thus, we set $\epsilon = 2\delta r^2$ to have the result. Finally, from Theorem 3.2.1 we have that $\mathbf{y}^* \in P(\mathcal{M})$.

□

Time Complexity

Here we quantify the running time in terms of value oracle calls to the submodular function and matroid independence oracle calls. First we quantify the cost of the most expensive step in a single iteration: finding $\tilde{\mathbf{v}}^*$. Then, we estimate the number of steps in the integration to quantify the total cost.

Lemma 3.2.12. *Finding $\tilde{\mathbf{v}}^*$ requires $O\left(\frac{r^2}{\epsilon^2}|E|\log(|E|)\left(\frac{\bar{d}+\underline{d}}{\bar{d}}\right)^2\right)$ value oracle calls, and $O(|E|)$ independence oracle calls.*

Proof. With Lemma 3.1.4, estimating each $\Delta\tilde{F}_i$ up to a additive error of $\delta r(\bar{d}+\underline{d})$ with high probability takes $O(\frac{\log(|E|)}{\delta^2 r^2})$ value oracle calls. We have to estimate $\Delta\tilde{F}_i$ for each $i \in E$, hence estimating all the marginal values takes:

$$O\left(\frac{1}{\delta^2 r^2}|E|\log(|E|)\right) \tag{3.35}$$

value oracle calls. Now let's study the cost of solving $\max_{\mathbf{v}\in P(\mathcal{M})} \tilde{\mathbf{\Delta F}}(\mathbf{y}(t))^T\mathbf{v}$. Given that $P(\mathcal{M})$ is a matroid polytope, we can solve it using the greedy algorithm [85]. This costs $O(|E|\log(|E|))$ steps to sort all the $\Delta\tilde{F}_i$ for $i \in E$ and $O(|E|)$ independence oracle calls to pass over them in sorted order to select the solution.

Now we express the cost in terms of the factor ϵ. Recall that in Corollary 3.2.8 we set $\epsilon = 2r^2\frac{\bar{d}+\underline{d}}{\bar{d}}\delta$, hence $\delta = \frac{1}{2r^2}\frac{\bar{d}}{\bar{d}+\underline{d}}\epsilon$. Which we can plug into Equation 3.35 above to obtain the total number of value oracle calls:

$$O\left(\frac{r^2}{\epsilon^2}|E|\log(|E|)\left(\frac{\bar{d}+\underline{d}}{\bar{d}}\right)^2\right). \tag{3.36}$$

□

Now that we have the cost of a single iteration of the main loop of Algorithm 1, we can quantify the total cost of the algorithm:

Lemma 3.2.13. *Algorithm 1 requires $O\left(\frac{r^4}{\epsilon^3}|E|\log(|E|)\left(\frac{\bar{d}+\underline{d}}{\bar{d}}\right)^3\right)$ value oracle calls, and $O\left(\frac{r^2|E|}{\epsilon}\left(\frac{\bar{d}+\underline{d}}{\bar{d}}\right)\right)$ independence oracle calls.*

Proof. The total cost of the algorithm is simply the cost of a single iteration (Lemma 3.2.12) multiplied by the number of iterations ($\frac{1}{\delta}$), i.e.: $O\left(\frac{1}{\delta^3 r^2}|E|\log(E)\left(\frac{\bar{d}+\underline{d}}{\bar{d}}\right)\right)$ value oracle calls; and $O\left(\frac{|E|}{\delta}\right)$ independence oracle calls. Which with we can now reformulate in terms of ϵ. Recall that in Corollary 3.2.8 we set $\epsilon = 2r^2\frac{\bar{d}+\underline{d}}{\bar{d}}\delta$, hence $\delta = \frac{1}{2r^2}\frac{\bar{d}}{\bar{d}+\underline{d}}\epsilon$. Hence we have

$$O\left(\frac{r^4}{\epsilon^3}|E|\log(|E|)\left(\frac{\bar{d}+\underline{d}}{\bar{d}}\right)^3\right) \tag{3.37}$$

value oracle calls, and

$$O\left(\frac{r^2|E|}{\epsilon}\left(\frac{\bar{d}+\underline{d}}{\bar{d}}\right)\right) \tag{3.38}$$

independence oracle calls. □

These results hold both for the monotone and non-monotone cases, but in the monotone case we can dispense with the $\left(\frac{\bar{d}+\underline{d}}{\bar{d}}\right)$ terms noting that for a monotone function $\underline{d} = 0$.

3.2.3 Numerical Experiments

In this section we show several numerical experiments that illustrate how our algorithm outperforms the existing methods. The methods we compare against are the Euler-like approach by [32], and two simple Runge-Kutta methods of 3rd and 4th order.

Le us now describe the experiment setup. We will solve instances of the Submodular Welfare Problem (SWP) [96], which is equivalent to the Task Allocation Problem. In this problem we have a set of agents, each with their own submodular utility function, and a set of tasks. The goal is to find a set of non-overlapping allocation of tasks to agents that maximises the sum of the utilities. The problem can be formulated as a maximisation of a submodular function over a matroid. The submodular function is the sum of the utilities of the agents, and the matroid is a partition matroid that enforces that each task is allocated to no more than one agent.

To generate instances of the problems we used several utility functions for each agent, both monotone and non-monotone. The way we have constructed these functions is by composing a modular function with a concave function, which yields a submodular function. For each agent i and item j, we have defined a weight w_{ij} drawn uniformly at random between 0 and 1, and we normalise so that the sums of all the weights for each agent is 1. Then, we define agent's i utility as $f^i(S) \triangleq g\big(\sum_{j\in S} w_{ij}\big)$ where g is a concave function. We have used a variety of concave functions, each of them is shown in the corresponding figure to produce a variety of submodular functions. In each instance we solve a problem with 67 tasks and 17 agents.

To illustrate the performance of our algorithm against the other methods we have carried out experiments with different values for the stepsize δ. For each instance we compute a baseline solution by running each method with two orders of magnitude smaller δ than the smallest point plotted and we take the average among all the four methods. Then, we compare the result of each method to the baseline, by computing the absolute relative error given by $|\frac{F(\mathbf{y})-F(\mathbf{y}_{\text{baseline}})}{F(\mathbf{y}_{\text{baseline}})}|$.

The results of the experiments are shown in Figure 3.1. We can see three noteworthy features. The first is that for all methods that we used, as it would be expected, the error reduces monotonically with the step size. Further, in the classic RK methods we can observe how increasing the order of the integration scheme results in steeper slopes. The second is that first is that while Euler and RK methods show a constant slope of error reduction, our method shows markedly two clear slopes. An initial steep slope followed by a shallower one. The transition happens around before $\delta = 0.01$. To understand the reasons behind this, we need to consider that the only source of error in our algorithm comes from propagating too far a given $\tilde{\mathbf{v}}^*$ solution. That is, when we increase the coordinates beyond the point where the solution to the linear program that determines the maximal marginal improvement changes. Considering this, the initial steep slope is the phase where a reduction in δ removes error because the $\tilde{\mathbf{v}}^*$ would have been propagated beyond validity. The second shallow slope

phase, occurs when most of the steps happen within the boundaries of validity of $\tilde{\mathbf{v}}^*$, and hence, further reductions do not change significantly the solution. The third noteworthy feature is that our algorithm starts with about an order of magnitude smaller error. Since the $y'(t)$ is concave, the propagation of Euler and RK methods tend to overestimate the gradient used in the increment, more so than the solution than our method. We believe that this incurs in larger errors when δ is too large and hence our proposed method outperforms both the Euler and RK methods.

In summary, we can see that our method generates a solution that is orders of magnitude closer to the baseline solution with orders of magnitude less steps. This is the case with both monotone and non-monotone submodular functions, and for the modular linear function. In practice this means that we should aim to use the δ that lies close to the steep-to-shallow transition, because using a larger delta would incur in a relatively large error, while reducing it further would not bring significant benefits.

3.2.4 Discussion

In this section we have presented a modified continuous greedy algorithm that reduces the steps required to solve submodular maximization problems subject to a matroid constraint. The algorithm has an approximation of $1 - \frac{1}{e}$ for monotone submodular functions and $\frac{1}{e}$ for non-monotone submodular functions. Compared with the current $\frac{1}{e}$-approximation of Feldman et a. [32], our algorithm achieves a reduction in the computational burden of several orders of magnitude. The reduction in the computational expense is achieved by using a bespoke integration step that makes the algorithm more continuous-like, leveraging a local analytical solution of the continuous greedy process. This analytical solution remains valid in a small region around a given point, allowing us to use orders of magnitude larger stepsizes. We provide formal proofs of the approximation guarantees and running times, as well as abundant evidence of the speed-up in the form of numerical experiments for a variety of submodular functions. Indeed, the original measured continuous greedy of [32] was not meant to be efficient, but rather an instrument to prove the existence of $\frac{1}{e}$ polynomial time approximations, and therefore did not provide with an estimate of the complexity. From their results we deduce that the complexity of their algorithm is in the order of $O(n^6)$ in the multilinear oracle model, which would mean an $O(n^7)$ or $O(n^8)$ in the value oracle model. This contrasts with our algorithm's $O\left(\frac{r^4}{\epsilon^3} n \log(n) \left(\frac{\bar{d}+d}{\bar{d}}\right)^3\right)$ also in the value oracle model. Which implies a minimum speed-up of $O(n^2)$. But we can do much better than this, in the next section we adapt the ideas in the decreasing threshold of [6] to achieve a further $O(r^2)$ speedup. This results in the first $\frac{1}{e}$-approximation algorithm that is tailored for efficiency and can enable many new applications for which the original measured continuous greedy was impractical.

Before moving on to the next section, we would like to remark that the proofs presented here are based on a matroid polytope, but they can be adapted to work for a more general constraint class of down-closed solvable polytopes.

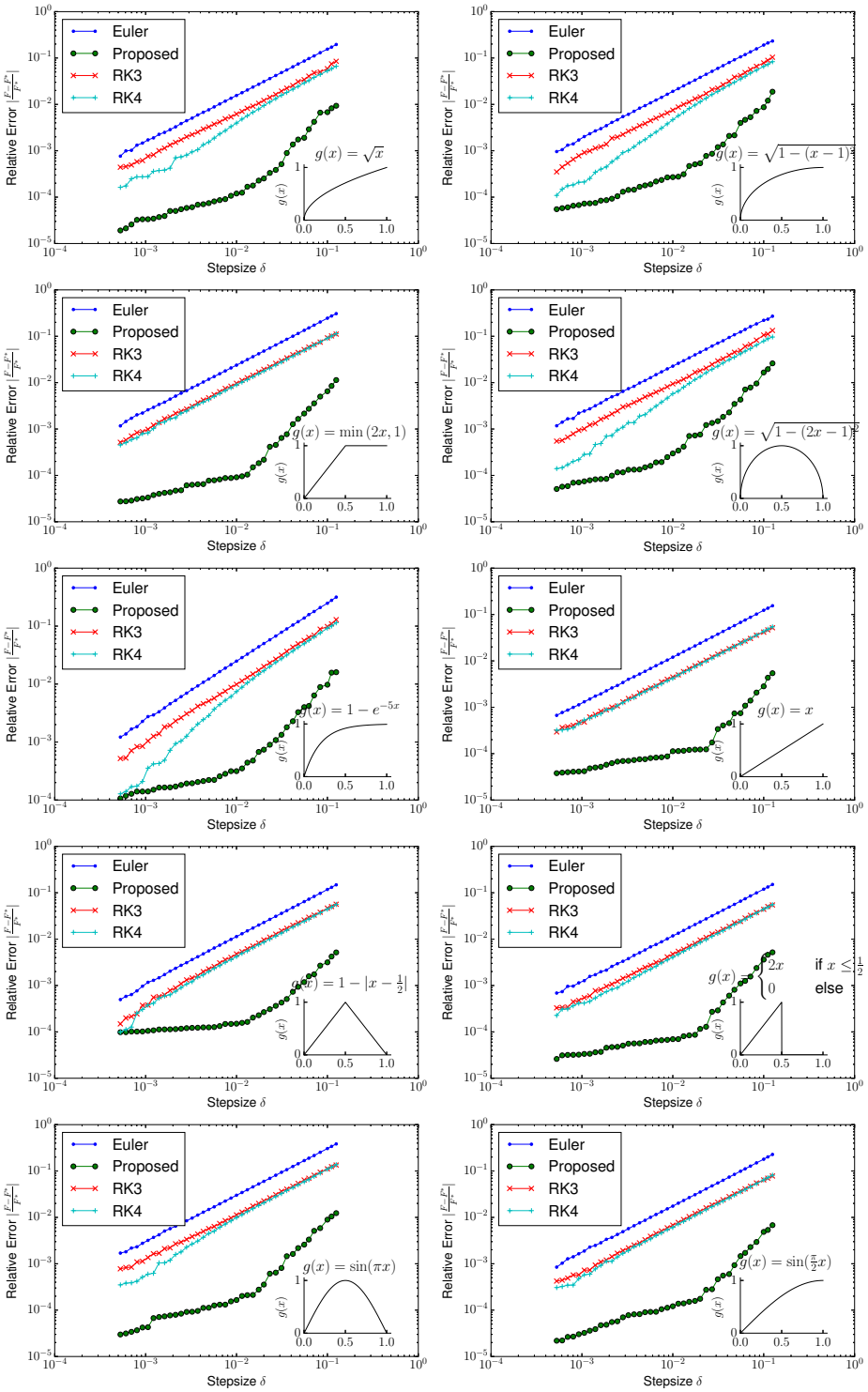

Figure 3.1: Comparison of the error reduction with stepsize δ for the different functions and methods. Euler refers to the integration scheme proposed by Feldman et al. in [32], while RK3 and RK4 refers to Runge-Kutta schemes of 3rd and 4th order.

3.3 Accelerated Measured Continuous Greedy

In the previous section we have presented a more a smoother measured continuous greedy algorithm. The algorithm we presented required $O\left(\frac{r^4}{\epsilon^3} n \log(n) \left(\frac{\bar{d}+\underline{d}}{\underline{d}}\right)^3\right)$ value oracle calls. In this algorithm from Lemma 3.2.6 we have that the improvement in each step is $F(\mathbf{y}(t+\delta)) - F(\mathbf{y}(t)) \geq \delta(e^{-t} f(\mathrm{OPT}) - F(\mathbf{y}(t)) - 2r^2(\bar{d}+\underline{d})\delta)$. That is, we incur in a loss that scales with $O(r^2)$, which forces us to use a smaller δ driving up the number of steps required and the accuracy of the sampling of the multilinear extension. In this section we use the ideas in the Decreasing-Threshold procedure of [6] to remove the $O(r^2)$ dependency of the loss in each step. Which enables us to reduce the number of steps accelerating further the algorithm. This accelerated measured greedy algorithm requires $O\left(\frac{nr^2}{\epsilon^4} \log^2(\frac{n}{\epsilon}) \left(\frac{\bar{d}+\underline{d}}{\underline{d}}\right)^2\right)$ value oracle calls. We believe it is the first practical $\frac{1}{e}$-approximation algorithm for the maximisation of a general non-negative submodular function subject to a matroid constraint. Let us now explain it in more detail.

3.3.1 Algorithm

Algorithm 2: Accelerated Measured Continuous Greedy

Input : $f : 2^E \to \mathbb{R}^+$, $\epsilon \in [0,1]$, $\mathcal{I} \subseteq 2^E$.
Output: A point $\mathbf{y} \in P(\mathcal{M})$, such that $F(\mathbf{y}) \geq (\frac{1}{e} - 2\epsilon) f(OPT)$.

```
// Initialisation
```
$\mathbf{y}(0) \leftarrow \mathbf{0}$
$\delta \leftarrow \epsilon$

```
// Main Loop
```
for $t = \{0, \delta, 2\delta, 3\delta, \ldots, 1-\delta\}$ **do**
  $B(t) \leftarrow$ `Decreasing-Threshold(`$f, \mathbf{y}(t), \epsilon, \delta, \mathcal{I}$`)`
  for $i \in E$ **do**
    if $i \in B(t)$ **then**
      $y_i(t+\delta) \leftarrow 1 + e^{-\delta}(y_i(t) - 1)$
    else
      $y_i(t+\delta) \leftarrow y_i(t)$

Return: $\mathbf{y}(1)$

The accelerated measured greedy is presented in Algorithm 2, it is essentially the same as the smooth measured continuous greedy in Algorithm 1 with a different procedure to find the elements whose coordinates are updated. Instead of finding the maximal marginal improvement set $\mathbf{v}^*$ we use the Decreasing-Threshold procedure that enabled Badanidiyuru and Vondrak [6] to achieve a efficient algorithm, $O(\frac{nr}{\epsilon^4} \log^2 \frac{n}{\epsilon})$, for the monotone case. In our previous algorithm we found $\mathbf{v}^*$ by estimating the value of each element at a given point, and then finding the maximum independent set. In contrast, the decreasing threshold procedure works like a greedy algorithm, it selects each item according to its marginal value computed considering the elements already selected, this saves

Algorithm 3: Decreasing-Threshold

Input : $f : 2^E \to \mathbb{R}^+$, $\mathbf{y} \in [0,1]^E$, $\epsilon \in [0,1]$, $\delta \in [0,1]$, $\mathcal{I} \subseteq 2^E$.
Output: A set $B \subseteq E$, such that $B \in \mathcal{I}$.

`// Initialisation`
$B \leftarrow \emptyset$;
$\bar{d} \leftarrow \max_{i \in E} f(i)$;
$\bar{y}' \leftarrow \max_{i \in E}(1 + e^{-\delta}(y_i - 1))$;

`// Main Loop`
for $(w = \bar{d};\ w \geq \epsilon \frac{\bar{d}}{r}(1 - \bar{y}');\ w \leftarrow w(1-\epsilon))$ **do**
  for $e \in E$ **do**
    $w_e(B, \mathbf{y}) \leftarrow \Delta F_e(\mathbf{y}(B, \delta))$,
    `// averaging` $O\left(\frac{r^2}{\epsilon^2}\left(\frac{\bar{d}+\underline{d}}{\bar{d}}\right)^2 \log(|E|)\right)$ `iid random samples.`
    if $w_e(B, \mathbf{y}) \geq w$ *and* $B + e \in \mathcal{I}$ **then**
      $B \leftarrow B + e$

Return: B

*Note that the notation $\mathbf{y}(B, \delta)$ means $y_i(B, \delta) = y_i(t)$ for $i \notin B$, and $y_i(B, \delta) = 1 + e^{-\delta}(y_i - 1)$ for $i \in B$.

us the losses from having to propagate the marginal values -i.e. the derivatives of the multilinear extension- in parallel without regard for what the other elements coordinate is. This saves us, an order of $O(r)$. On top of this, instead of using a standard greedy algorithm, it uses a Decreasing-Threshold variant procedure which further accelerates the running time of the greedy algorithm, achieving another $O(r)$ speed-up. The key insight from [6] is that, instead of selecting the maximal element each time - and consequently, having to loop each time over all elements r times- it keeps a geometrically decreasing threshold, and selects the first item that happens to be above threshold level. This way, while the classic greedy requires $O(nr)$ steps, it removes the dependency on the rank of the matroid $O(r)$, to achieve a running time of $O(\frac{n}{\epsilon}\log(\frac{n}{\epsilon}))$, where ϵ is the fraction by which the threshold is reduced in each iteration.

In their version of the Decreasing-Threshold Badanidiyuru and Vondrak [6] sampled the multilinear extension up to an additive and multiplicative error bound, which enabled them to use only $O(\frac{1}{\epsilon^2} r \log(n))$ samples. However, since our algorithm is meant to work with non-monotone functions, we may have negative marginal values, this increases the number of samples that we require for two reasons: first, it prevents us from using a multiplicative-additive error bound; and second the marginal values, when sampled, have inherently more spread. To quantify our error we use Hoeffding's concentration inequality, which forces us to use $O\big(\frac{r^2}{\epsilon^2}\big(\frac{\bar{d}+\underline{d}}{\bar{d}}\big)^2 \log(n)\big)$. (recall that $\underline{d}, \bar{d} \in \mathbb{R}^+$ are the absolute values of the minimum and maximum marginal values that the function f can take, i.e. $-\underline{d} \leq f_S(i) \leq \bar{d}$, for all $i \in E$ and $S \subseteq E$; if f were to be monotone we would have $\underline{d} = 0$.) The resulting Decreasing-Threshold procedure is presented in Algorithm 3, and it takes an additional $O(r\big(\frac{\bar{d}+\underline{d}}{\bar{d}}\big)^2)$ value oracle calls. Using only an additive bound instead of an additive and multiplicative introduces an extra $O(r)$ factor in the number of samples required per evaluation of the

multilinear extension. While we also incur the additional $O\big(\big(\frac{\bar{d}+\underline{d}}{\bar{d}}\big)^2\big)$ term to cope with the larger spread that is introduced by the possibility of negative marginal values. This results in an algorithm that finds an $\frac{1}{e}-\epsilon$ approximation with a number of value oracle calls of $O(\frac{nr^2}{\epsilon^4}\big(\frac{\bar{d}+\underline{d}}{\bar{d}}\big)^2\log^2(\frac{n}{\epsilon}))$. It is not as efficient as the algorithm for the monotone case, but we believe this constitutes the first practical $\frac{1}{e}-\epsilon$ approximation algorithm for general non-negative submodular function subject to a matroid constraint.

3.3.2 Algorithm Analysis

We split the analysis of the accelerated measured continuous greedy algorithm in three parts: first we show that the solution produced is feasible, i.e. $\mathbf{y}(1) \in P(\mathcal{M})$; then we prove the $\frac{1}{e}-\epsilon$ approximation; and finally we study its running time.

Feasibility

Theorem 3.3.1. *The accelerated measured continuous greedy algorithm produces a feasible fractional solution, i.e.,* $\mathbf{y}(1) \in P(\mathcal{M})$.

Proof. We follow the approach used by [32]. Indeed, this proof is identical to that of Lemma 3.2.1. We first define a vector $\mathbf{x}$ that coordinate wise upper-bounds $\mathbf{y}(1)$. Then, given that $P(\mathcal{M})$ is down-monotone, we only need to show that $\mathbf{x}$ is in $P(\mathcal{M})$ to show that $\mathbf{y}(1) \in P(\mathcal{M})$. Consider the vector $\mathbf{x} = \delta\sum_{l=0}^{\frac{1}{\delta}-1}\mathbf{1}_{B(\mathbf{y}(l\delta))}$. This is a coordinate-wise upper bound of $\mathbf{y}(1)$ because when $i \in B$, we have that $y_i(t+\delta) - y_i(t) = 1 + e^{-\delta}(y_i(t)-1) - y_i(t) = (1-e^{-\delta})(1-y_i(t)) \leq 1-e^{-\delta} \leq \delta$, for all $\delta \in [0,1]$; and when $i \notin B$ $y_i(t+\delta)-y_i(t) = 0$. We now show that $\mathbf{x}$ is in $P(\mathcal{M})$. First, note that by definition $\mathbf{1}_B \in P(\mathcal{M})$. Then, observe that $\mathbf{x}/\delta$ is the sum of $\frac{1}{\delta}$ points in $P(\mathcal{M})$, thus $(\mathbf{x}/\delta)/(1/\delta) = \mathbf{x}$ is a convex combination of points in $P(\mathcal{M})$, hence $\mathbf{x} \in P(\mathcal{M})$, and consequently $\mathbf{y}(1) \in P(\mathcal{M})$. □

Approximation Ratio

First we show the gain that the algorithm makes in a single step, and then use this to build a recurrence relation that yields the approximation ratio. But before we do that, we bound the error introduced by sampling:

Corollary 3.3.2. *Given a non-negative submodular function* $f : 2^E \to \mathbb{R}^+$*, and a point* $\mathbf{y} \in [0,1]^E$*; let* $\underline{d}, \bar{d} \in \mathbb{R}^+$ *be the minimum and maximum marginal values of* f*, such that* $-\underline{d} \leq f_S(j) \leq \bar{d}$ *for all* $S \subseteq E$ *and* $j \in E$*; let* $R_1, R_2, ..., R_m$ *be iid samples drawn from* $R(\mathbf{y})$*, let* $w_j(\mathbf{y}) = \frac{1}{m}\sum_{i=1}^{m} f_{R_i}(j)$*; and let* $f(OPT) = \max_{S\in\mathcal{I}} f(S)$*. Then,*

$$\Pr(|w_j(\mathbf{y}) - \Delta F_j(\mathbf{y})| \geq \beta f(OPT)) \leq 2e^{-2m\beta^2\left(\frac{\bar{d}}{\bar{d}+\underline{d}}\right)^2}.$$

Proof. Immediate application from the Hoeffding bound in Lemma 3.1.3, noting that $f(OPT) = \max_{S\in\mathcal{I}} f(S) \geq \max_{e\in E} f(\{e\}) = \bar{d}$.

□

Now we can present the improvement made by the algorithm in a single step:

Lemma 3.3.3. *Let OPT be an optimal solution. Given a fractional solution* $\mathbf{y}$*, the Decreasing-Threshold produces a set* B *such that, with* $\mathbf{y}' = \mathbf{1} + e^{-\mathbf{1}_B \delta} \odot (\mathbf{y} - \mathbf{1})$*, we have:*

$$F(\mathbf{y}') - F(\mathbf{y}) \geq (1 - e^{-\delta})\Big((1 - 4\epsilon)(1 - \bar{y}')f(OPT) - F(\mathbf{y}')\Big) \tag{3.39}$$

Proof. This proof follows closely the proof of Claim 4.1 in [6] but with several modifications to avoid assuming monotonicity, namely: the stopping threshold, the sampling error, and the increment bound. Assume that the Decreasing-Threshold procedure returns a sequence of r elements $B = \{b_1, b_2, \ldots, b_r\}$, indexed in the order in which they were chosen. Let $O = \{o_1, o_2, \ldots, o_r\}$ be an optimal solution indexed as per the exchange property of the matroids in Lemma 3.1.1 such that $\phi(b_i) = o_i$. Additionally, let B_i and O_i denote the first i elements of B and O respectively, i.e. B_i is the sequence in which the elements have been added to B in algorithm 3 up until the ith element was added. If the procedure returns fewer than r elements or the optimal solution contained fewer than r elements, formally we just add dummy elements with value 0, so that $|B| = r$ and $|O| = r$.

Now let us bound the marginal values of the elements selected by the Decreasing-Threshold procedure with respect to those in the optimal solution. Recall that $\mathbf{y}(S, \delta)$ is the notation that we use to refer to the point such that $y_k(S, \delta) = y_k$ if $k \notin S$ and $y_k(S, \delta) = 1 + e^{-\delta}(y_k - 1)$ if $k \in S$. When b_i is selected, let w be the current threshold, hence $w_{b_i}(\mathbf{y}(B_{i-1}, \delta)) \geq w$. At this point o_i is a candidate element, thus we have one of two situations depending on whether the procedure has finished: if the threshold has not dropped below $\frac{\epsilon}{r} d(1 - \bar{y}')$, the value of $w_{o_i}(\mathbf{y}(B_{i-1}, \delta))$ must be below the threshold in the previous iteration, i.e. $w_{o_i}(\mathbf{y}(B_{i-1}, \delta)) \leq \frac{w}{(1-\epsilon)}$ (otherwise it would have been chosen already); conversely, if the procedure has terminated, b_i is a dummy element with value 0, and the value of $w_{o_i}(\mathbf{y}(B_{i-1}, \delta))$ is below the stopping threshold, i.e. $w_{o_i}(\mathbf{y}(B_{i-1}, \delta)) \leq \epsilon \frac{\bar{d}}{r}(1 - \bar{y}')$. Consequently, we can relate the marginal value estimate of b_i to that of o_i:

$$w_{b_i}(\mathbf{y}(B_{i-1}, \delta)) \geq (1 - \epsilon) w_{o_i}(\mathbf{y}(B_{i-1}, \delta)) - \epsilon \frac{\bar{d}}{r}(1 - \bar{y}').$$

Note that when b_i is selected we have that $B_{i-1} + b_i \in \mathcal{I}$, and by the definition of o_i (i.e. the matroid exchange property) $B_{i-1} + o_i \in \mathcal{I}$.

Now we need to bound the error incurred by sampling. From Lemma 3.3.2 we can sample the marginal values up to an additive error of $\beta = \frac{\epsilon}{r} f(OPT)(1 - \bar{y}')$ by taking the average of $O\left(\frac{r^2}{\epsilon^2}\left(\frac{\bar{d}+d}{d}\right)^2 \log(|E|)\right)$ samples with high probability (i.e. with a bad estimate probability decreasing with $\frac{1}{|E|}$). (We do not have a term $O(\frac{1}{1-\bar{y}'})$ in the number of samples because our stopping time, $t = 1$, and the update rule enforce (see Lemma 3.2.2) that $(1 - \bar{y}') \geq \frac{1}{e}$). Thus, we can write:

$$w_{b_i}(\mathbf{y}(B_{i-1}, \delta)) \leq \Delta F_{b_i}(\mathbf{y}(B_{i-1}, \delta)) + \frac{\epsilon}{r} f(OPT)(1 - \bar{y}')$$
$$w_{o_i}(\mathbf{y}(B_{i-1}, \delta)) \geq \Delta F_{o_i}(\mathbf{y}(B_{i-1}, \delta)) - \frac{\epsilon}{r} f(OPT)(1 - \bar{y}')$$

which, with $\bar{d} \leq f(OPT)$, can be combined with the prior bound to yield:

$$\Delta F_{b_i}(\mathbf{y}(B_{i-1}, \delta)) \geq (1-\epsilon)\Delta F_{o_i}(\mathbf{y}(B_{i-1}, \delta)) - 3\frac{\epsilon}{r} f(OPT)(1-\bar{y}'). \tag{3.40}$$

Then, we can bound the improvement that the Decreasing-Threshold procedure obtains:

$$\begin{aligned}
&F(\mathbf{y}') - F(\mathbf{y}) \\
&= \sum_{i=1}^{r} (F(\mathbf{y}(B_i, \delta)) - F(\mathbf{y}(B_{i-1}, \delta)) \\
&= \sum_{i=1}^{r} (y'_{b_i} - y_{b_i}) \frac{\partial F}{\partial y_{b_i}}\bigg|_{\mathbf{y}=\mathbf{y}(B_{i-1},\delta)} \\
&= \sum_{i=1}^{r} (1-e^{-\delta})(1-y_{b_i}) \frac{\partial F}{\partial y_{b_i}}\bigg|_{\mathbf{y}=\mathbf{y}(B_{i-1},\delta)} \\
&= (1-e^{-\delta}) \sum_{i=1}^{r} \Delta F_{b_i}(\mathbf{y}(B_{i-1}, \delta)) \\
&\geq (1-e^{-\delta}) \sum_{i=1}^{r} \left((1-\epsilon)\Delta F_{o_i}(\mathbf{y}(B_{i-1}, \delta)) - 3\frac{\epsilon}{r} f(OPT)(1-\bar{y}') \right) \\
&= (1-e^{-\delta}) \left((1-\epsilon) \sum_{i=1}^{r} \left(\Delta F_{o_i}(\mathbf{y}(B_{i-1}, \delta)) \right) - 3\epsilon f(OPT)(1-\bar{y}') \right) \\
&\geq (1-e^{-\delta}) \left((1-\epsilon) \big(F(\mathbf{y}' \vee \mathbf{1}_{\mathrm{OPT}}) - F(\mathbf{y}') \big) - 3\epsilon f(OPT)(1-\bar{y}') \right) \\
&\geq (1-e^{-\delta}) \left((1-4\epsilon)(1-\bar{y}') f(OPT) - F(\mathbf{y}') \right).
\end{aligned}$$

The second equality comes from the the multilinearity of F. With the update step $y' = 1 + e^{-\delta}(y-1)$, we have that the increment is $y' - y = (1-e^{-\delta})(1-y)$, which gives the third inequality. The fourth equality is by definition of ΔF_e. The fifth inequality is by the bound in equation 3.40, the sixth by submodularity, and the last one by Lemma 3.1.2. □

Finally, we can use the above result to build a recurrence relation that yields the $\frac{1}{e} - \epsilon$ approximation ratio.

Theorem 3.3.4. *The accelerated measured continuous greedy algorithm returns a point $\mathbf{y}^* \in P(\mathcal{M})$, such that:*

$$F(\mathbf{y}^*) \geq \left(\frac{1}{e} - 2\epsilon \right) f(OPT). \tag{3.41}$$

Proof. From Lemma 3.3.3 we have that:

$$F(\mathbf{y}(t+\delta)) - F(\mathbf{y}(t)) \geq (1-e^{-\delta}) \left((1-4\epsilon)(1-\bar{y}(t+\delta)) f(OPT) - F(\mathbf{y}(t+\delta)) \right)$$

We can now use the bound on the value of the coordinates of $\mathbf{y}$ in Lemma 3.2.2, we have that $y_i(t) \leq 1 - e^{-t} \quad \forall i \in E$, hence we can write:

$$F(\mathbf{y}(t+\delta)) - F(\mathbf{y}(t)) \geq (1-e^{-\delta})\Bigg((1-4\epsilon)e^{-(\delta+t)}f(OPT) - F(\mathbf{y}(t+\delta))\Bigg). \quad (3.42)$$

Consider the recurrence relation $a(n+1) - a(n) = k_1(k_2 \exp(-(n+1)\epsilon) - a(n+1))$, which, with $a(0) = a_0$, has the following solution

$$a(n) = \frac{(\frac{1}{k_1+1})^n(a_0(-k_1 + e^{\epsilon} - 1) + k_1k_2) - k_1k_2e^{-n\epsilon}}{-k_1 + e^{\epsilon} - 1}. \quad (3.43)$$

This recurrence is equivalent to equation 3.3.2 if we set $k_1 = (1 - e^{-\delta})$, $k_2 = (1 - 4\epsilon)$, $n = \frac{t}{\delta}$, and $\delta = \epsilon$. Thus, substituting and simplifying assuming that $F(\mathbf{0}) \geq 0$ (due to the non-negativity of f), we have the following lower bound on the value of the solution $\mathbf{y}(t)$ for any time $t \in [0, 1]$:

$$F(\mathbf{y}(t)) \geq \frac{(1 - e^{-\epsilon})(1 - 4\epsilon)e^{\epsilon(-(\frac{t}{\epsilon}+1))}(e^{\epsilon(\frac{t}{\epsilon}+1)}(\frac{1}{2-e^{-\epsilon}})^{t/\epsilon} - e^{\epsilon})}{e^{-\epsilon} + e^{\epsilon} - 2} f(OPT).$$

So, when the algorithm ends at $t = 1$, we have:

$$F(\mathbf{y}(1)) \geq \frac{(1 - e^{-\epsilon})(1 - 4\epsilon)e^{\epsilon(-(\frac{1}{\epsilon}+1))}(e^{\epsilon(\frac{1}{\epsilon}+1)}(\frac{1}{2-e^{-\epsilon}})^{1/\epsilon} - e^{\epsilon})}{e^{-\epsilon} + e^{\epsilon} - 2} f(OPT).$$

We can find a more intuitive version of this bound by observing that, clearly, for $0 \leq \epsilon \leq 1$:

$$\frac{1}{e} - \frac{(1 - e^{-\epsilon})(1 - 4\epsilon)e^{\epsilon(-(\frac{1}{\epsilon}+1))}(e^{\epsilon(\frac{1}{\epsilon}+1)}(\frac{1}{2-e^{-\epsilon}})^{1/\epsilon} - e^{\epsilon})}{e^{-\epsilon} + e^{\epsilon} - 2} \leq \frac{5}{e}\epsilon \leq 2\epsilon \quad (3.44)$$

Hence:

$$F(\mathbf{y}(1)) \geq \left(\frac{1}{e} - 2\epsilon\right) f(OPT). \quad (3.45)$$

Finally, from Lemma 3.2.2 we have that $\mathbf{y}(1) \in P(\mathcal{M})$.

□

Running Time

We quantify the running time in terms of value oracle calls to the submodular function and matroid independence oracle calls. We analyse the running time of the algorithm in two steps: first we study the running time of the Decreasing-Threshold procedure, and then that of the the continuous greedy.

Lemma 3.3.5. *The Decreasing-Threshold procedure makes* $O\left(\frac{|E|r^2}{\epsilon^3}\log(|E|)\log(\frac{r}{\epsilon})\left(\frac{\bar{d}+d}{\bar{d}}\right)^2\right)$ *value oracle calls, and* $O(\frac{|E|}{\epsilon}\log\frac{r}{\epsilon})$ *independence oracle calls.*

Proof. First, the number of values that the threshold takes in the Decreasing-Threshold procedure to reach the stopping threshold is, considering that the term $(1 - \bar{y}) \geq \frac{1}{e}$ due to Lemma 3.2.2, $O(\frac{\log\frac{\epsilon}{r}}{\log(1-\epsilon)})$. Second, for each threshold

value, the algorithm performs $O(|E|)$ estimates of ΔF_e, and $O(|E|)$ calls to the independence oracle. Therefore, the number of independence oracle calls is the number of calls per threshold step multiplied by the number of threshold steps, i.e.:

$$O\left(\frac{|E|}{\epsilon}\log\frac{r}{\epsilon}\right), \tag{3.46}$$

which has been simplified noting that $\frac{\log\frac{\epsilon}{r}}{\log 1-\epsilon} \leq \frac{1}{\epsilon}\log(\frac{r}{\epsilon})$.
Now, each estimate of ΔF_e requires $O\left(\frac{r^2}{\epsilon^2}\left(\frac{\bar{d}+d}{\bar{d}}\right)^2\log(|E|)\right)$ samples. Hence, we can conclude that the number of value oracle calls is

$$O\left(\frac{|E|r^2}{\epsilon^3}\log(|E|)\log\left(\frac{r}{\epsilon}\right)\left(\frac{\bar{d}+\underline{d}}{\bar{d}}\right)^2\right).$$

□

We can now quantify the running time of the whole algorithm.

Theorem 3.3.6. *The accelerated measured continuous greedy algorithm makes* $O\left(\frac{|E|r^2}{\epsilon^4}\log(|E|)\log(\frac{r}{\epsilon})\left(\frac{\bar{d}+d}{\bar{d}}\right)^2\right)$ *value oracle calls, and* $O\left(\frac{|E|}{\epsilon^2}\log\frac{r}{\epsilon}\right)$ *independence oracle calls.*

Proof. Considering that the number of steps of the procedure, with $\delta = \epsilon$, is $\frac{1}{\epsilon}$. The number of calls to boths oracles is simply the number of calls by the Decreasing-Threshold procedure (Algorithm 3) in Lemma 4.2.9 multiplied by the number of steps in the continuous greedy algorithm (Algorithm 2). □

3.3.3 Discussion and Future Work

In this section we have presented a $\frac{1}{e}-\epsilon$-approximation algorithm for general non-negative submodular function maximisation that requires $O(\frac{nr^2}{\epsilon^4}\left(\frac{\bar{d}+d}{\bar{d}}\right)^2\log^2(\frac{n}{\epsilon}))$ value oracle calls. This is the fastest $\frac{1}{e}$-approximation algorithm currently available, which enables the use of general (non-monotone) matroid-constrained submodular maximisation for many applications for which existing algorithms were implausibly slow. We think this is of significance, even beyond Task Allocation problems, because there has been a recent surge of interest for applying submodular maximisation in fields where large problem instances are paramount, such as Machine Learning [10, 64, 72], particularly in the field of summarisation where non-monotone submodular functions are natural [23, 94]. However, the ability to solve problems with non-monotone submodular functions comes at a cost. Our algorithm trades-off computational cost in exchange for a broader set of objective functions (non-monotone submodular). This means that it is slower than the one presented for the monotone case in [6] by $O\left(r\left(\frac{\bar{d}+d}{\bar{d}}\right)^2\right)$. This is because when the function is assumed to be non-monotone, we can only sample the marginal values of the multilinear extension up to an additive bound, compared with an additive and multiplicative in [6]. If this idea is feasible, then we could reduce the additional value oracle calls required to achieve an algorithm with the same running time as [6].

A future avenue of research would be to combine our work with the very interesting results in [13], where an efficient algorithm is proposed to allow the trade-off of value oracle calls and matroid independence calls for non-negative monotone submodular functions, to enable query trade-off for general non-negative submodular functions. Another interesting path is to combine the more continuous-like measured continuous greedy update step that we present here with the acceleration techniques for strong submodular functions presented in [101] to produce an adaptive step algorithm. This way, in each step we could use a large δ that extended to the boundary of the region of validity of the set B, instead of taking a δ that is small enough to satisfy the worst case. Another obvious improvement on the algorithms presented here would be to combine the ideas from the Lazy Greedy Algorithm [71] to adaptively change the decrement of the threshold in the Decreasing-Threshold procedure. Finally, in the next chapter we build on the theoretical foundations laid in the algorithms of this chapter to enable efficient constant-factor approximation algorithms for decentralised Task Allocation.

Chapter 4

Decentralised Submodular Task Allocatiton

4.1 Introduction

In this chapter we adapt the algorithms presented in the previous chapter to solve the task allocation problem. Our algorithm is the first decentralised task allocation algorithm that finds a guaranteed approximation with *non-monotone* submodular utilities, and it is also the first that gives an approximation factor for *monotone* submodular that is optimal, i.e. an algorithm with a better factor would require an exponential number of calls to the value oracle.

Multi Robot Systems (MRS) are gaining increasing popularity both in the research community and in industry. A fundamental problem that underpins the effective coordinate operation of these systems is Task Allocation. The MRS should be able to find an answer quickly, reliably, and effectively to the question: "given the robots available in the system and the tasks that ought to be carried out, what is the best allocation of these tasks among us?". In general, the task allocation problem is NP-Hard in all but its simplest incarnations [35], [52]. There exist tools in the Operations Research and Combinatorial Optimisation literature that enable the centralised solution of instances of practical interest. However, the centralised solution of the problem involves having to communicate all the agent and environment data to a centralised entity. This may not be the most appropriate approach for some scenarios. Obvious examples of these situations are when relying on a central entity removes resilience by introducing a single point of failure; or when, in some communication environments, the bandwidth to communicate all the information to the central entity from every agent is not available. Thus, there is a need for decentralised solution strategies that only rely on each robot knowing their own utility function and communicating with neighbours rather than with a central planning entity. Let us start our introduction by stating more formally the problem that our algorithm solves and discussing some of its features.

Problem Definition

Adapting the classical definition, from [27], we can define the general Task Allocation Problem as follows:

Given a set of tasks $\mathcal{T}$, a set of agents $\mathcal{A}$, and a function for each agent $a \in \mathcal{A}$ specifying the utility of completing each subset of tasks $f_a : 2^{\mathcal{T}} \to \mathbb{R}^+$, find a non-overlapping allocation, $\mathcal{S}^* \in \mathcal{A}^{\mathcal{T}}$, that minimises/maximises a global objective function $\mathcal{J} : \mathcal{A}^{\mathcal{T}} \to \mathbb{R}^+$.

In its full generality this problem can be reduced, for example, to the well studied Set Packing Problem which, unfortunately, not only happens to be NP-Hard, but it is also inapproximable within a constant factor [80]. If we define the sense of the optimisation as a maximisation, and the objective function, $\mathcal{J}$, as the sum of the utilities of each robot the problem is reduced to the Social Welfare Problem [78], also known as the Winner Determination Problem in combinatorial auctions [100]. Even in this restricted case, both problems still remain NP-Hard and known to be inapproximable within a constant factor. We can, however, further restrict the utility functions to satisfy the submodularity condition, and the problem can be reduced to a maximisation of a submodular function subject to a partition matroid constraint. This problem is still NP-Hard, nevertheles, as we have seen in Chapter 3, there are polynomial time algorithms that can guarantee a constant-factor approximation. In this chapter we will leverage the results that we have presented in the previous chapter to design decentralised, constant-factor approximation algorithms for the task allocation problem where the agent's utilities are monotone or non-monotone submodular.

Let us now place our problem in context. A widely accepted taxonomy that maps the nature of each problem to a well known combinatorial problem was introduced in [35] and was extended to support task dependencies in [52]. These two works provide a map of the task allocation problem space. In the taxonomy of [35] the algorithm we present in this chapter aims to tackle problems with: both Single-Task robots (ST) and Multi-Task robots (MT); Single-Robot tasks (SR); and both Instantaneous Assignment (IA) and Time Extended Assignment (TA). With respect to the task dependencies taxonomy in [52] our algorithm can accommodate: No Dependencies (ND) and In-Schedule Dependencies (ID). Following general definition from [27], let us now define the particular instance of the task allocation problem that our algorithm solves:

Given a set of tasks $\mathcal{T}$, a set of agents $\mathcal{A}$, and a non-negative submodular function for each agent $a \in \mathcal{A}$ specifying the utility of completing each subset of tasks $f_a : 2^{\mathcal{T}} \to \mathbb{R}^+$, find a non-overlapping allocation, $\mathcal{S}^* \in \mathcal{A}^{\mathcal{T}}$, that maximises a global objective function $\mathcal{F} : \mathcal{A}^{\mathcal{T}} \to \mathbb{R}^+$ defined as $\mathcal{F}(S) = \sum_{a \in \mathcal{A}} f_a(S_a)$.

We focus on solving this problem in the decentralised setting, that is, when each agent only has access to its own utility function and does not have any knowledge of the functions corresponding to other agents. In that sense, we refer to the local objective functions of each agent as being *local* or *private*. We now review some of the previous approaches to solve task allocation problems in a decentralised setting.

Decentralised Solution Approaches

The earliest decentralised task allocation algorithms were proposed as solutions to the Assignment Problem. This is a slightly different problem to the version we have presented above, instead of having a different number of tasks and agents. In the assignment problem we are restricted to have n tasks to assign to n agents, that is: each agent must have a task and each task must have an agent. The assignment problem can be solved optimally in polynomial time, the classic approach is the Hungarian Algorithm [14, 56]. Due to the good tractability of these problems there have been a number of algorithms proposed that do guarantee optimal performance in a decentralised setup. The first distributed task allocation strategy was that proposed by [8] where an auction algorithm based on the idea of a shared memory model was presented. However, the shared memory model required a topology of the networked system that is not always achievable in real scenarios. To address this issue in [105] Zavlanos et al give an algorithm is to handle a networked system in which agents interact with its neighbours, rather than having access to a shared database.

When we remove the constraint on allocating a task for each agent, and an agent for each task, we need to consider how to value bundles of tasks that are assigned to the same agent. The simplest model is adding the values of each task, this is a modular (i.e. linear) set function. In this case, the problem becomes the optimisation of a modular function subject to a matroid constraint which, as we have seen, is solved optimally using the greedy algorithm. Several works have implemented versions of the greedy algorithm in a decentralised setting. In [19] the authors present the Consensus-Based Auction Algorithm CBAA, this algorithm uses the concept of maximum consensus to distribute a series of single task auctions across the network which, in effect, implements a distributed version of the greedy algorithm. Liu and Shell [66] present another approach to implement a decentralised greedy algorithm based on on local task swaps. While in [73] Moon et al present an application of a qualitatively similar algorithm for UAV task allocation in a dynamic environment alongside an account of its performance in real flight.

The problem becomes significantly harder when the utility functions are not linear, such as when inter-task dependencies appear, because agents must evaluate bundles of tasks instead of individual tasks. Indeed, we have seen that when the utilities are general functions and the global objective is a simple sum, the problem is NP-Hard. As a consequence, most of the algorithms presented in this case are heuristic in their nature, see for example [27] for a classic review on market based algorithms, or [104] for a recent survey. A great breakthrough occurred when Choi et al [19] presented a decentralised algorithm for the maximisation of *monotone* submodular utilities that had rigorous approximation guarantees: the Consensus Based Bundle Algorithm (CBBA), spurring a lot of interest in the research community. This was achieved through a decentralisation of the greedy algorithm, that brought the approximation guarantee, $\frac{1}{2}$, of the classic result of Nemhauser et al. [77] to the decentralised Task Allocation domain. However, this only applies to the non-negative *monotone* submodular functions. This can be limiting because it cannot model situations of practical interest where non-monotonicity is a feature. For example, in a multi-robot surveillance mission, if a robot is assigned too many targets to track it is possible that it ends up spending its time traveling between targets and not gathering

enough useful information at the targets' locations. Therefore, adding tasks to a robot's assignment could, indeed, reduce the utility obtained. This ubiquitous situation cannot be modelled by monotone utility functions, and therefore, CBBA could perform arbitrarily poorly in this case. In fact, we are not aware of any constant factor approximation algorithm that can solve the task allocation problem with the maximisation of *non-monotone* submodular utilities in a decentralised setting. The algorithms that we present in this chapter address this issue by solving problems with both monotone and non-monotone submodular utility functions.

Contributions

In this chapter we extend the state of the art in decentralised task allocation in two ways. First, we present the first decentralised task allocation algorithm with approximation guarantees for general non-negative submodular utility functions, more specifically we guarantee $\frac{1}{e}$ w.r.t. the optimal. Second, we improve the approximation ratio of Choi et al. [19] for monotone non-negative submodular utility functions from $\frac{1}{2}$ to $1 - \frac{1}{e}$, which is asymptotically optimal.

Chapter Structure

The chapter has two main parts. First we give a detailed exposition of the centralised version of our main algorithm and prove its approximation guarantees. Then we present the decentralised version, and show how it is equivalent to the centralised version.

4.2 Equivalent Centralised Algorithm

In this section we give a detailed account of how the centralised algorithm works and why it works. To design our algorithm we will leverage a relaxation-rounding approach that we have developed in Chapter 3. This strategy is composed of two steps. The first step is to relax the problem by allowing variables to attain a fractional value. This continuous optimisation problem, called the relaxed problem or relaxation, can be solved either optimally or with a certain approximation. The second step is to bring back to the discrete domain the fractional solution of the relaxed problem without losing the stablished guarantees on the relaxation solution. This second step is often referred to as rounding. In this chapter we solve the relaxation using an algorithm based on the accelerated measured continuous greedy from chapter 3. Our relaxation domain is the polytope of the partition matroid, which encapsulates the task allocation problem, and we use the multilinear extension to evaluate fractional allocations. To round the solution we leverage the special structure of the partition matroid using randomised rounding.

4.2.1 Preliminary Concepts

Before we proceed to the description of the algorithm we need to review some key concepts and establish some notation definitions used in its development.

A function $f : 2^{\mathcal{T}} \to \mathbb{R}^+$ on a set $\mathcal{T}$ is said to be *submodular* if, given subsets $Y, X \subseteq \mathcal{T}$ satisfying $X \subseteq Y$ and $|X| \leq |Y|$, then $f(X \cup \{x\}) - f(X) \geq$

$f(Y \cup \{x\}) - f(Y), \forall x \in \mathcal{T}$. Similarly, it is said to be *monotone* if for every two sets X, Y such that $X \subseteq Y \subseteq \mathcal{T}$ then $f(X) \leq f(Y)$.

Let us now introduce some notational quirks. Recall from the previous chapter that $\underline{d}, \bar{d} \in \mathbb{R}^+$ are the absolute values of the minimum and maximum marginal values that the agent's utility functions, f_a, can take i.e.: $-\underline{d} \leq f_a(S + \{i\}) - f(S) \leq \bar{d}$, for all $i \in \mathcal{T}$, $S \subseteq \mathcal{T}$ and $a \in \mathcal{A}$. Naturally, for monotone utility functions we have that $\underline{d} = 0$. For clarity during our exposition of the algorithm and its analysis, we use the subscript notation to describe allocations of tasks to agents. For example, $S_a \subseteq \mathcal{T}$ represents the subset of the tasks of agent $a \in \mathcal{A}$. While S denotes the set of allocations of all agents, that is $S = \{S_a | a \in \mathcal{A}\}$. Note that an allocation is feasible if for each pair of agents $a, b \in \mathcal{A}$, such that $a \neq b$, we have that $S_a \cap S_b = \emptyset$. We usually denote an allocation by a capital letter: e.g. B_a is the set of tasks allocated to agent a, and B represents the set of all agents' allocations. A special case is OPT, which denotes an optimal allocation, and OPT_a denotes the set of tasks allocated to agent $a \in \mathcal{A}$ in an optimal solution. Finally, we refer to the number of agents by $n_{\mathcal{A}}$, i.e. $n_{\mathcal{A}} = |\mathcal{A}|$ and the number of tasks by $n_{\mathcal{T}}$, i.e. $n_{\mathcal{T}} = |\mathcal{T}|$. We will refer to the number of randomly rounded solutions that we will produce by m.

Let us describe the fractional allocation domain and its objective function in more detail. The fractional allocation (or relaxation) contains a value between 0 and 1 for each task and agent. Thus, this continuous allocation space is in $[0,1]^{\mathcal{A}\times\mathcal{T}}$. We use $\mathbf{y} \in [0,1]^{\mathcal{A}\times\mathcal{T}}$ to refer to a fractional allocation. Similar to our discrete allocation notation, we use $\mathbf{y}_a \in [0,1]^{\mathcal{T}}$ to refer to the fractional allocation of agent $a \in \mathcal{A}$, and we use $y_{aj} \in [0,1]$ to denote the fraction of task $j \in \mathcal{T}$ allocated to agent $a \in \mathcal{A}$. Now, we constrain this space to prevent the allocation of more than one unit of each task among the agents. These are the packing constraints of the partition matroid, whose polytope is defined by:

$$\mathcal{K} = \{\mathbf{y}_a \in [0,1]^{\mathcal{A}\times\mathcal{T}} | \sum_{a\in\mathcal{A}} y_{aj} \leq 1 \, \forall j \in \mathcal{T}\}. \tag{4.1}$$

To evaluate fractional allocations we will use the multilinear extension introduced in section 3.1.1. In our problem, each agent $a \in \mathcal{A}$ has its own utility function $f_a : \mathcal{T} \to \mathbb{R}^+$ and we define its multilinear extension, $F_a : [0,1]^{\mathcal{T}} \to \mathbb{R}^+$, as:

$$F_a(\mathbf{y}) \triangleq \mathbb{E}[f_a(R(\mathbf{y}_a))] = \sum_{\mathcal{S}\subseteq E} f_a(\mathcal{S}) \prod_{i\in\mathcal{S}} y_{ai} \prod_{j\notin\mathcal{S}} (1 - y_{aj}) \tag{4.2}$$

where $R(\mathbf{y}_a)$ is a random set that contains each task $j \in \mathcal{T}$ independently with probability y_{aj}.

Given our global objective function $\mathcal{F}(S) = \sum_{a\in\mathcal{A}} f_a(S_a)$ we denote its multilinear extension by $F : [0,1]^{\mathcal{A}\times\mathcal{T}} \to \mathbb{R}^+$ and is defined as:

$$F(\mathbf{y}) = \sum_{a\in\mathcal{A}} F_a(\mathbf{y}_a). \tag{4.3}$$

Finally, we will use $\Delta F_{aj}(\mathbf{y})$ to denote the marginal value of task $j \in \mathcal{T}$ to agent $a \in \mathcal{A}$, given the fraction allocation $\mathbf{y}_a \in [0,1]^{\mathcal{T}}$, defined as:

$$\Delta F_{aj}(\mathbf{y}_a) = F_a(\mathbf{y}_a \vee \mathbf{1}_j) - F_a(\mathbf{y}_a). \tag{4.4}$$

4.2.2 Algorithm Definition

We can now describe the structure of our algorithm. The main algorithm is presented in Algorithm 4, and it has two main steps: relaxation, and rounding. To solve the relaxation we use a variant of the accelerated smoothed measured continuous greedy algorithm introduced in section 3.3. Once we have the relaxation result, the special structure of the partition matroid enables us to round the fractional solution using a simple randomised rounding procedure. Let us now describe the relaxation and the rounding algorithms in more detail.

Algorithm 4: Centralised Task Allocation

Input : $\mathcal{T}, \mathcal{A}, f_a : 2^{\mathcal{T}} \rightarrow \mathbb{R}^+ \, \forall a \in \mathcal{A}$, $\epsilon \in [0, 1]$, and $m \in \mathbb{Z}^+$
Output: $B_a^* \subseteq \mathcal{T}$ for all $a \in \mathcal{A}$, s.t. $B_i \cap B_j = \emptyset$ for all $j, i \in \mathcal{A}$ with $i \neq j$
$\mathbf{y}^* \leftarrow$ `Solve-Relaxation(`$\mathcal{T}, \mathcal{A}, f_a \forall a \in \mathcal{A}$, ϵ`)`
$B^* \leftarrow$`Round-Relaxation(`$\mathbf{y}^*, \mathcal{T}, \mathcal{A}, f_a \forall a \in \mathcal{A}$, m`)`
Return: B_a^* for all $a \in \mathcal{A}$

Relaxation Solution

As we have seen in chapter 3, the smooth measured greedy algorithm finds an approximate solution to the relaxation, $\mathbf{y}^* \in \mathcal{K}$, by integrating between $t = 0$ to $t = 1$ the differential equation $\frac{d\mathbf{y}(t)}{dt} = (1 - \mathbf{y}(t)) \odot \text{argmax}_{v \in \mathcal{K}} \sum_{a \in \mathcal{A}} \sum_{j \in \mathcal{T}} \Delta F_{aj}(\mathbf{y}_a) v_{aj}$ (here, $\odot$ represents element by element multiplication). This integration is approximated by incrementing at a rate of $\frac{dy(t)}{dt} = 1 - y(t)$ in small steps the coordinates of the elements (in this case task-agent pairs) that provide the maximum marginal value. The integration algorithm is presented in Algorithm 5, and it is essentially the same as Algorithm 5. Basically, this algorithm in each iteration finds the maximal improvement set and then increments it, see Section 3.2 where we provide a detailed rationale behind it. However, in this chapter we propose a different algorithm, called Threshold-Greedy, to find the maximal improvement set, i.e. the task-agent pairs with the maximum marginal value. The Threshold-Greedy algorithm is presented in Algorithm 6, and differs from what we presented in Section 3.2 in that it is designed specifically such that all computations that involve information from more than one agent are performed using only *max* operations to make allocation decisions. This enables us to decentralise it efficiently using max-consensus protocols. Let us explain the Threshold-Greedy algorithm in more detail. It is called threshold-greedy because it is essentially a greedy algorithm that, in addition to selecting the task with the highest marginal value, includes any task whose marginal value is within a factor $(1 - \epsilon)$ of the highest. Let us now describe it in more detail.

The Threshold-Greedy algorithm starts by initialising the remaining task set $\mathcal{R}$, the allocation sets B_a for each agent $a \in \mathcal{A}$, the maximum value of a propagated coordinate $\bar{y}'$, and the maximum marginal value $\bar{d}$. Then, the algorithm proceeds to the *while* loop. This is the main loop of the algorithm and is constituted by two main parts: the pre-allocation stage, and the allocation stage. In the pre-allocation stage each agent finds the task which has the maximum marginal value and pre-allocates it alongside all the tasks whose marginal value is within a factor of $(1 - \epsilon)$ of the maximum. This is done is two steps. In

the first step the agent, say a, finds which task, given its current allocation B_a has the maximum marginal value. The agent a then sets the value of the best task as its threshold, and initialises its pre-allocation bundle B'_a to contain the tasks previously allocated to it, B_a, and the best task. In the second step the agent scans through all the remaining tasks, and adds them if their marginal value, given the current pre-allocation set B'_a, is within a factor of $(1-\epsilon)$ of the maximum computed in the first step. The order in which the agent scans through the remaining tasks is not important, but each time that the agent checks a task it must compute its marginal value considering the tasks in the pre-allocated bundle B'_a at that point. Crucially, each agents' pre-allocation solution only considers its own allocation set and utility function, this facilitates the decentralisation of the algorithm.

The second part of the main loop is where tasks are allocated. Each iteration the algorithm finds which is the maximum threshold among the agents and adds the tasks whose marginal values are within a factor of $(1-\epsilon)$. This is done is two steps. In the first step the algorithm finds which is the maximum threshold among the agents, w^*, we refer to this as the *global threshold*. In the second step the algorithm loops over all the tasks that are unallocated and allocates them to the agent with the highest marginal value if this value is above $(1-\epsilon)w^*$, adding them the to agent's allocation set B_a. Once the algorithm has checked all the pre-allocated tasks and updated the remaining task set, $\mathcal{R}$, a new iteration starts. The algorithm ends when either all the tasks have been allocated, or the global threshold value has dropped below $\epsilon\frac{\bar{d}}{n_{\mathcal{T}}}(1-\bar{y}')$. We call this value the stopping threshold. The tasks that remain unallocated, if any, have a value below the stopping threshold and can therefore safely be ignored because they do not make a significant contribution to the solution value. The result of the algorithm is a group of non-overlapping sets B_a for each agent $a \in \mathcal{A}$ that contain the tasks allocated to each agent. These sets are used to propagate their coordinates in the measured continuous greedy integration (Algorithm 5). Critically, observe that all the exchange of information between agents is based around computing the maximum among magnitudes, which can readily be decentralised efficiently using max-consensus. Let us now describe the rounding procedure.

Rounding

Once we have a solution to the relaxation, the rounding procedure takes the fractional allocation and produces a feasible discrete solution that maintains the approximation ratio. The procedure is presented in Algorithm 7 and it is based on the randomised rounding strategy for the Submodular Welfare Problem of [15, 96]. The essence of the algorithm is to generate random allocations where each task $j \in \mathcal{T}$ is allocated independently to agent $a \in \mathcal{A}$ with probability y_{aj}. This makes the expected value of a random allocation have the same value as the multilinear extension of the relaxation result. Therefore, by generating m random allocations and then selecting the one with the highest value among them, we can guarantee that the best discrete allocation will have a value close or above the relaxation result with a probability that converges to 1 exponentially with m. Note that by the design of the relaxation procedure we do not necessarily have that $\sum_{a\in\mathcal{A}} y_{aj} = 1$, this implies that some tasks will not

Algorithm 5: Centralised Solve-Relaxation

Input : $f_a : 2^{\mathcal{T}} \to \mathbb{R}^+ \, \forall a \in \mathcal{A}$, $\epsilon \in [0, 1]$.
Output: A fractional allocation $\mathbf{y} \in \mathcal{K}$.

```
// Initialisation
```
$\mathbf{y}(0) \leftarrow \mathbf{0}$
$\delta \leftarrow \epsilon$

```
// Main Loop
```
for $t = \{0, \delta, 2\delta, 3\delta, \ldots, 1 - \delta\}$ **do**
  $B_a \underset{\forall a \in \mathcal{A}}{\leftarrow}$ `Threshold-Greedy`$(\mathcal{T}, \mathcal{A}, f, \mathbf{y}(t), \epsilon, \delta)$
  for $a \in \mathcal{A}$ **do**
    if $j \in B_a$ **then**
      $y_{aj}(t + \delta) \leftarrow 1 + e^{-\delta}(y_{aj}(t) - 1)$
    else
      $y_{aj}(t + \delta) \leftarrow y_{aj}(t)$

Return: $\mathbf{y}(1)$

necessarily be allocated. To emphasise this fact, in the `RandomRound` subroutine (Algorithm 8) we break the rounding procedure of each task in two steps, first, we decide if a task will be allocated at all with probability $\sum_{a \in \mathcal{A}} y_{aj}$. Then, once a task has been decided to be allocated, we randomly chose among the agents with probability proportional to their value in the relaxation, i.e. $\frac{y_{aj}}{\sum_{a \in \mathcal{A}} y_{aj}}$.

4.2.3 Algorithm Analysis

Let us now analyse the algorithm, we first show that the algorithm produces a feasible solution to the relaxation. Then, we prove the approximation ratios of the relaxation for the monotone and non-monotone cases. And, finally, we show that these ratios are preserved in the rounding procedure.

Relaxation Feasibility

Let us show that the algorithm produces a feasible solution.

Theorem 4.2.1. *The accelerated measured continuous greedy algorithm produces a feasible fractional solution, i.e.,* $\mathbf{y}(1) \in \mathcal{K}$.

Proof. We follow the approach used by [32]. Indeed, this proof is identical to that of Lemma 3.2.1 and of Lemma 4.2.1. We first define a vector $\mathbf{x}$ that coordinate wise upper-bounds $\mathbf{y}(1)$. Then, given that $\mathcal{K}$ is down-monotone, we only need to show that $\mathbf{x}$ is in $\mathcal{K}$ to show that $\mathbf{y}(1) \in \mathcal{K}$. Let $\mathbf{x}$ be the vector s.t. $\mathbf{x} = \delta \sum_{l=0}^{\frac{1}{\delta}-1} \mathbf{1}_{B(\mathbf{y}(l\delta))}$. This is a coordinate-wise upper bound of $\mathbf{y}(1)$ because for all $a \in \mathcal{A}$ when $j \in B_a$, we have that $y_{aj}(t + \delta) - y_{aj}(t) \triangleq 1 + e^{-\delta}(y_{aj}(t) - 1) - y_{aj}(t) = (1 - e^{-\delta})(1 - y_{aj}(t)) \leq 1 - e^{-\delta} \leq \delta$, for all $\delta \in [0, 1]$; and when $j \notin B_a$ $y_{aj}(t + \delta) - y_{aj}(t) = 0$. We now show that $\mathbf{x}$ is in $\mathcal{K}$. First, note that because, by the design of the algorithm, there can only be at most one

Algorithm 6: Centralised Threshold-Greedy Algorithm

Input : $\mathcal{T}$, $\mathcal{A}$, $f_a : 2^{\mathcal{T}} \to \mathbb{R}^+ \, \forall a \in \mathcal{A}$, $\mathbf{y} \in [0,1]^{\mathcal{A}\times\mathcal{T}}$, $\epsilon \in [0,1]$, $\delta \in [0,1]$.
Output: Sets B_a for all $a \in \mathcal{A}$, such that $\sum_{a\in\mathcal{A}} |B_a \cap \{j\}| \leq 1$ for all $j \in \mathcal{T}$.

`// Initialisation`
$B_a \leftarrow \emptyset$, for all $a \in \mathcal{A}$
$\mathcal{R} \leftarrow \mathcal{T}$
$\bar{d} \leftarrow \max_{\substack{a\in\mathcal{A}\\ j\in\mathcal{T}}} f_a(j)$
$\bar{y}' \leftarrow \max_{\substack{a\in\mathcal{A}\\ j\in\mathcal{T}}} (1 + e^{-\delta}(y_{aj} - 1))$
$w^* \leftarrow \bar{d}$
`// main loop`
while $\mathcal{R} \neq \emptyset$ *and* $w^* \geq \epsilon\frac{\bar{d}}{n_{\mathcal{T}}}(1-\bar{y}')$ **do**
  for $a \in \mathcal{A}$ **do**
    `// agent's loop initialisation`
    for $j \in \mathcal{R}$ **do**
      $w_{aj} \leftarrow \Delta F_{aj}(\mathbf{y}_a(B_a, \delta))$
      `// averaging` $O\left(\frac{n_{\mathcal{T}}^2}{\epsilon^2}\left(\frac{\bar{d}+d}{\bar{d}}\right)^2 \log(n_{\mathcal{T}})\right)$ `iid random samples.`
    $j^* \leftarrow \operatorname{argmax}_{j\in\mathcal{R}} w_{aj}$
    $w_a^* \leftarrow w_{aj^*}$
    $B'_a \leftarrow B_a + j^*$
    `// agent's pre-allocation loop`
    for $j \in \mathcal{R} - j^*$ **do**
      $w_{aj} \leftarrow \Delta F_{aj}(\mathbf{y}_a(B'_a, \delta))$
      `// averaging` $O\left(\frac{n_{\mathcal{T}}^2}{\epsilon^2}\left(\frac{\bar{d}+d}{\bar{d}}\right)^2 \log(n_{\mathcal{T}})\right)$ `iid random samples.`
      if $w_{aj} \geq (1-\epsilon)w_a^*$ **then**
        $B'_a \leftarrow B'_a + j$
  $\mathcal{W}^* \leftarrow \emptyset$
  for $j \in \mathcal{R}$ **do**
    $w_{a^*j} \leftarrow \max_{a\in\mathcal{A}} w_{aj}$
    $\mathcal{W}^* \leftarrow \mathcal{W}^* + w_{a^*j}$
  `// task allocation loop`
  $w^* \leftarrow \max_{w_{a^*j}\in\mathcal{W}^*} w_{a^*j}$
  if $w^* \geq \epsilon\frac{\bar{d}}{n_{\mathcal{T}}}(1-\bar{y}')$ **then**
    for $w_{a^*j} \in \mathcal{W}^*$ **do**
      if $w_{a^*j} \geq (1-\epsilon)w^*$ **then**
        $B_{a^*} \leftarrow B_{a^*} + j$
        $\mathcal{R} \leftarrow \mathcal{R} - j$
Return: B_a for all $a \in \mathcal{A}$
*The notation $\mathbf{y}_a(B_a, \delta)$ means $y_{aj}(B_a, \delta) = y_{aj}(t)$ for $j \notin B_{aj}$, and $y_{aj}(B_a, \delta) = 1 + e^{-\delta}(y_{aj} - 1)$ for $j \in B_{aj}$.

Algorithm 7: Centralised Round-Relaxation

Input : $\mathbf{y} \in \mathcal{K}$, $\mathcal{T}$, $\mathcal{A}$, $f_a : 2^{\mathcal{T}} \to \mathbb{R}^+ \, \forall a \in \mathcal{A}$, m
Output: Sets B_a^* for all $a \in \mathcal{A}$, s.t. $\sum_{a \in \mathcal{A}} |B_a^* \cap \{j\}| \leq 1$ for all $j \in \mathcal{T}$.

for $i \in \{1, 2, ...m\}$ **do**
  $B^i \leftarrow$ `RandomRound(`$\mathbf{y}$, $\mathcal{A}$, $\mathcal{T}$`)`
$i^* \leftarrow \underset{i}{\text{argmax}} \sum_{a \in \mathcal{A}} f_a(B_a^i)$
$B^* \leftarrow B^{i^*}$

Return: B_a^* for all $a \in \mathcal{A}$

Algorithm 8: Random-Round

Input : $\mathcal{T}$, $\mathcal{A}$, $\mathbf{y} \in \mathcal{K}$
Output: Sets B_a^* for all $a \in \mathcal{A}$, s.t. $\sum_{a \in \mathcal{A}} |B_a^* \cap \{j\}| \leq 1$ for all $j \in \mathcal{T}$.

for $a \in \mathcal{A}$ **do**
  $B_a \leftarrow \emptyset$

`// main loop`
for $j \in \mathcal{T}$ **do**
  $\alpha \leftarrow$ `Uniform-Random(0,1)`
  if $\alpha \leq \sum_{a \in \mathcal{A}} y_{aj}$ **then**
    choose $a \in \mathcal{A}$, at random with probability $\frac{y_{aj}}{\sum_{a \in \mathcal{A}} y_{aj}}$
    $B_a \leftarrow B_a + j$

Return: B_a for all $a \in \mathcal{A}$

agent per task, hence $\mathbf{1}_B \in \mathcal{K}$. Then, observe that $\mathbf{x}/\delta$ is the sum of $\frac{1}{\delta}$ points in $\mathcal{K}$, thus $(\mathbf{x}/\delta)/(1/\delta) = \mathbf{x}$ is a convex combination of points in $\mathcal{K}$, hence $\mathbf{x} \in \mathcal{K}$, and consequently $\mathbf{y}(1) \in \mathcal{K}$. □

Now let us establish a bound to the coordinates of $\mathbf{y}(t)$ that will become useful later on in the analysis of the approximation ratio.

Lemma 4.2.2. *At time $0 \leq t \leq 1$, we have that:*

$$y_{aj}(t) \leq 1 - e^{-t}, \quad \textit{for all} \quad a \in \mathcal{A},\, j \in \mathcal{T}. \tag{4.5}$$

Proof. Consider the recurrence $g(n+1) = 1 + e^{-\delta}(1 - g(n))$, with $g(0) = 0$. This recurrence has the following solution: $g(n) = 1 - e^{-n\delta}$. Now, in our algorithm t is incremented linearly, so the number of iteration, n, and t are related by $t = n\delta$. At t, all the coordinates of $\mathbf{y}$ either, they stay constant, i.e. $y_{aj}(t+\delta) = y_{aj}(t)$, when $j \notin B_a$, or increase, i.e. $y_{aj}(t+\delta) = 1 + e^{-\delta}(y_{aj}(t) - 1)$, when $j \in B_a$. Hence, given that $g(n)$ is non-decreasing, the recurrence g is an upper bound because it corresponds to incrementing in each and every iteration. Consequently, at t, $y(t) \leq g(\frac{t}{\delta}) = 1 - e^{-t}$. □

Relaxation Approximation Ratios

Now we prove the approximation ratios of the algorithm. First we show the gain made in a single step, and then we build a recurrence relation to show the approximation ratios for the monotone and non-monotone cases.

Lemma 4.2.3. *Let OPT be an optimal solution. Given a fractional solution $\mathbf{y}$, algorithm 6 produces a collection of sets B_a for all $a \in \mathcal{A}$, such that, with $\mathbf{y}'_a = \mathbf{1} + e^{-\mathbf{1}_{B_a}\delta} \odot (\mathbf{y}_a - \mathbf{1})$ for all $a \in \mathcal{A}$, we have:*

$$F(\mathbf{y}') - F(\mathbf{y}) \geq (1 - e^{-\delta})\Big((1-\epsilon)\big(F(\mathbf{y}' \vee \mathbf{1}_{OPT}) - F(\mathbf{y}')\big) - 3\epsilon\mathcal{F}(OPT)(1 - \bar{y}')\Big) \tag{4.6}$$

Proof. It is very similar to the one we present for the general matroid constraint in Lemma 3.3.3, (which in turn follows closely the proof of Claim 4.1 in [6]) but takes into account the different logic to update the threshold and the tasks that are allocated. The proof sketch is as follows: first, we find a lower bound on the marginal value of an allocated task with respect to the threshold; second, we find an upper bound of the marginal value of that task in the optimal solution with respect to the threshold; third, we combine this two relations to find a lower bound on the marginal value of the allocated task with respect to its marginal value in the optimal solution; finally, given this lower bound, we derive the result, i.e. a lower bound on the increment in terms of the optimal solution.

For simplicity, we will refer to the tasks by the order in which they were added to agents' sets, hence $j_1, j_2, ..., j_{n_\mathcal{T}}$ is the ordered sequence of tasks, where $j_i \in \mathcal{T}$ is the ith task that was added to an agent's set. Let $a_1, a_2, ..., a_{n_\mathcal{T}}$ be the agent sequence in order in which they received a task, that is, $a_i \in \mathcal{A}$ is the agent that received the task j_i, which was the ith allocated. Naturally, if

say the third and the fifth tasks are allocated to the same agent, we have that a_3 and a_5 are the same agent. Now, let $o_1, o_2, ..., o_{n_\mathcal{T}}$ be the agents that have tasks $j_1, j_2, ..., j_{n_\mathcal{T}}$ in an optimal solution. If the number of tasks allocated by the algorithm, or the number of tasks in the optimal solution, were fewer than $n_\mathcal{T}$, as part of the formal analysis we add dummy tasks and agents with value 0. Finally, let $B_{a_i}(i)$ denote the set of tasks allocated to agent a_i before the ith task, j_i, was allocated to it, and similarly let $B'_{a_i}(i)$ denote the set of tasks pre-allocated to agent a_i before the ith task was pre-allocated to it.

Recall that $\mathbf{y}_a(S, \delta)$ is the notation that we use to refer to the fractional allocation of agent $a \in \mathcal{A}$ such that $y_k(S, \delta) = y_k$ if $k \notin S$ and $y_k(S, \delta) = 1 + e^{-\delta}(y_k - 1)$ if $k \in S$. By submodularity, for any task $j \in \mathcal{T}$ and agent $a \in \mathcal{A}$, and any two sets $S_1 \subseteq S_2 \subseteq \mathcal{T}$ we have that: $\Delta F_{aj}(\mathbf{y}_a(S_1, \delta)) \geq \Delta F_{aj}(\mathbf{y}_a(S_2, \delta))$. Therefore, since the algorithm design enforces that $B_{a_i}(i) \subseteq B'_{a_i}(i)$, we have that:

$$\Delta F_{a_i j_i}(\mathbf{y}_{a_i}(B_{a_i}(i), \delta)) \geq \Delta F_{a_i j_i}(\mathbf{y}_{a_i}(B'_{a_i}(i), \delta)). \tag{4.7}$$

Let us now account for the error incurred by sampling. From Lemma 3.3.2 we can sample w_e to an additive error of $\frac{\epsilon}{r}\mathcal{F}(OPT)(1-\bar{y}')$ by taking the average of $O\left(\frac{n_\mathcal{T}^2}{\epsilon^2}\left(\frac{\bar{d}+d}{\bar{d}}\right)^2 \log(n_\mathcal{T})\right)$ samples with high probability, i.e. with a bad estimate probability decreasing with $\frac{1}{n_\mathcal{T}}$. (We do not have a term $O(\frac{1}{1-\bar{y}'})$ in the number of samples because our stopping time, $t = 1$, and the update rule enforce (see Lemma 4.2.2) that $(1 - \bar{y}') \geq \frac{1}{e}$).

We can now combine the sampling errors with the relations that the algorithm's design enforces with respect to the threshold w^*, to obtain a lower bound on the marginal values of task j_i at the point when it was allocated. First we look at the bound on the marginal values that the task j_i added to the agent that won it, a_i, we have that:

$$w^*(1-\epsilon) \leq w_{j_i a_i} \leq \Delta F_{a_i j_i}(\mathbf{y}_{a_i}(B'_{a_i}(i), \delta)) + \frac{\epsilon}{n_\mathcal{T}}\mathcal{F}(OPT)(1-\bar{y}'). \tag{4.8}$$

The first inequality is because j_i was allocated to a_i, and the second due to the sampling error.

Now, we can bound the marginal value, given the current allocation, that task j_i adds to the agent o_i, i.e. the agent that got it in the optimal solution. We have that:

$$w^* \geq w_{j_i o_i} \geq \Delta F_{o_i j_i}(\mathbf{y}_{o_i}(B_{o_i}(i), \delta)) - \frac{\epsilon}{n_\mathcal{T}}\mathcal{F}(OPT)(1-\bar{y}'). \tag{4.9}$$

The first inequality is because w^* is the maximum marginal value given the allocation state in the previous iteration, and the second is because of the error introduced by sampling.

At this point we can establish a lower bound on the marginal value of task j_i in agent a_i in terms of the marginal value of j_i in the agent that had it in the optimal solution o_i. When combining the two results above in equations 4.8 and 4.9 by means of w^*, we need to subtract an additional $\epsilon\frac{\bar{d}}{n_\mathcal{T}}(1-\bar{y}')$ term from the right side to account for the case when j_i is a dummy task with $w_{a_i j_i} = 0$, and $w_{o_i j_i} \leq \epsilon\frac{\bar{d}}{n_\mathcal{T}}(1-\bar{y}')$. Hence, we have

$$w_{a_i j_i} \geq (1-\epsilon) w_{o_i j_i} - \epsilon\frac{\bar{d}}{n_\mathcal{T}}(1-\bar{y}'), \tag{4.10}$$

and using the right hand side of equations 4.8 and 4.9

$$\Delta F_{a_i j_i}(\mathbf{y}_{a_i}(B'_{a_i}(i),\delta)) \geq (1-\epsilon)\Delta F_{o_i j_i}(\mathbf{y}_{o_i}(B_{o_i}(i),\delta)) - 3\frac{\epsilon}{n_{\mathcal{T}}}\mathcal{F}(OPT)(1-\bar{y}'),$$

which with Equation 4.7 above, yields the desired lower bound:

$$\Delta F_{a_i}(\mathbf{y}_{a_i}(B_{a_i}(i),\delta)) \geq (1-\epsilon)\Delta F_{o_i}(\mathbf{y}_{o_i}(B_{o_i}(i),\delta)) - 3\frac{\epsilon}{n_{\mathcal{T}}}\mathcal{F}(OPT)(1-\bar{y}'). \tag{4.11}$$

Note that here we have used the fact that $\bar{d} \leq \mathcal{F}(OPT)$. Now, based on this, we can derive a lower bound on the value gained between $\mathbf{y}$ and $\mathbf{y}'$:

$$\begin{aligned}
&F(\mathbf{y}') - F(\mathbf{y}) \\
&= \sum_{a\in\mathcal{A}} (F_a(\mathbf{y}_a(B_a,\delta)) - F_a(\mathbf{y}_a)) && \text{definition of } F \\
&= \sum_{i=1}^{n_{\mathcal{T}}} (F_{a_i}(\mathbf{y}_{a_i}(B_{a_i}(i)+j_i,\delta)) - F_{a_i}(\mathbf{y}_{a_i}(B_a(i),\delta)) && \text{telescoping sum} \\
&= \sum_{i=1}^{n_{\mathcal{T}}} \left((y'_{a_i j_i} - y_{a_i j_i}) \frac{\partial F_{a_i}}{\partial y_{a_i j_i}}\bigg|_{\mathbf{y}=\mathbf{y}_{a_i}(B_{a_i}(i),\delta)} \right) && \text{multilinearity} \\
&= \sum_{i=1}^{n_{\mathcal{T}}} (1-e^{-\delta})(1-y_{a_i j_i}) \frac{\partial F_{a_i}}{\partial y_{a_i j_i}}\bigg|_{\mathbf{y}=\mathbf{y}_{a_i}(B_{a_i}(i),\delta)} && \text{update step: } y' = 1 + e^{-\delta}(y-1) \\
&= (1-e^{-\delta}) \sum_{i=1}^{n_{\mathcal{T}}} \Delta F_{a_i j_i}(\mathbf{y}(B_{a_i},\delta)) && \text{definition of } \Delta F \\
&\geq (1-e^{-\delta}) \sum_{i=1}^{n_{\mathcal{T}}} \left((1-\epsilon)\Delta F_{o_i}(\mathbf{y}_{o_i}(B_{o_i}(i),\delta)) - 3\frac{\epsilon}{n_{\mathcal{T}}}\mathcal{F}(OPT)(1-\bar{y}') \right) && \text{equation 4.11} \\
&= (1-e^{-\delta}) \left((1-\epsilon) \sum_{i=1}^{n_{\mathcal{T}}} \left(\Delta F_{o_i}(\mathbf{y}(B_i,\delta)) \right) - 3\epsilon\mathcal{F}(OPT)(1-\bar{y}') \right) && \text{re-arranging} \\
&\geq (1-e^{-\delta}) \left((1-\epsilon)\big(F(\mathbf{y}' \vee \mathbf{1}_{\mathrm{OPT}}) - F(\mathbf{y}')\big) - 3\epsilon\mathcal{F}(OPT)(1-\bar{y}') \right) && \text{submodularity}
\end{aligned}$$

□

Now we can use the bound on the gain made in a single iteration to derive the approximation ratio for the non-monotone case:

Lemma 4.2.4. *If $f_a : 2^{\mathcal{T}}$ for all $a \in \mathcal{A}$ are non-negative submodular functions, then Algorithm 5 returns a point $\mathbf{y}^* \in \mathcal{K}$, such that:*

$$F(\mathbf{y}^*) \geq \left(\frac{1}{e} - 2\epsilon\right)\mathcal{F}(OPT). \tag{4.12}$$

Proof. The gain in a single step is:

$$
\begin{aligned}
&F(\mathbf{y}(t+\delta)) - F(\mathbf{y}(t)) \\
&\geq (1-e^{-\delta})\Big((1-\epsilon)\big(F(\mathbf{y}(t+\delta)) \vee \mathbf{1}_{\mathrm{OPT}}) - F(\mathbf{y}(t+\delta))\big)\Big) - 3\epsilon\mathcal{F}(OPT)(1-\bar{y}(t+\delta))\Big) \\
&\geq (1-e^{-\delta})\Big((1-4\epsilon)(1-\bar{y}(t+\delta))\mathcal{F}(OPT) - F(\mathbf{y}(t+\delta))\Big).
\end{aligned}
$$

The first inequality is by Lemma 4.2.3, and the second by Lemma 3.1.2. We can see that this result is identical to Theorem 3.3.3. Therefore, given that Algorithm 2 is equivalent to Algorithm 5, the proof of the ratio is exactly the same as in Theorem 3.3.4 and we shall not repeat it here. □

Similarly, we can now prove the ratio for the monotone case.

Lemma 4.2.5. *If $f_a : 2^{\mathcal{T}} \to \mathbb{R}^+$ for all $a \in \mathcal{A}$ are non-negative monotone submodular functions, then Algorithm 5 returns a point $\mathbf{y}^* \in \mathcal{K}$, such that:*

$$
F(\mathbf{y}^*) \geq \left(1 - \frac{1}{e} - 3\epsilon\right)\mathcal{F}(OPT). \tag{4.13}
$$

Proof. The gain in a single step is:

$$
\begin{aligned}
&F(\mathbf{y}(t+\delta)) - F(\mathbf{y}(t)) \\
&\geq (1-e^{-\delta})\Big((1-\epsilon)\big(F(\mathbf{y}(t+\delta) \vee \mathbf{1}_{\mathrm{OPT}}) - F(\mathbf{y}(t+\delta))\big) - 3\epsilon\mathcal{F}(OPT)(1-\bar{y}(t+\delta))\Big) \\
&\geq (1-e^{-\delta})\Big((1-4\epsilon)\mathcal{F}(OPT) - F(\mathbf{y}(t+\delta))\Big).
\end{aligned}
$$

The first inequality is by Lemma 4.2.3, and the second by monotoninicy. Therefore, the improvement in a given step is:

$$
F(\mathbf{y}(t+\delta)) - F(\mathbf{y}(t)) \geq (1-e^{-\delta})\Big((1-4\epsilon)\mathcal{F}(OPT) - F(\mathbf{y}')\Big). \tag{4.14}
$$

Now consider the recurrence relation, $a(n+1) - a(n) = k_1(k_2 - a(n+1))$, with $a(0) = a_0$, which has the following solution:

$$
a(n) = a_0\left(\frac{1}{k_1+1}\right)^n + \left(1 - \left(\frac{1}{k_1+1}\right)^n\right)k_2. \tag{4.15}
$$

This recurrence is equivalent to Equation 4.14 if we set $k_1 = 1 - e^{-\delta}$, $k_2 = (1-4\epsilon)\mathcal{F}(OPT)$, $n = \frac{t}{\delta}$, and $a_0 = F(\mathbf{0})$. Therefore, we can write:

$$
F(\mathbf{y}(t)) \geq F(\mathbf{0})\left(\frac{1}{2-e^{-\delta}}\right)^{t/\delta} + \left(1 + \left(\frac{1}{2-e^{-\delta}}\right)^{t/\delta}\right)(1-4\epsilon)\mathcal{F}(OPT), \tag{4.16}
$$

which noting that $F(\mathbf{0}) \geq 0$, by non-negativity of f, yields:

$$F(\mathbf{y}(t)) \geq \left(1 + \left(\frac{1}{2 - e^{-\delta}}\right)^{t/\delta}\right)(1 - 4\epsilon)\mathcal{F}(OPT). \tag{4.17}$$

So with $\delta = \epsilon$, when the algorithm finishes at $t = 1$, we have:

$$F(\mathbf{y}(1)) \geq \left(1 + \left(\frac{1}{2 - e^{-\epsilon}}\right)^{1/\epsilon}\right)(1 - 4\epsilon)\mathcal{F}(OPT). \tag{4.18}$$

This we can be simplified noting that, for $0 \leq \epsilon \leq 1$, the following holds:

$$\left(1 + \left(\frac{1}{2 - e^{-\epsilon}}\right)^{1/\epsilon}\right)(1 - 4\epsilon) \geq 1 - \frac{1}{e} - \left(\frac{3}{e} - 4\right)\epsilon \geq 1 - \frac{1}{e} - 3\epsilon. \tag{4.19}$$

Therefore

$$F(\mathbf{y}(1)) \geq \left(1 - \frac{1}{e} - 3\epsilon\right)\mathcal{F}(OPT). \tag{4.20}$$

Finally, our solution is feasible because from Theorem 4.2.1, $\mathbf{y}(1) \in \mathcal{K}$.

□

Rounding Analysis

Let us now analyse the rounding procedure. Its aim is to find a discrete solution given a relaxation value (i.e. a fractional solution) without loosing too much on the appximation ratios. The procedure is presented in Algorithm 7 and it is based on the randomised rounding strategy for the Submodular Welfare Problem of [15, 96]. The essence of the algorithm is to generate allocation sets whose expected value is the same as the value of the relaxation. Then, if we generate enough sets we can guarantee with the desired probability that their sample mean will be close to the true mean. Therefore the best randomly generated allocation with the maximum value will also be close (or above) to the value of the relaxation, which we have already bounded with respect to the optimal. Let us now proceed to the formal analysis. First, we show that the expected value of a randomly rounded set using algorithm 7 is precisely the value of the multilinear extension of the solution. Finally, we combine this result with a Chernoff bound to show that the discrete solution found is close to the guarantee of the relaxation.

Lemma 4.2.6. *Given a point $\mathbf{y} \in \mathcal{K}$ and the sets of agents $\mathcal{A}$ and tasks $\mathcal{T}$, the subroutine* `RandomRound` *(Algorithm 8), generates a feasible random allocation B such that:*

$$\mathbb{E}[\mathcal{F}(B)] = F(\mathbf{y}). \tag{4.21}$$

Proof. The random allocation sets that each agent $a \in \mathcal{A}$ receives are not independent of each other, but the tasks $j \in \mathcal{T}$ in each set B_a do appear independently with probability y_{aj}. This is because each task is allocated separately, and

the probability that a task $j \in \mathcal{T}$ is allocated to a given agent $a \in \mathcal{A}$ is the probability that the task is allocated, $\sum_{a \in \mathcal{A}} y_{aj}$, multiplied by the probability of it being subsequently allocated to agent a, $\frac{y_{aj}}{\sum_{a \in \mathcal{A}} y_{aj}}$, that is: $\left(\sum_{a \in \mathcal{A}} y_{aj}\right) \frac{y_{aj}}{\sum_{a \in \mathcal{A}} y_{aj}} = y_{aj}$.

Now, recall that the definition of the multilinear extension stated that $F_a(\mathbf{y}_a) \triangleq \mathbb{E}[f_a(R(\mathbf{y}_a))]$, where $R(\mathbf{y}_a)$ is defined as the random set that contains each task in $\mathcal{T}$ independently with probability y_{aj}. Hence we have that $\mathbb{E}[f_a(B_a)] = \mathbb{E}[f_a(R(\mathbf{y}_a))] = F_a(\mathbf{y}_a)$. Thus, $\sum_{a \in \mathcal{A}} \mathbb{E}[f_a(B_a)] = \sum_{a \in \mathcal{A}} F_a(\mathbf{y}_a) = F(\mathbf{y})$. Therefore, by linearity of the expectation $\sum_{a \in \mathcal{A}} \mathbb{E}[f_a(B_a)] = \mathbb{E}[\sum_{a \in \mathcal{A}} f_a(B_a)] = \mathbb{E}[\mathcal{F}(B)] = F(\mathbf{y})$.

Finally, the solution is feasible because by the design of the `RandomRound` subroutine each task is allocated to only one agent, if any. □

To bound the value of the best solution found we need the following Chernoff bound:

Lemma 4.2.7. *(Theorem 1.1 in [28]) Let $X_1, ..., X_m$ be independent random variables such that for each i, $X_i \in [0, 1]$. Let $X = \sum_{i=1}^{m} X_i$. Then*

$$\Pr[X \geq (1 + \delta)\mathbb{E}[X]] \leq e^{-\frac{\delta^2}{3}\mathbb{E}[X]},$$
$$\Pr[X \leq (1 - \delta)\mathbb{E}[X]] \leq e^{-\frac{\delta^2}{2}\mathbb{E}[X]}.$$

We can now show that our rounded solution keeps the guarantees established for the relaxation with a probability that we can set arbitrarily high. Indeed, this probability converges exponentially to one with the number of solutions randomly rounded (recall that m is the number of rounded solutions).

Theorem 4.2.8. *The Centralised Task Allocation Algorithm, Algorithm 4, returns a feasible set B^* such that, with probability at least $1 - e^{-\frac{\epsilon^2}{2}(\frac{1}{e} - 2\epsilon)m}$, the following bounds hold:*

- *If $f_a : 2^{\mathcal{T}} \to \mathbb{R}^+$ for all $a \in \mathcal{A}$ are non-negative monotone submodular functions then:*
$$\mathcal{F}(B^*) \geq \left(1 - \frac{1}{e} - 4\epsilon\right) \mathcal{F}(OPT). \tag{4.22}$$
- *If $f_a : 2^{\mathcal{T}} \to \mathbb{R}^+$ for all $a \in \mathcal{A}$ are general (possibly non-monotone) non-negative submodular functions then:*
$$\mathcal{F}(B^*) \geq \left(\frac{1}{e} - 3\epsilon\right) \mathcal{F}(OPT). \tag{4.23}$$

Proof. Let $\mathbf{y}^*$ be the solution of the relaxation provided by Algorithm 5. Now we show that the randomised rounding procedure in Algorithm 7 returns a feasible allocation solution B^* that satisfies the above bounds.

We apply the Chernoff bound from Lemma 4.2.7 to bound the lower tail of the sample mean value of the random allocations. Let $B^1, B^2, ..., B^m$ be the random allocations resulting from the repeated execution of the `RandomRound` subroutine (Algorithm 8). From Lemma 4.2.6 we have that for each i, $\mathbb{E}[\mathcal{F}(B^i)] = F(\mathbf{y}^*)$. Let $\tilde{F}(\mathbf{y}^*) = \frac{1}{m}\sum_{i=1}^{m}\mathcal{F}(B^i)$. Now, in order to apply Lemma 4.2.7, we need to normalise by $\mathcal{F}(OPT)$ to ensure that the values lie in $[0,1]$. Therefore we have:

$$\Pr[\tilde{F}(\mathbf{y}^*) \leq (1-\epsilon)F(\mathbf{y}^*)] \leq e^{-\frac{\epsilon^2}{2}\frac{F(\mathbf{y}^*)}{\mathcal{F}(OPT)}m}. \tag{4.24}$$

Now, from Lemma 4.2.4 for a general non-negative submodular function we have that $F(\mathbf{y}^*) \geq \left(\frac{1}{e} - 2\epsilon\right)\mathcal{F}(OPT)$. Hence, we can write:

$$\Pr[\tilde{F}(\mathbf{y}^*) \leq (1-\epsilon)F(\mathbf{y}^*)] \leq e^{-\frac{\epsilon^2}{2}\left(\frac{1}{e}-2\epsilon\right)m}. \tag{4.25}$$

Therefore, we have with probability at least $1 - e^{-\frac{\epsilon^2}{2}\left(\frac{1}{e}-2\epsilon\right)m}$ that:

$$\tilde{F}(\mathbf{y}^*) \geq (1-\epsilon)F(\mathbf{y}^*). \tag{4.26}$$

And since $\tilde{F}(\mathbf{y}^*) = \frac{1}{m}\sum_{i=1}^{m}\mathcal{F}(B^i) = \frac{1}{m}\sum_{i=1}^{m}\sum_{a\in\mathcal{A}} f_a(B_a^i) \leq \max_i \sum_{a\in\mathcal{A}} f_a(B_a^i) = \mathcal{F}(B^{i^*})$, we have that the allocation returned, B^{i^*}, satisfies that:

$$\mathcal{F}(B^{i^*}) \geq (1-\epsilon)F(\mathbf{y}^*). \tag{4.27}$$

At this point we use the relaxation approximation ratios to bound the value of the rounded solution. First we look at the monotone case, and then at the non-monotone. If f_a for all $a \in \mathcal{A}$ are non-negative monotone submodular functions, from Lemma 4.2.5 and equation 4.27, we have that:

$$\mathcal{F}(B^{i^*}) \geq (1-\epsilon)\left(1 - \frac{1}{e} - 3\epsilon\right)\mathcal{F}(OPT) \geq \left(1 - \frac{1}{e} - 4\epsilon\right)\mathcal{F}(OPT). \tag{4.28}$$

Now if f_a for all $a \in \mathcal{A}$ are general (possibly non-monotone) non-negative submodular functions, from Lemma 4.2.4 and equation 4.27, we have that:

$$\mathcal{F}(B^{i^*}) \geq (1-\epsilon)\left(\frac{1}{e} - 2\epsilon\right)\mathcal{F}(OPT) \geq \left(\frac{1}{e} - 3\epsilon\right)\mathcal{F}(OPT). \tag{4.29}$$

Finally, the solution is feasible by the design of the `RandomRound` subroutine, as stated by Lemma 4.2.6.

□

Hence, if we set $m = O(\frac{1}{\epsilon^2})$ we can make the probability of a good solution as large as desired, e.g. setting $m = \frac{100}{\epsilon^2}$ would make the probability of a good solution greater than 0.999998 for $0 < \epsilon \leq 0.05$.

We would like to make a clarifying remark on the bounds that relate the value of the rounded (discrete) solution, and the relaxation solution. In a classical Mixed Integer Program algorithm, the strategy is to solve the relaxation optimally, using for example Linear Programming, and this is used as an upper bound of the discrete solution. Then the solution is progressively discretised and the integrality gap keeps closing, with the relaxation solution offering always an upper bound. Our approach here contrasts with this. Since the multilinear extension is neither convex nor concave we cannot solve the relaxation optimally,

we can only approximate it. The continuous greedy process gives a relaxation solution that is within a constant factor of the optimal discrete solution (not the optimal continuous solution). And, to carry over this approximation guarantees to the discrete solution we show that the rounding procedure does not loose too much, only a factor of ϵ.

Finally, as we shall see later, the number of communication rounds per step that our algorithm will require is precisely the number of iterations of the outer while loop in Algorithm 6. Therefore, we now derive an upper bound on their number.

Lemma 4.2.9. *The while loop in the Threshold-Greedy (Algorithm 6) procedure makes at most $O(n_{\mathcal{T}})$ iterations.*

Proof. First, note one of the invariants of that the design of the algorithm enforces: no tasks are de-allocated, and submodularity enforces that the marginal value cannot increase, hence once the threshold drops to a certain level, it can only decrease from there. Second, note that by the design of the algorithm, at each iteration of the while loop at least one task is allocated, the one with the highest marginal value that sets w^*. Therefore the total number of iterations of the while loop will be at most the number of tasks, because the threshold can only force it to stop earlier.

□

Thus far we have presented a centralised algorithm that has constant factor approximation ratios for monotone and non-monotone submodular functions. We have designed this algorithm carefully such that the comparison logic to allocate tasks relies solely in *max* operations. In the next section we will see that we can take this algorithm and decentralise it using max-consensus protocols.

4.3 Decentralised Algorithm

In this section we take the centralised algorithm presented above and decentralise it so that it can run based on max-consensus exchanges among the agents, with each agent having only access to its own *private* utility function.

4.3.1 Algorithm Definition

Main Algorithm

The main decentralised algorithm that each agent runs is presented in Algorithm 9. This is a decentralised version of Algorithm 4, and has two main parts, first the agents run a max-consensus-based version of the Threshold-Greedy algorithm to obtain a fractional solution $\mathbf{y}^*$ and then the agents round the solution using a rounding procedure based on combining the Random-Round procedure with a wave protocol to exchange valuations. In the following we describe, in more detail, how the relaxation and rounding algorithm work. By design, we present the decentralisation protocols, max-consensus and wave, as a conceptual abstractions and we do not describe a specific algorithm or protocol for them. This is because there are many different alternatives that could be used to implement each of them, each with its own advantages and disadvantages

depending on what the specific network topology is and what the communication setup is. In this sense, our algorithm network-agnostic because it does not carry any implicit assumptions about what form of communication is most beneficial in different settings, enabling trade-offs between convergence speed and communication costs.

Algorithm 9: Decentralised Task Allocation

Input : $\mathcal{T}, \mathcal{A}, f_a : 2^{\mathcal{T}} \to \mathbb{R}^+$, $\epsilon \in [0, 1]$, and $m \in \mathbb{Z}^+$
Output: $B_a^* \subseteq \mathcal{T}$ for all $a \in \mathcal{A}$, s.t. $B_i \cap B_j = \emptyset$ for all $j, i \in \mathcal{A}$ with $i \neq j$
$\mathbf{y}^* \leftarrow$ `Consensus-Solve-Relaxation(`$\mathcal{T}, \mathcal{A}, f_a, \epsilon$`)`
$B^* \leftarrow$ `Decentralised-Round-Relaxation(`$\mathbf{y}^*, \mathcal{T}, \mathcal{A}, f_a, m$`)`
Return: B_a^* for all $a \in \mathcal{A}$

Relaxation

To solve the relaxation, the agents run a local algorithm (Algorithm 10), which incrementally builds the fractional solution based on the Threshold-Greedy algorithm that we introduced before. The coordination between agents happens in the selection of the task-agent pairs that are incremented. We call this decentralised version of the Threshold-Greedy, the Consensus Threshold-Greedy Algorithm (Algorithm 11) which is essentially the Threshold-Greedy algorithm introduced in the previous section (Algorithm 6) but we replace the *max* operations with a max-consensus protocol.

Broadly speaking the algorithm works as follows. Each iteration the agents find which, among the unallocated tasks, provides the maximum marginal value, w_a^*, and pre-allocates that task alongside the set of remaining tasks that, if selected, have a marginal value above $(1-\epsilon)w_a^*$ given the current pre-allocation set B'_a. This step is exactly the same as in the centralised version. Then, a max-consensus round is carried out in which agents exchange their marginal values for each remaining task to find which is the best marginal value for each remaining task among all the agents. For the max-consensus exchange, the tasks that are the pre-allocated bundles B'_a get their exchange values set to their w_{aj} values, and those that are not in the pre-allocated bundles get their values set to $-\infty$. Consequently, the result of the max-consensus round is that all the agents have the same set $\mathcal{W}^*$ which contains for each remaining task $j \in \mathcal{R}$ the maximum valuation w_{a^*j} and the agent responsible for it $a^* \in \mathcal{A}$ (we break ties deterministically e.g. using the agents's ID lexicographical order). Subsequently, each agent locally finds which is the task with the highest marginal value and sets it as the global threshold, w^*, and then it allocates all the tasks which have a marginal value within a factor of $1-\epsilon$ of it to their respective agents. Since all the agents have the same set $\mathcal{W}^*$, the local copy of the allocation sets B_a for all $a \in \mathcal{A}$ that each agent has is the same between agents. Finally, since the agents converge to the same B_a for all $a \in \mathcal{A}$, the relaxation solution is updated each time in exactly the same manner across agents, (see Algorithm 5 and 10), therefore the relaxation solution obtained is the same too.

The the algorithm is decentralised, in the sense that it runs locally with only access to each agent's own utility function, while the coordination is carried out by means of the max-consensus protocol. The advantage of this approach is allows the use of our algorithm in any network/communication setting where a max-consensus protocol is available (which, for example, can include asyn-

chronous networks and/or time delays.) We do not describe the max-consensus protocol here because it has been extensively studied in the literature, see for example [21, 38, 43]. Nor we describe which specific flavour of max-consensus to use, since these depend on the specifics of the network model to which it applies and we want keep our algorithm general. Broadly speaking in terms the number of rounds required depends on whether the network is synchronous or asynchronous. In a synchronous network without delays max-consensus takes $O(D)$ time steps. While in asynchronous networks with time delays, max-consensus protocols have been devised that run in $O(T \cdot D)$ time steps [38] (Where D is the network diameter and T is an asynchronism measure defined as the maximum time between updates of a node.) Naturally, these results only apply to connected networks, since no consensus is possible on a disconnected network.

Algorithm 10: Decentralised Solve-Relaxation

Input : $f_a : 2^{\mathcal{T}} \to \mathbb{R}^+ \, \forall a \in \mathcal{A}$, $\epsilon \in [0, 1]$.
Output: A fractional allocation $\mathbf{y} \in \mathcal{K}$.

```
// Initialisation
```
$\mathbf{y}(0) \leftarrow \mathbf{0}$
$\delta \leftarrow \epsilon$

```
// Main Loop
```
for $t = \{0, \delta, 2\delta, 3\delta, \ldots, 1 - \delta\}$ **do**
  $\underset{\forall a \in \mathcal{A}}{B_a} \leftarrow$ `Consensus-Threshold-Greedy`$(\mathcal{T}, \mathcal{A}, f, \mathbf{y}(t), \epsilon, \delta)$
  for $a \in \mathcal{A}$ **do**
    if $j \in B_a$ **then**
      $y_{aj}(t + \delta) \leftarrow 1 + e^{-\delta}(y_{aj}(t) - 1)$
    else
      $y_{aj}(t + \delta) \leftarrow y_{aj}(t)$

Return: $\mathbf{y}(1)$

Algorithm 11: Consensus Threshold-Greedy Algorithm

Input : $\mathcal{T}$, $\mathcal{A}$, $f_a : 2^{\mathcal{T}} \to \mathbb{R}^+$, $\mathbf{y} \in [0,1]^{\mathcal{A}\times\mathcal{T}}$, $\epsilon \in [0,1]$, $\delta \in [0,1]$.
Output: Sets B_a for all $a \in \mathcal{A}$, such that $\sum_{a\in\mathcal{A}} |B_a \cap \{j\}| \leq 1$ for all $j \in \mathcal{T}$.

`// Initialisation`
$B_a \leftarrow \emptyset$, for all $a \in \mathcal{A}$
$\mathcal{R} \leftarrow \mathcal{T}$
$\bar{y}' \leftarrow \max_{\substack{a\in\mathcal{A}\\ j\in\mathcal{T}}} (1 + e^{-\delta}(y_{aj} - 1))$

`// compute local maximum`
$\bar{d}_a \leftarrow \max_{j\in\mathcal{T}} f_a(j)$

`// find global maximum`
$\bar{d} \leftarrow$ `Max-Consensus(`$\bar{d}_a$`)`
$w^* \leftarrow \bar{d}$

`// main Loop`
while $\mathcal{R} \neq \emptyset$ *and* $w^* \geq \epsilon \frac{\bar{d}}{n_{\mathcal{T}}}(1 - \bar{y}')$ **do**
  `// agent's loop initialisation`
  for $j \in \mathcal{R}$ **do**
    $w_{aj} \leftarrow \Delta F_{aj}(\mathbf{y}_a(B_a, \delta))$
    `// averaging` $O\left(\frac{n_{\mathcal{T}}^2}{\epsilon^2}\left(\frac{\bar{d}+d}{d}\right)^2 \log(n_{\mathcal{T}})\right)$ `iid random samples.`
  $j^* \leftarrow \operatorname{argmax}_{j\in\mathcal{R}} w_{aj}$
  $w_a^* \leftarrow w_{aj^*}$
  $B'_a \leftarrow B_a + j^*$
  `// agent's pre-allocation loop`
  for $j \in \mathcal{R} - j^*$ **do**
    $w_{aj} \leftarrow \Delta F_{aj}(\mathbf{y}_a(B'_a, \delta))$
    `// averaging` $O\left(\frac{n_{\mathcal{T}}^2}{\epsilon^2}\left(\frac{\bar{d}+d}{d}\right)^2 \log(n_{\mathcal{T}})\right)$ `iid random samples.`
    if $w_{aj} \geq (1-\epsilon) w_a^*$ **then**
      $B'_a \leftarrow B'_a + j$
    else
      $w_{aj} \leftarrow -\infty$
  `// find the best marginal values`
  $\mathcal{W}^* \leftarrow$ `Max-Consensus(`w_{aj} *for each* $j \in \mathcal{R}$`)`
  `// task allocation loop`
  $w^* \leftarrow \max_{w_{a^*j}\in\mathcal{W}^*} w_{a^*j}$
  if $w^* \geq \epsilon \frac{\bar{d}}{n_{\mathcal{T}}}(1 - \bar{y}')$ **then**
    for $w_{a^*j} \in \mathcal{W}^*$ **do**
      if $w_{a^*j} \geq (1-\epsilon) w^*$ **then**
        $B_{a^*} \leftarrow B_{a^*} + j$
        $\mathcal{R} \leftarrow \mathcal{R} - j$

Return: B_a for all $a \in \mathcal{A}$
*The notation $\mathbf{y}_a(B_a, \delta)$ means $y_{aj}(B_a, \delta) = y_{aj}(t)$ for $j \notin B_{aj}$, and $y_{aj}(B_a, \delta) = 1 + e^{-\delta}(y_{aj} - 1)$ for $j \in B_{aj}$.

Rounding

The decentralised rounding procedure is presented in Algorithm 12. It takes the relaxation solution y^* produced by Algorithm 11 and produces a feasible allocation that maintains the approximation ratio. This algorithm is essentially the same as the centralised rounding procedure presented in Algorithm 7. The differences stem from the fact that in a decentralised setting we do not have access to the utility functions of other agents. Let us now describe in more detail how an agent $a \in \mathcal{A}$ runs the algorithm. First, the agent generates $\lceil \frac{m}{n_{\mathcal{A}}} \rceil$ random allocations B^i using the `RandomRound` subroutine (Algorithm 8), which randomly allocates each task $j \in \mathcal{T}$ to an agent $a \in \mathcal{A}$ with probability y_{ij}. Second, it requests from each agent $k \in \mathcal{A} - a$ its valuation for all the allocation sets B^i_k for all $i \in [1, 2, ..., \lceil \frac{m}{n_{\mathcal{A}}} \rceil]$ calling the subroutine `Wave-Valuation-Request`. This subroutine works using a distributed wave protocol, in which agents relay the requests for valuations among them until the target agent (requestee) receives it and responds with its valuations, subsequently its reply is relayed in the same manner until it reaches the originating agent (requester). After all the requests have been sent, the agent waits until it receives the responses with the valuations for each agent $k \in \mathcal{A} - a$ for all the $\lceil \frac{m}{n_{\mathcal{A}}} \rceil$ random allocations. Once all the responses have been received, for each random allocation B^i, the agent retrieves the valuation sent by all the other agents $\mathcal{V}(B^i)$. Therefore, the agent is now able to obtain the total value $\mathcal{F}(B^i)$ for each of the random allocation $i \in \{1, ..., \lceil \frac{m}{n_{\mathcal{A}}} \rceil\}$. Once the value of each allocation B^i has been computed, the algorithm finds which is the allocation with the highest value and exchanges it with the other agents using a max-consensus protocol. In the max-consensus round, each agent exchanges its best valuation to find which is the allocation with the best score among all the agents.

We do not describe which is the specific algorithm used to implement the wave protocol for the same reasons that we did not specify a max-consensus protocol: because this depends on the specific network topology and/or communication mechanism used. Broadly speaking *wave protocols*, also known as *information distribution protocols*, can be split into three categories: flooding algorithms; echo algorithms; and virtual ring algorithms. Let E denote the number of edges in the network, D be the diameter, and, as usual, let $n_{\mathcal{A}}$ be the number of agents. The complexity of these algorithm ranges from $O(E \cdot D)$ messages and $O(D)$ for flooding algorithms to $O(E)$ messages and $O(n_{\mathcal{A}})$ time in echo and ring algorithms [92]. The interested reader is referred to any standard distributed algorithms references [34, 68, 92] for further insight into the topic. Our algorithm remains agnostic to which exact flavour of wave protocol is implemented. Hence, it provides flexibility to perform message vs time trade-offs depending on the specific situation. Nevertheless, our algorithm does require that the network is connected.

Algorithm 12: Decentralised Round-Relaxation

Input : $\mathbf{y} \in \mathcal{K}$, $\mathcal{T}$, $\mathcal{A}$, f_a, m
Output: Sets B_a^* for all $a \in \mathcal{A}$, s.t. $\sum_{a \in \mathcal{A}} |B_a^* \cap \{j\}| \leq 1$ for all $j \in \mathcal{T}$.

for $i \in \{1, 2, ... \lceil \frac{m}{n_{\mathcal{A}}} \rceil \}$ **do**
  $B^i \leftarrow$ `RandomRound(`$\mathbf{y}$, $\mathcal{A}$, $\mathcal{T}$`)`

`Wave-Valuation-Request(`B^i *for each* $i \in [1, 2, ..., \lceil \frac{m}{n_{\mathcal{A}}} \rceil]$`)`

`Wait-Valuation-Response()`

for $i \in \{1, 2, ... \lceil \frac{m}{n_{\mathcal{A}}} \rceil \}$ **do**
  $\mathcal{V}(B^i) \leftarrow$ `Wave-Valuation-Response(`B^i`)`
  $\mathcal{F}(B^i) \leftarrow \mathcal{V}(B^i) + f_a(B_a^i)$

$i^* \leftarrow \operatorname{argmax}_i \mathcal{F}(B^i)$
$B^* \leftarrow$ `Max-Consensus(`$\mathcal{F}(B^{i^*})$`)`

Return: B_a^* for all $a \in \mathcal{A}$

4.3.2 Algorithm Analysis

The analysis of the decentralised algorithm has two sides, one is the quality of the solution, i.e. the approximation ratios; and the other is the computational cost. In the decentralised setting the computational cost has in turn two sides: the communication cost; and the number of calls to the value oracle. Here we analyse each of these. First we show that the decentralised algorithm is equivalent to the centralised version that we presented in the first part of the chapter, in order to conclude that the decentralised algorithm has the same approximation ratio. Second, we quantify the communication cost of the algorithm. And finally, we quantify the number of calls to the submodular value oracle that each agent needs.

Equivalence and Approximation Ratio

In order to analyse the performance of Algorithm 9 we only need to show that it is equivalent the centralised version, Algorithm 4. Therefore, all the results for the centralised algorithm would apply to it. By equivalent, we mean that both algorithms behave in the same way, that is, that they produce solution with the same probability distribution. First we show that the equivalence of the relaxation solution, then we investigate the rounding procedure. Both algorithms are very similar. The only differences come from the fact that the centralised version does have access to every agent utility function while in the decentralised version each agent has access only to its own utility function, and exchanges information using max-consensus procedures.

Lemma 4.3.1. *Given the same inputs, the Consensus Decreasing Threshold Algorithm (Algorithm 11) provides an equivalent solution to that provided by the Centralised Threshold Greedy Algorithm (Algorithm 6).*

Proof. The algorithms have an inherent random component due to the errors introduced by the sampling of the multilinear extension. Both algorithms follow

the same sampling approach, therefore the errors in both algorithms have the same distribution. Given this, in order to show that both algorithms are equivalent, we need to show that when the random errors are the same, they provide the same solution. That is, we show that when we assume that the estimates of the marginal values, given a common allocation state $B_a^{\text{cent}} = B_a^{\text{dec}}$ for all $a \in \mathcal{A}$, are the same in both algorithms, i.e. $\bar{w}_{aj}^{\text{cent}} = \bar{w}_{aj}^{\text{dec}}$ for all $a \in \mathcal{A}$ for all $j \in \mathcal{T}$; then both algorithms return the same allocation sets.

It is easy to see that the both algorithms are equivalent if the following three facts hold: first, they start with with the same allocation set state, $B_a = \emptyset$ for all $a \in \mathcal{A}$; second in each iteration both algorithms select the same tasks-agent pairs; and third, both have the same termination logic. The first fact is trivially true because both algorithms are initialised in the same way. In the following we prove prove the second and third facts.

Let us now show that both algorithms add the same task-agent pairs. Because the algorithms have the same allocation logic, given the same input set $\mathcal{W}^*$, both algorithms produce the same allocation sets. Therefore, to show that both algorithms add the same task-agent pairs we need to show that, given the same allocation state B_a for all $a \in \mathcal{A}$, the set $\mathcal{W}^*$ is the same in both algorithms. This is true for two reasons. First, both the centralised and decentralised algorithms follow the same pre-allocation logic to compute the marginal values w_{aj} for the unallocated tasks. Second, the max-consensus protocol, by definition, finds the agent with the maximum marginal value for each task, hence:

$$w_{a^* j_{\text{dec}}} = \max_{a \in \mathcal{A}} w_{aj} = w_{a^* j_{\text{cent}}} \tag{4.30}$$

for each $j \in \mathcal{R}$. Thus, the set $\mathcal{W}^*$ is the same in both algorithms.

Now, since both algorithms start with the same state, and add the same task-agent pairs at each iteration, we can conclude that both have the same allocation state in any given iteration before termination.

Finally, we show that both algorithms terminate in the same iteration, and hence, in the same state. In the centralised version, we have that $\bar{d}_{\text{cent}} = \max_{\substack{a \in \mathcal{A} \\ j \in \mathcal{T}}} f_a(j)$; whereas in the decentralised algorithm we have that the max-consensus protocol finds $\bar{d}_{\text{dec}} = \max_{a \in \mathcal{A}} \bar{d}_a$, where $\bar{d}_a = \max_{j \in \mathcal{T}} f_a(j)$ for all $a \in \mathcal{A}$. Therefore:

$$\bar{d}_{\text{dec}} = \max_{a \in \mathcal{A}} \left(\max_{j \in \mathcal{T}} f_a(j) \right) = \max_{\substack{a \in \mathcal{A} \\ j \in \mathcal{T}}} f_a(j) = \bar{d}_{\text{cent}}, \tag{4.31}$$

hence, both algorithms have the same global maximum marginal value, and as a consequence, they have the same stopping thresholds $\epsilon \frac{\bar{d}}{n_{\mathcal{T}}}(1 - \bar{y}')$. The algorithms have the same termination logic, satisfied in two situations: if all the tasks are allocated; or if the threshold drops bellow the stopping threshold. Observe that because both select the same task-agent pairs each round, the set $\mathcal{R}$ is the same in both algorithms in any given round, and would stop at the same round if the allocation of all tasks was the termination event. Similarly, we have also seen that both algorithms, at each iteration, have the same set $\mathcal{W}^*$ and, thus, the same threshold w^*. This implies that if the termination event was the threshold dropping below the stopping threshold, both would stop in the same iteration. Consequently, in both algorithms both termination events

(allocation of all tasks or threshold below the stop level) would occur in the same iteration with the same allocation state.

□

We can now that the solution of the relaxation is equivalent in both algorithms.

Lemma 4.3.2. *Given the same inputs, the Decentralised Solve-Relaxation Algorithm (Algorithm 10) returns an equivalent solution to that returned by the Centralised Solve-Relaxation Algorithm (Algorithm 5).*

Proof. Each agent runs locally Algorithm 10, which is exactly the same as the centralised version, Algorithm 5, with the exception of the threshold-greedy algorithm used to select the set of task-agent pairs that are incremented, $B(t)$. But from Lemma 4.3.1 the centralised and decentralised version of the Threshold-Greedy algorithm, Algorithms 6 and 11, are equivalent. Hence, the solution provided by the centralised and decentralised versions of the measured continuous greedy, Algorithms 5 and 10 respectively, are equivalent. □

We now look at the rounding step. Both algorithms use the same rounding routine, with the only difference that the since the decentralised algorithm does not have access to all the agents utility functions it needs to use the a wave-valuation protocol and use max-consensus to find the best rounded solution.

Lemma 4.3.3. *Given the same inputs, and if $n_{\mathcal{A}}$ is an exact divisor of m, the Decentralised Round-Relaxation (Algorithm 12) returns a solution equivalent to that returned by the Centralised Round-Relaxation (Algorithm 7).*

Proof. When $n_{\mathcal{A}}$ is an exact divisor of m, both algorithms generate the same number of random allocations. This is because in the decentralised algorithm each of the $n_{\mathcal{A}}$ agents produces $\frac{m}{n_{\mathcal{A}}}$ random allocations. Note that both use the same routine, Algorithm 8, to produce the random allocations. Further, both algorithms estimate the value of each allocation in the same way, because the decentralised version does not continue until it has gathered all the valuations from all the agents. Therefore, given that the max operation in the centralised version is equivalent to the decentralised max-consensus over the maxima of each agent, both algorithms obtain random solutions with the same distribution. Therefore, both algorithms produce equivalent rounded solutions. □

Note that in the case when $n_{\mathcal{A}}$ is not an exact divisor of m, the solution obtained by the decentralised version will be based on $\lceil \frac{m}{n_{\mathcal{A}}} \rceil n_{\mathcal{A}}$ random allocations, and hence we know from Theorem 4.2.8 that the probability of a good answer can only be higher than that of the centralised equivalent that used only m random allocations.

At this point we have established that both the relaxation and the rounding are equivalent in the centralised and the decentralised setting. Hence, we can now state the main result of this chapter.

Theorem 4.3.4. *The decentralised task allocation procedure, Algorithm 9, returns a feasible allocation solution B^* such that, with probability at least $1 - e^{-\frac{\epsilon^2}{2}\left(\frac{1}{e}-2\epsilon\right)m}$, the following bounds hold:*

- *If* $f_a : 2^{\mathcal{T}} \to \mathbb{R}^+$ *for all* $a \in \mathcal{A}$ *are non-negative monotone submodular functions then:*

$$\mathcal{F}(B^*) \geq \left(1 - \frac{1}{e} - 4\epsilon\right) \mathcal{F}(OPT). \tag{4.32}$$

- *If* $f_a : 2^{\mathcal{T}} \to \mathbb{R}^+$ *for all* $a \in \mathcal{A}$ *are general (possibly non-monotone) non-negative submodular functions then:*

$$\mathcal{F}(B^*) \geq \left(\frac{1}{e} - 3\epsilon\right) \mathcal{F}(OPT). \tag{4.33}$$

Proof. This result is immediate from the equivalence relations derived above and Theorem 4.2.8. From Lemma 4.3.2, the relaxation solution is equivalent to the centralised solution. And from Lemma 4.3.3 the rounded solution is equivalent, in the case when $n_{\mathcal{A}}$ is a exact divisor of m; if it is not, then $\lceil \frac{m}{n_{\mathcal{A}}} \rceil n_a > m$, which implies that the probability of a solution is higher than the centralised solution. Therefore, Theorem 4.2.8 readily applies to the decentralised algorithm, Algorithm 9. □

Communication Complexity

Let us now quantify the number of communication rounds that our algorithm requires. We do so in terms of rounds of the max-consensus protocols and wave requests. This is because our algorithm is presented using these protocols as an abstraction layer to maintain generality and enable its applicability to different communication setups. First, we quantify the number of communication rounds needed to solve the relaxation and second we look at the number of communication round required by the rounding step.

Lemma 4.3.5. *The relaxation solution requires* $O(\frac{n_{\mathcal{T}}}{\epsilon})$ *max-consensus rounds.*

Proof. The number of max-consensus rounds is determined by the number of iterations of the while loop in Algorithm 11 multiplied by the number of times that Algorithm 10 calls Algorithm 11. In each run of Algorithm 10 there are at most $O(n_{\mathcal{T}})$ iterations of the while loop (because in each of them at least one task is allocated), and Algorithm 10 performs $\frac{1}{\epsilon}$ iterations, calling once Algorithm 10 in each iteration. Thus the total number of max-consensus rounds needed to solve the relaxation is at most $O\left(\frac{n_{\mathcal{T}}}{\epsilon}\right)$. □

Let us now quantify the number of communication rounds needed in the rounding step.

Lemma 4.3.6. *The rounding solution requires 1 max-consensus round and* $\lceil \frac{m}{n_{\mathcal{A}}} \rceil n_{\mathcal{A}}$ *wave-valuation rounds.*

Proof. Max-consensus is only called once at the end of the rounding solution in Algorithm 12. Each of the $n_{\mathcal{A}}$ agent triggers $\lceil \frac{m}{n_{\mathcal{A}}} \rceil$ wave-valuation requests, that is, one per random allocation produced. Hence the total number of wave-valuation rounds is $\lceil \frac{m}{n_{\mathcal{A}}} \rceil n_{\mathcal{A}}$. □

Local Computational Cost

Now we quantify the number of calls that each agent has to perform to its local utility function. As it is typical in the submodular optimisation literature, we quantify it in terms of the number of calls to a *value oracle*. We first quantify the number of calls required by to solve the relaxation and then by the rounding.

In order to bound the number of calls required by the relaxation we need to quantify the number of calls that the decreasing threshold subroutine needs in each time it is called.

Lemma 4.3.7. *Each agent's call to the Consensus Threshold-Greedy Algorithm (Algorithm 11) requires* $O\left(\frac{n_{\mathcal{T}}^4}{\epsilon^2}\left(\frac{\bar{d}+\underline{d}}{\underline{d}}\right)^2\log(n_{\mathcal{T}})\right)$ *calls to the value oracle.*

Proof. There are at most $O(n_{\mathcal{T}})$ iterations of the while loop (from Lemma 4.2.9). In each of those iterations there are at most $O(n_{\mathcal{T}})$ calls to estimate the marginal value, i.e. $\Delta F_{aj}(\mathbf{y}_a(B_a,\delta))$. Hence, there is at most $O(n_{\mathcal{T}}^2)$ estimates of the marginal value. And each approximation by sampling of the marginal value requires $O\left(\frac{n_{\mathcal{T}}^2}{\epsilon^2}\left(\frac{\bar{d}+\underline{d}}{\underline{d}}\right)^2\log(n_{\mathcal{T}})\right)$. Therefore, each iteration requires $O\left(\frac{n_{\mathcal{T}}^4}{\epsilon^2}\left(\frac{\bar{d}+\underline{d}}{\underline{d}}\right)^2\log(n_{\mathcal{T}})\right)$ calls to the value oracle. □

Note that this bounds are assuming a general submodular function for which we do not have a closed form to evaluate the marginal values of the multilinear extension. However, Iyer [44] showed that there exist a closed form which would eliminate the burden due to sampling for many submodular functions of practical interest such as: graph cuts, weighted sums of matroid ranks, set coverage functions, facility location, and concave compositions over cardinality, among others. Hence, instead of doing $O\left(\frac{n_{\mathcal{T}}^2}{\epsilon^2}\left(\frac{\bar{d}+\underline{d}}{\underline{d}}\right)^2\log(n_{\mathcal{T}})\right)$ calls in each iteration we would do just 1. And, the total number of calls to the *multilinear oracle* would be $O(n_{\mathcal{T}}^2)$.

Now we can quantify the number of calls needed to solve the relaxation.

Lemma 4.3.8. *Solving the relaxation using the decentralised continuous greedy algorithm 10 requires that each agent calls the value oracle* $O\left(\frac{n_{\mathcal{T}}^4}{\epsilon^3}\left(\frac{\bar{d}+\underline{d}}{\underline{d}}\right)^2\log(n_{\mathcal{T}})\right)$ *times.*

Proof. There are $\frac{1}{\epsilon}$ calls to the Consensus Threshold-Greedy Algorithm (Algorithm 11), and from Lemma 4.3.7 each call requires $O\left(\frac{n_{\mathcal{T}}^4}{\epsilon^2}\left(\frac{\bar{d}+\underline{d}}{\underline{d}}\right)^2\log(n_{\mathcal{T}})\right)$, thus the total calls required is $O\left(\frac{n_{\mathcal{T}}^4}{\epsilon^3}\left(\frac{\bar{d}+\underline{d}}{\underline{d}}\right)^2\log(n_{\mathcal{T}})\right)$. □

Note that in the monotone case the lower bound on the marginal value $\underline{d}$ is, by definition, zero, hence with $\underline{d}=0$, the term $\left(\frac{\bar{d}+\underline{d}}{\underline{d}}\right)^2$ vanishes. And if we have access to a multilinear oracle, the total number of calls is only $O\left(\frac{n_{\mathcal{T}}^2}{\epsilon}\right)$.

Finally we quantify the number of calls to round the relaxation.

Lemma 4.3.9. *The Decentralised Round-Relaxation Algorithm (Algorithm 12) requires $n_{\mathcal{A}}\lceil\frac{m}{n_{\mathcal{A}}}\rceil$ calls to the value oracle function.*

Proof. Each of the $n_{\mathcal{A}}$ agent produces $\lceil\frac{m}{n_{\mathcal{A}}}\rceil$ random allocations. Hence, to respond to the corresponding wave valuation requests each agent must make $n_{\mathcal{A}}\lceil\frac{m}{n_{\mathcal{A}}}\rceil$ calls to the value oracle function. □

Note that, since in practical cases tipically $m \ll O\left(\frac{n_{\mathcal{T}}^4}{\epsilon^3}\left(\frac{\bar{d}+\underline{d}}{\underline{d}}\right)^2\log(n_{\mathcal{T}})\right)$, most of the computational effort is spent on solving the relaxation.

The local running time is not very good at $O\left(\frac{n_{\mathcal{T}}^4}{\epsilon^3}\left(\frac{\bar{d}+\underline{d}}{\underline{d}}\right)^2\log(n_{\mathcal{T}})\right)$ however, we believe that this is a very loose upper bound for two reasons. First the number of samples required to estimate the marginal value has potential to be reduced by using more intelligent sampling strategies. Such as for example adaptively sampling only those tasks that have potential to be selected in a given round. That is, rather than starting from scratch at each iteration, information from previous rounds could be used to inform which tasks to sample. The second reason why we believe the number of samples can be reduced is because we have used an overly pessimistic upper bound in the number of iterations of the while loop, $O(n_{\mathcal{T}})$, inside the decreasing threshold procedure. We estimate this number will be in the order of $O\left(\min\left(n_{\mathcal{A}}\frac{1}{\epsilon}\log\left(\frac{n_{\mathcal{T}}}{\epsilon}\right), n_{\mathcal{T}}\right)\right)$ due to the speed at which the threshold decreases, but we have been unable to prove it and so it remains a conjecture. Therefore we think that the most immediate work would be to prove or disprove it.

Furthermore, in our work we have set the number of samples in each iteration to be $O\left(\frac{n_{\mathcal{T}}^2}{\epsilon^2}\left(\frac{\bar{d}+\underline{d}}{\underline{d}}\right)^2\log(n_{\mathcal{T}})\right)$ this is a consequence of Lemma 3.3.2 and it supports the analysis for general non-negative submodular functions. However, as shown in the work of [6] to use a decreasing-threshold approach in the monotone case we only need to sample $O(\frac{n_{\mathcal{T}}\log n_{\mathcal{T}}}{\epsilon})$. Which if combined with the conjecture on the tighter analysis on the number of iterations of the while loop above, would yield a local complexity of $O\left(\frac{n_{\mathcal{A}}n_{\mathcal{T}}}{\epsilon^3}\log(n_{\mathcal{T}})^2\right)$ value oracle calls which is asymptotically better than CBBA's $O(n_{\mathcal{T}}^2)$ when $n_{\mathcal{A}} \ll n_{\mathcal{T}}$. It is also interesting to point out that for monotone functions $\underline{d} = 0$, and so the term $\frac{\bar{d}+\underline{d}}{\underline{d}}$ cancels out.

Moreover, this analysis has been conducted quantifying the computational cost on the *value oracle* model because for general submodular functions no closed form of the multilinear extension exists. However, for many submodular functions of practical interest such a closed form does exist. Iyer [44] shows how to calculate the multilinear extensions efficiently of the following functions: graph cuts, weighted sums of matroid ranks, set coverage functions, and facility location, among others. When we have access to a closed form of the multilinear extension we say we have access to a *mulitlinear oracle* rather than to a *value oracle*, and it would reduce the local computational burden in each iteration from $O\left(\frac{n_{\mathcal{T}}^2}{\epsilon^2}\left(\frac{\bar{d}+\underline{d}}{\underline{d}}\right)^2\log(n_{\mathcal{T}})\right)$ value oracle calls to just 1 multilinear oracle call. Therefore, if the conjecture above were to be true and we were to have access to a *multilinear oracle*, then our algorithm could be proven to be much more efficient than CBBA when $n_{\mathcal{A}} \ll n_{\mathcal{T}}$.

Another option to reduce the computational burden of our approach is to implement less naive sampling strategies. As we have defined the algorithm we sample up to an additive error, and then make decisions based on the result of the sampling. That is, we sample until we have the highest accuracy required by the theorems, even when we may not need it. A more efficient strategy would be to keep online confidence bounds on the estimates of the marginal values and only sample until we have satisfactory confidence to make a given decision. For example, to tell whether a marginal value is above or below the threshold, we would only have to sample until we have enough confidence that the value is below, or above, $w(1-\epsilon)$. This will require a number of samples that is less or equal that the number required by the analysis here. If we have confidence that a given task-agent pair has a negative value, there is no point in finding this value to the accuracy required, we already know what the decision to be made by the algorithm is, and hence, we can safely skip it, and move on to the next task. Another example would be at the beginning of an iteration, here each agent needs to find which is the task that provides the highest marginal value. In our algorithm we state that all marginal values are sampled up to a certain level, and then we chose the highest. A more efficient strategy would be to use an adapted version of the Upper Confidence Bound (UCB) Algorithm [90], where the agent would sample the task with the highest confidence bound on the marginal value. We believe this could yield reduction in the number of samples required to $O(\frac{1}{n\tau})$. Similarly, we could re-use information between iterations, since our utilities are submodular, the marginal values cannot increase from one iteration to the next (because the fractional allocations can either increase or stay constant). Therefore, if we have confidence from a previous iteration that the marginal value of a task-agent pair is below certain level, in the following iterations we can safely assume that the marginal value of that task-agent pair is below that certain level.

4.4 Summary

We have presented a decentralised Task Allocation algorithm that provides approximation guarantees for non-negative monotone ($1-\frac{1}{e} \approx 63\%$) and non-monotone ($\frac{1}{e} \approx 37\%$) submodular functions whilst relying, only, on local utility function calls and neighbour to neighbour communications. We have given a full formal analysis on the approximation guarantees, communications and computational complexity. This is the first decentralised algorithm, i.e. that assumes only *local* or *private* access to each agent's utility function, that is able to provide a constant factor approximation for the non-monotone case. We believe this to be of relevance as it enables the use of non-monotone functions that are characteristic of many practical situations. This is because non-monotonicity captures the natural situation when agents *"bite off more than they can chew"* and allocating too many tasks to a given agent ends up destroying utility. This is a situation in which previous decentralised algorithms, such as CBBA [19], could not offer approximation guarantees. In Table 4.1 we present a summary of the performance of our algorithm compared with CBBA.

	Non-Negative Submodular Monotone			Non-Negative Submodular Non-Monotone		
	Approx. Ratio	Comms. Cost	Local Cost	Approx. Ratio	Comms. Cost	Local Cost
CBBA [19]	$\frac{1}{2}$	$O\left(n_{\mathcal{T}}\right)$	$O\left(n_{\mathcal{T}}^2\right)$	NA	NA	NA
This Chapter	$1-\frac{1}{e}-4\epsilon$	$O\left(\frac{n_{\mathcal{T}}}{\epsilon}+m\right)$	$\tilde{O}\left(\frac{n_{\mathcal{T}}^4}{\epsilon^2}\right)*$	$\frac{1}{e}-3\epsilon$	$O\left(\frac{n_{\mathcal{T}}}{\epsilon}+m\right)$	$\tilde{O}\left(\frac{n_{\mathcal{T}}^4}{\epsilon^2}\right)*$

Table 4.1: Comparison of the performance of the state of the art decentralised task allocation algorithm with the algorithm presented here. The communication cost is quantified in terms of Max-Consensus rounds, while the local cost is quantified in number of calls to the *value oracle*. (*The notation $\tilde{O}$ encapsulates a logarithmic dependency on the number of tasks: $\log n_{\mathcal{T}}$; and in the square of the ratio of marginal values: $\left(\frac{\bar{d}+\underline{d}}{\bar{d}}\right)^2$; that is: $\tilde{O}(x) = O\left(x\left(\frac{\bar{d}+\underline{d}}{\bar{d}}\right)^2 \log(n_{\mathcal{T}})\right)$. Note that for monotone functions $\underline{d}=0$, and so the term $\frac{\bar{d}+\underline{d}}{\bar{d}}$ cancels out. Also note that if we have an analytical form of the multilinear extension, we do not need to sample. Hence, the local cost would be reduced to $O\left(\frac{n_{\mathcal{T}}^2}{\epsilon}\right)$ calls to the *multilinear oracle*)

Chapter 5

Application of Submodular Task Allocation: a Multi-UAV Surveillance Mission

5.1 Introduction

In the previous chapter we have presented our decentralised task allocation algorithm alongside a formal analysis of the approximation bounds. In this chapter we present a preliminary illustration of the performance of our algorithm with a Multi-UAV Surveillance mission. In this scenario, we have to carry out a variety of tasks, e.g.: visual-spectrum imaging, data harvesting from ground sensors, radio signal intelligence collection, infra-red imaging, etc. To execute them, we have a team of UAVs with different capabilities. For example, one UAV may have capability to harvest information from a ground based sensor network, while another may have the sensor required for an imaging task, etc. Additionally, we assume that this is a covert mission in which a hostile enemy is trying to detect our UAVs. Therefore, we need to factor in the chance of detection, and ensure that important tasks are spread among the team, so that the detection of any individual UAV does not compromise the success of the mission. In this context, we present a function that comprises three elements. First, the task-agent value: which is a combination of the task requirements, the UAV skills, and the task importance. Second, the probability of survival, which quantifies the chance of avoiding detection by the enemy. And third, the cross-task penalties which prevent the concentration of important tasks in a single UAV, encouraging the spread of tasks among the team.

To asses the quality of the solutions obtained by our algorithm, we compare it against the state of the art in decentralised task allocation, the Consensus-Based Bundle Auction (CBBA) algorithm [19]. We run experiments with a range of problem sizes: from 30 tasks and 5 UAVs to 100 tasks and 20 UAVs. These experiments show that our algorithm performs significantly better than CBBA, both in absolute terms and in the implicit asymptotic trends. Further,

these trends are observed consistently across a variety of problem sizes. More importantly, the experiments show that our algorithm performs well even with relatively large ϵ values.

The chapter is structured as follows. First, we describe in detail the elements of the problem that we use to build our utility function model. Second, we carry out a formal analysis to prove that our value function is non-negative submodular. Finally, we conclude with the results of the numerical experiments and a discussion of the kind of multi-robot missions that our algorithm enables.

5.2 A Multi-UAV Surveillance Mission

We aim to solve a surveillance mission problem in which a group of UAVs need to explore an enemy area. The tasks have distinct requirements and importance, and the UAVs have different capabilities to execute them. Since the mission is carried out in hostile territory, we also model the probability of enemy detection, and inter-task penalties to prevent accumulation of tasks in a single agent. Finally, we combine all these elements to formulate a Non-Negative Submodular Task Allocation Problem. Let us now formalise all these concepts.

5.2.1 Elements of the Problem

We have a set $\mathcal{A}$ of heterogeneous UAVs, and a set $\mathcal{T}$ of heterogeneous tasks that need to be carried out. We need to allocate to each UAV $a \in \mathcal{A}$ a mission $S_a \subseteq \mathcal{T}$, such that we maximise the value in the team. But what is the value of every possible mission? we know that each task has a different importance, and a set of requirements to be carried out, also, each UAV has different abilities to satisfy those requirements. Naively, we might aim to maximise the total value obtained, however, we need to consider that we are working in an adversarial scenario in which an enemy is capable of detecting our UAVs and put their mission to an end. Therefore, we design an utility function that contemplates this situation, and thus favours missions that are neither too risky (i.e. resulting in a low survival probability) nor are too fragile (i.e. concentrating too many important tasks in a single UAV). In the following we quantify and formalise these matters.

Values

In our problem we have a heterogeneous set of tasks with different importances and that require different abilities or resources for its execution. We distill this in two key magnitudes: the importance of the task $q_j \in \mathbb{R}^+$, and the task-agent match fitness $m_{aj} \in \mathbb{R}^+$. We formalise this as follows. For each task $j \in \mathcal{T}$ we define its importance $q_j \in \mathbb{R}^+$, and a unit vector of requirements for its execution, $\mathbf{v}_j$, the vector is defined such that $v_{ij} \geq 0$ and $\|\mathbf{v}_j\| = 1$. Similarly, for each UAV $a \in \mathcal{A}$ we define a unit vector of attributes $\mathbf{u}_a$, which also satisfies that $v_{ia} \geq 0$ and $\|\mathbf{u}_a\| = 1$. Now, the match fitness for task $j \in \mathcal{T}$ and UAV $a \in \mathcal{A}$ is defined as $m_{aj} \triangleq \mathbf{u}_a^\top \cdot \mathbf{v}_j$. With these definitions we can state how the value that an UAV a can obtain from doing task j is calculated:

$$w_{aj} \triangleq q_j m_{aj} = q_j \left(\mathbf{u}_a^\top \cdot \mathbf{v}_j\right) \tag{5.1}$$

Essentially, we calculate the value of UAV $a \in \mathcal{T}$ executing task j as the importance of j discounted by the misalignment of attributes and requirements between UAV and task. We remark that we have defined the fitness match m_{aj} as the cross product of unit vectors, but it can be defined arbitrarily as long as it is a positive real. For example, in a real-world scenario, this value could be set by an operator or domain expert.

Thus, in a mission $S \subseteq \mathcal{T}$ the agent a can obtain a value of:

$$v_a(S) \triangleq \sum_{j \in S} w_{aj}. \tag{5.2}$$

This is the value that the agent would obtain if it were not detected. However the enemy may detect our UAV. Let us formalise this further.

Probability of a Successful Mission

Each time an UAV carries out a task there is a chance that it is detected by the enemy and thus fail the mission. The probability of detection while executing a task increases as the enemy is given more and more opportunities to become aware about the presence of an UAV. We assume that if an UAV has been detected the mission has failed, and thus produced a zero value. Further, the probability of detection by the enemy also varies depending on the kind UAV being used. To model this, we give the parameters P_{0_a}, and α_a to each UAV $a \in \mathcal{A}$. P_{0_a} is the probability of detection of an UAV by the enemy when executing a single task without having executed any previous tasks before, i.e.:

$$\Pr(\text{detection of } a \text{ at the 1st task}|0) \triangleq P_{0_a}. \tag{5.3}$$

Naturally we have that $1 > P_0 > 0$. Now, to model the increasing awareness of the enemy of an UAV's presence, we define the probability of being detected at the $n+1$th task given that the UAV has executed previously n tasks as:

$$\Pr(\text{detection of } a \text{ at the } n+1\text{th task}|n) \triangleq \frac{P_{0_a}}{1 - \alpha_a n P_{0_a}} \tag{5.4}$$

where $\alpha_a \geq 1$ is a parameter that governs how fast does the probability of detection increases. That is, we assume that the probability of detection increases dramatically (hyperbolically) as we give more chances to the enemy to detect the UAV. Naturally, we need to set up α_a such that $\frac{P_0}{1-\alpha_a(|\mathcal{T}|-1)P_0} < 1$.

Now, we can quantify the probability of detection during the execution of n tasks, $\Pr_{\text{detect}} : \{0, 1, ..., |\mathcal{T}|\} \to [0, 1)$, as:

$$\Pr_{\text{detect},a}(n) = \Pr_{\text{detect},a}(n-1) + \left(1 - \Pr_{\text{detect},a}(n-1)\right) \frac{P_0}{1 - \alpha_a(n-1)P_0} \tag{5.5}$$

This recurrence does not have a nice closed-form analytical solution, but it allows for efficient, linear-time, pre-computation. Note that the base case is simply:

$$\Pr_{\text{detect},a}(0) = P_{0_a}. \tag{5.6}$$

At this point, we can are ready to define the dual of the probability of detection, the probability of surviving the mission, $\Pr_{\text{srv},a} : 2^{\mathcal{T}} \to (0, 1]$, as:

$$\Pr_{\text{srv},a}(S) \triangleq 1 - \Pr_{\text{detect},a}(|S|). \tag{5.7}$$

This is the probability of the UAV $a \in \mathcal{A}$ not being detected during the execution of a mission involving the tasks in $S \subseteq \mathcal{T}$. Please note that we have defined the probability of survival as a set function while the probability of detection has been defined as a function on the integers between 0 of the cardinality of the task set $\mathcal{T}$. We have done this to clarify the exposition later on, but we remind the reader that both of them can be defined in either of the two ways.

Inter-Task Dependencies

In our model, we are assuming an adversarial scenario in which a hostile enemy is able and willing to the detect our UAVs. This introduces a further subtlety: consider the situation when an UAV is very efficient at executing many important tasks while other UAVs have a comparable but worse performance. In this situation, the solution might concentrate many important tasks in a single UAV, and leave other UAVs underused. This would couple the completion of many important tasks to survival of a single UAV. To remove this inherent fragility from our utility function, we introduce inter-task penalties.

We denote by $c^a_{ij} \in \mathbb{R}^+$ the penalty of UAV $a \in \mathcal{A}$ when executing each pair $i, j \in \mathcal{T}$ of distinct tasks in its mission. Therefore, we define the total penalty function, $g_a : 2^{\mathcal{T}} \to \mathbb{R}^+$, for a given mission $S \subseteq \mathcal{T}$ as:

$$g_a(S) \triangleq \sum_{\substack{i,j \in S \\ i \neq j}} c^a_{ij} \tag{5.8}$$

We use $c^a_{ij} = e^{q_i q_j}$ as penalty for each pair of tasks $i, j \in \mathcal{T}$. This is to discourage concentration of important tasks in a single UAV. With this, important tasks will have high penalties when combined with other important tasks, while when combined with comparatively less important tasks, they will have effectively no penalty. Note that the absolute value of the total penalty is not relevant, what matters is the relative value of penalties, because of a scaling factor λ_a, that we will introduce next.

Utility Function

Let us now combine all the terms that have been introduced above to obtain the utility function for our scenario. Intuitively, we define the value of a mission $S \subseteq \mathcal{T}$ of agent $a \in \mathcal{A}$ to be the expected value obtained (considering that the agent might be detected and thus obtain a value of zero) minus by inter-task penalties:

$$f_a(S) = \mathbb{E}_{\text{srv}}[v_a(S)] - \lambda_a g_a(S). \tag{5.9}$$

Where $\lambda_a \in \mathbb{R}^+$ is a scaling factor that is used to ensure that the total penalty is never higher than the value of the tasks, i.e. $\frac{\mathbb{E}_{\text{srv}}[v_a(S)]}{g_a(S)} \geq \lambda_a$ for all $S \subseteq \mathcal{T}$. This ensures that the resulting function remains non-negative.

Now, we can plug in all the definitions from above to yield the final utility function:

$$f_a(S) = \Pr_{\text{srv},a}(S) \sum_{j \in S} w_{aj} - \lambda_a \sum_{\substack{i,j \in S \\ i \neq j}} c_{ij}^a. \tag{5.10}$$

5.2.2 Submodularity Analysis

Before we proceed any further, we need to show that the utility function is indeed non-negative submodular. To do so, first we show that the functions that constitute our utility function are submodular.

Let us start by showing that the probability of survival is submodular.

Lemma 5.2.1. *The probability of survival,* $\Pr_{srv,a} : 2^{\mathcal{T}} \to (0, 1]$*, is submodular, non-negative, and non-increasing.*

Proof. We will show that $\Pr_{\text{srv},a}$ is submodular and non-increasing by leveraging its definition, $\Pr_{\text{srv},a}(S) = 1 - \Pr_{\text{detect},a}(|S|)$, showing that $\Pr_{\text{detect},a}(n)$ is supermodular and non-decreasing. For notational compactness during the derivation, let $p : \{0, 1, ..., |\mathcal{T}|\} \to \mathbb{R}^+$ be the probability of detection after executing $n \in \{0, 1, ..., |\mathcal{T}|\}$ tasks, i.e. $p(n) \triangleq \Pr_{\text{detect}}(n)$. Now, by definition we have:

$$p(n+1) = p(n) + (1 - p(n)) \frac{P_0}{1 - \alpha n P_0}, \tag{5.11}$$

with $p(0) = P_0$.

First, since by assumption on the values of α and P_0 we have that $\frac{P_0}{1-\alpha(|\mathcal{T}|-1)P_0} < 1$, this implies that $p(|\mathcal{T}|) < 1$.

Second, let us show that $p(n)$ is non-decreasing. For all $n \in \{0, 1, ..., |\mathcal{T}|\}$, we have that:

$$\begin{aligned} & p(n) \leq p(n+1) && \text{hypothesis} \\ & p(n) \leq p(n) + (1 - p(n)) \frac{P_0}{1 - \alpha n P_0} && \text{definition} \\ & 0 \leq (1 - p(n)) \frac{P_0}{1 - \alpha n P_0}. \end{aligned} \tag{5.12}$$

and since $(1-p(n)) \geq 0$ and $\frac{P_0}{1-\alpha n P_0} \geq P_0$, the last inequality holds. This implies that $\Pr_{\text{detect},a}$ is non-decreasing, and hence $\Pr_{\text{srv},a}(S) = 1 - \Pr_{\text{detect},a}(|S|)$ is non-increasing. Further, from above we have that $\Pr_{\text{detect},a}(|\mathcal{T}|) < 1$, therefore $\Pr_{\text{srv},a}(|\mathcal{T}|) > 0$, which, given that $\Pr_{\text{srv},a}$ is non-increasing, implies that $\Pr_{\text{srv},a}$ is non-negative.

Finally, we show that the probability of detection is supermodular, by showing that for $0 \leq n \leq |\mathcal{T}| - 2$, we have:

$$p(n+1) - p(n) \leq p(n+2) - p(n+1) \tag{5.13}$$

We prove this by reducing it to the assumption that $\alpha \geq 1$ as follows:

$$
\begin{aligned}
&p(n+1) - p(n) \leq p(n+2) - p(n+1) && \text{hypothesis}\\
&(1-p(n))\frac{P_0}{1-\alpha n P_0} \leq (1-p(n+1))\frac{P_0}{1-(n+1)\alpha P_0} && \text{definition of } p\\
&(1-p(n))\frac{P_0}{1-\alpha n P_0} \leq \left(1-\left(p(n)+(1-p(n))\frac{P_0}{1-\alpha n P_0}\right)\right)\frac{P_0}{1-(n+1)\alpha P_0} && \text{definition of } p\\
&(1-p(n))\frac{P_0}{1-\alpha n P_0} \leq (1-p(n))\left(1-\frac{P_0}{1-\alpha n P_0}\right)\frac{P_0}{1-(n+1)\alpha P_0} && \text{extracting factors}\\
&\frac{P_0}{1-\alpha n P_0} \leq \left(1-\frac{P_0}{1-\alpha n P_0}\right)\frac{P_0}{1-(n+1)\alpha P_0} && 0 < (1-p(n)) \leq 1\\
&\frac{\frac{1}{1-\alpha n P_0}}{1-\frac{P_0}{1-\alpha n P_0}} \leq \frac{1}{1-(n+1)\alpha P_0} && \text{rearranging}\\
&\frac{1}{1-\alpha n P_0 - P_0} \leq \frac{1}{1-(n+1)\alpha P_0} && \text{rearranging}\\
&1-(n+1)\alpha P_0 \leq 1-\alpha n P_0 - P_0 && \text{rearranging}\\
&P_0 \leq \alpha P_0 && \alpha \geq 1
\end{aligned}
$$

which holds due to the assumption that $\alpha \geq 1$, and the hypothesis is proven.

Now, equation 5.13 directly implies that for any two $x, y \in 0, 1, ..., |\mathcal{T}|$ such that $x \geq y$ we have that:

$$p(x+1) - p(x) \leq p(y+1) - p(y) \tag{5.14}$$

hence proving that the probability of detection is supermodular. Now, recall that any negated supermodular function is submodular, therefore, the probability of survival is submodular because it is defined as $\Pr_{\text{srv},a}(S) = 1 - \Pr_{\text{detect},a}(|S|)$. □

Now, let us show that the value of mission discounted by the probability of survival (the expected value of a mission) is submodular.

Lemma 5.2.2. *The function $h(S) \triangleq \Pr_{srv,a}(S)v_a(S)$ is submodular and non-negative.*

Proof. We will prove the submodularity property by showing diminishing marginal values. Let A, B, and x be such that $x \in \mathcal{T}$ and $A \subseteq B \subseteq \mathcal{T} \setminus \{x\}$, then:

$$
\begin{aligned}
&h(x+A) - h(A) \geq h(x+B) - h(B) && \text{hypothesis}\\
&\Pr_{\text{srv},a}(A+x)\sum_{j\in A+x} w_{aj} - \Pr_{\text{srv},a}(A)\sum_{j\in A} w_{aj} \geq \Pr_{\text{srv},a}(B+x)\sum_{j\in B+x} w_{aj} - \Pr_{\text{srv},a}(B)\sum_{j\in B} w_{aj} && \text{definition}\\
&w_{ax}\Pr_{\text{srv},a}(A+x) + \left(\Pr_{\text{srv},a}(A+x) - \Pr_{\text{srv},a}(A)\right)\sum_{j\in A} w_{aj} \geq && \text{re-arranging}\\
&\qquad w_{ax}\Pr_{\text{srv},a}(B+x) + \left(\Pr_{\text{srv},a}(B+x) - \Pr_{\text{srv},a}(B)\right)\left(\sum_{j\in A} w_{aj} + \sum_{j\in B\setminus A} w_{aj}\right). && (5.15)
\end{aligned}
$$

Now from Lemma 5.2.1 we have that $\Pr_{\text{srv},a}$ is submodular, therefore, we have:

$$\Pr_{\text{srv},a}(B+x) - \Pr_{\text{srv},a}(B) \leq \Pr_{\text{srv},a}(A+x) - \Pr_{\text{srv},a}(A). \tag{5.16}$$

But $\Pr_{\text{srv}}$ is also non-increasing, therefore:

$$\Pr_{\text{srv},a}(B+x) - \Pr_{\text{srv},a}(B) \leq 0$$
$$\Pr_{\text{srv},a}(A+x) - \Pr_{\text{srv},a}(A) \leq 0.$$

Since $w_{aj} \geq 0$ for all $j \in \mathcal{T}$ and $a \in \mathcal{A}$, we have that:

$$\left(\Pr_{\text{srv},a}(A+x) - \Pr_{\text{srv},a}(A)\right) \sum_{j \in A} w_{aj} \geq \left(\Pr_{\text{srv},a}(B+x) - \Pr_{\text{srv},a}(B)\right) \left(\sum_{j \in A} w_{aj} + \sum_{j \in B \setminus A} w_{aj}\right), \tag{5.17}$$

and consequently, the inequality in 5.15, can be simplified to:

$$w_{ax} \Pr_{\text{srv},a}(A+x) \geq w_{ax} \Pr_{\text{srv},a}(B+x). \tag{5.18}$$

Which holds because we have from Lemma 5.2.1 that $\Pr_{\text{srv}}$ is non-increasing and $A \subseteq B$, and $w_{ax} \geq 0$. Therefore, h is submodular.

Finally, since $\Pr_{\text{srv}}$ is non-negative and $w_{aj} \geq 0$ for all $a \in \mathcal{A}$ and $j \in \mathcal{T}$, $\Pr_{\text{srv},a}(S) v_a(S)$ is non-negative for all missions $S \subseteq \mathcal{T}$. □

Now, we show that the sum of the penalties in a mission is supermodular.

Lemma 5.2.3. *The penalty function $g(S) \triangleq \sum_{\substack{i,j \in S \\ i \neq j}} c_{ij}$ is supermodular, and non-negative.*

Proof. It is trivial to show that g is non-negative because by definition $g(S) = \sum_{i,j \in S} c_{ij}$ and $c_{ij} \geq 0$ for all $i, j \in \mathcal{T}$.

Now we can easily prove supermodularity by showing increasing marginal values. Given any task $x \in \mathcal{T}$, let A and B be any such that $A \subseteq B \subseteq \mathcal{T} \setminus x$,

we have:

$$g(B+x)-g(B) \geq g(A+x)-g(A) \qquad \text{hypothesis}$$

$$\sum_{\substack{i,j\in B+x\\ i\neq j}} c_{ij} - \sum_{\substack{i,j\in B\\ i\neq j}} c_{ij} \geq \sum_{\substack{i,j\in A+x\\ i\neq j}} c_{ij} - \sum_{\substack{i,j\in A\\ i\neq j}} c_{ij} \qquad \text{definition}$$

$$\sum_{\substack{j\in B\\ j\neq x}} c_{xj} + \sum_{\substack{i,j\in B\\ i\neq j}} c_{ij} - \sum_{\substack{i,j\in B\\ i\neq j}} c_{ij} \geq \sum_{\substack{j\in A\\ j\neq x}} c_{xj} + \sum_{\substack{i,j\in A\\ i\neq j}} c_{ij} - \sum_{\substack{i,j\in A\\ i\neq j}} c_{ij} \qquad \text{expanding sums}$$

$$\sum_{\substack{j\in B\\ x\neq j}} c_{xj} \geq \sum_{\substack{j\in A\\ x\neq j}} c_{xj} \qquad \text{simplifying}$$

$$\sum_{\substack{j\in B\setminus A\\ x\neq j}} c_{xj} + \sum_{\substack{j\in A\\ x\neq j}} c_{xj} \geq \sum_{\substack{j\in A\\ x\neq j}} c_{xj} \qquad A\subseteq B$$

$$\sum_{\substack{j\in B\setminus A\\ x\neq j}} c_{xj} \geq 0$$

which holds because, by definition, $c_{ij} \geq 0$ for all $i, j \in \mathcal{T}$, $A \subseteq B \subseteq \mathcal{T} \setminus x$, and $x \in \mathcal{T}$.

□

Finally, we can combine the previous three lemmata to show that our utility function is submodular and non-negative.

Theorem 5.2.4. *The utility function* $f_a : 2^{\mathcal{T}} \to \mathbb{R}^+$ *for each of our UAVs* $a \in \mathcal{A}$*, defined as:* $f_a(S) \triangleq \Pr_{srv,a}(S) \sum_{j\in S} w_{aj} - \lambda_a \sum_{\substack{i,j\in S\\ i\neq j}} c^a_{ij}$ *is submodular and non-negative.*

Proof. To prove submodularity we will use the fact that the sum of submodular functions is also a submodular function. First, from Lemma 5.2.2 we have that $\Pr_{\text{srv},a}(S) \sum_{j\in S} w_{aj}$ is submodular and non-negative. Second, from Lemma 5.2.3, we have that $\sum_{\substack{i,j\in S\\ i\neq j}} c^a_{ij}$ is supermodular and non-negative, hence, $-\sum_{\substack{i,j\in S\\ i\neq j}} c^a_{ij}$, is submodular. Therefore, $f_a(S) \triangleq \Pr_{\text{srv},a}(S) \sum_{j\in S} w_{aj} - \lambda_a \sum_{\substack{i,j\in S\\ i\neq j}} c^a_{ij}$ is submodular.

Finally, f_a is non-negative because $\Pr_{\text{srv},a}(S) \sum_{j\in S} w_{aj}$ is non-negative and λ_a is defined such that the penalty is never higher than the value obtained discounted by the probability of survival, i.e. $\Pr_{\text{srv},a}(S) \sum_{j\in S} w_{aj} \geq \lambda_a \sum_{\substack{i,j\in S\\ i\neq j}} c^a_{ij}$ for all missions $S \subseteq \mathcal{T}$, which ensures that f_a is non-negative. □

5.3 Numerical Experiments

In this section we evaluate the performance of our algorithm with the utility function presented above and compare it against the state of the art in decentralised task allocation, Consensus-Based Bundle Algorithm (CBBA) [19]. We first explain the procedure to generate synthetic instances and then present the result of the numerical experiments.

5.3.1 Synthetic Instance Generation

In order to asses the performance of our algorithm, we needed to generate a variety of instances of our surveillance mission problem. Here we explain how we generate these synthetic scenarios.

For each task we need to generate an importance q_j and a requirements vector $\mathbf{v}_j$. To generate the importance for each task, we draw a random number from a uniform distribution between 0 and 1. And to generate the requirements vector for each task, we draw a uniformly random unit vector in $\mathbb{R}^3_+$.

Now for each UAV $a \in \mathcal{A}$ we need to generate an a priori detection probability P_{0_a}, an α_a parameter, and a capability vector $\mathbf{u}_a$. Similarly as we do with the tasks, to generate the capability vector we draw a uniformly random unit vector in $\mathbb{R}^3_+$. To find the parameters of the probability of detection for each UAV $a \in \mathcal{A}$, we first start by drawing a random α_a from a uniform distribution between 1.5 and 2.5. Then we draw a random number, x_a, from a uniform probability distribution between 0.3 and 0.7 and we set it to be the total probability of detection, i.e. $\Pr_{\text{detect},a}(|\mathcal{T}|) = x_a$. Subsequently, we use a simple bisection method to find which value of the a priori detection probability P_{0_a} would satisfy $\Pr_{\text{detect},a}(|\mathcal{T}|) = x$, given α_a. This ensures that in a hypothetic mission where UAV $a \in \mathcal{A}$ would have to carry out all the tasks in $\mathcal{T}$, the chances of a being detected -and not surviving- are between 30% and 70%. This approach for the selection of α_a and P_{0_a} gives us a broad spectrum of "detection dynamics" among the UAVs.

5.3.2 Results

Let us now describe the set of experiments that we have carried out. We have chosen the following range of problem sizes: 30 Tasks and 5 UAVs, 50 Tasks and 9 UAVs, 67 Tasks and 17 UAVS, and, finally, 100 Tasks and 20 UAVs. This range captures problem up to a medium sized multi-UAV system. For each problem size we have generated 10 random synthetic instances following the method described above. For each instance, we have run our algorithm with 10 logarithmically-spaced ϵ values between 0.1 and 0.001. For larger systems of UAVs, e.g. swarms, with hundreds of vehicles our algorithms would result impractical, given their asymptotic complexity. Further work is definitely granted in finding algorithms with similar approximation ratios but with a better running times.

Let us first take a look at how our at the impact of ϵ in the performance of our algorithm (Algorithm 10). Naturally, different instances of the same size have different values of $F(\mathbf{y}(1))$, i.e. the relaxation solution. Therefore, to ease the comparison between them, we have normalised it with respect to the value of the relaxation result obtained with the smallest ϵ. In Figure 5.1 we present the value of the solution obtained for each value of ϵ for each problem size. In this figure we can see that the lines are almost flat, that is, the mean of the different problems of the relaxation value is not greatly influenced by ϵ. While the spread in the relaxation values among problems seems greatly affected by ϵ. This seems a natural behaviour because a large ϵ forces the algorithm to take large steps in directions that, by chance, can be in a favourable or unfavourable direction, increasing the spread. Whereas with a small ϵ each step in the evolution of the relaxation is closer to the validity boundaries of

the threshold-greedy solution, and consequently, the algorithm is more likely to advance over the best direction, which irrespective of ϵ. Another noteworthy feature is that the spread tends to be towards the larger ϵ values. This is in line with the theoretical analysis, Lemma 4.2.4, which suggests that the solution bound with respect to the optimal should reduce as ϵ decreases approaching the ratio of $\frac{1}{e}$ in the limit $\epsilon \to 0$. An additional trait that we would like to note is that the spread of the solution value decreases as the size of the problem increases. We believe this is because as number of tasks and agents increase the number of allocation options increases. Hence, with a larger pool of candidate agents for a given task, the difference between the best and second best alternative agent in terms of marginal value tends to be smaller, and the large steps induced by a large ϵ tend to become less punishing.

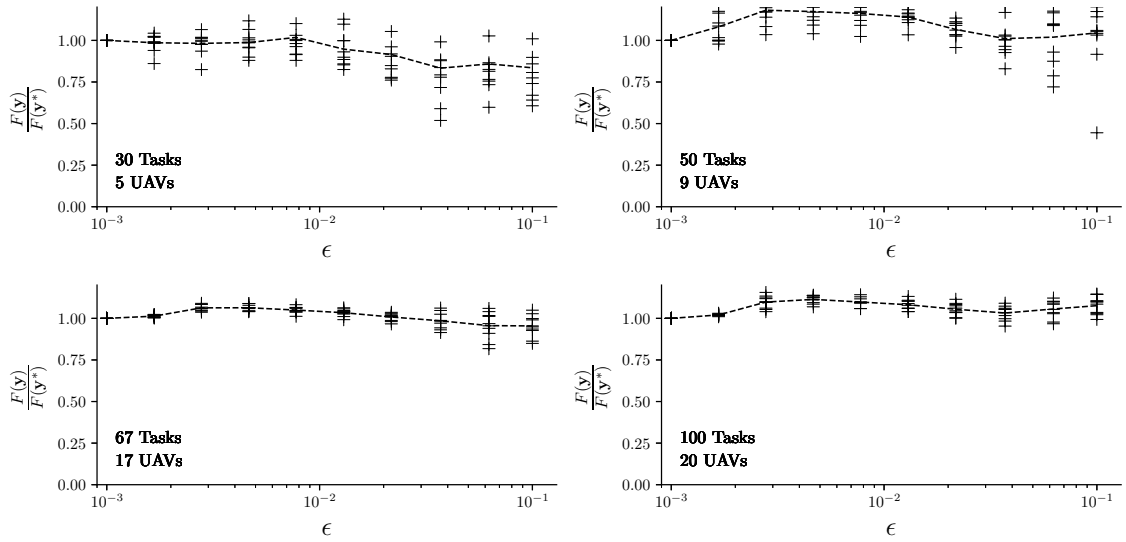

Figure 5.1: Normalised relaxation values of the solutions of Algorithm 10 for a sweep of the parameter ϵ. Each + represents a problem instance and the dashed line joins the means for the same ϵ. *Note:* $\mathbf{y}^*$ *is the relaxation obtained with the smallest* ϵ*, i.e. with* $\epsilon = 0.001$.

Let us now compare the results of the relaxation and the rounded solution with the state of the art in decentralised task allocation with approximation guarantees: the Consensus-Based Bundle Algorithm (CBBA) [19]. We will study:

- the relaxation value $F(\mathbf{y})$ (Algorithm 10),
- the rounded solution value $f(S^*)$ (Algorithm 12), and
- CBBA [19].

To do the comparison, we ran our algorithm with an ϵ around the middle of the studied range, $\epsilon = 0.0078$. And, to round the solution we used 1000 samples distributed among the agents. We present the results in Figure 5.2. In this figure we can see that the value of the rounded solution and relaxation are

always superior to the CBBA. In fact, we can observe that while the value of the rounded and the relaxation solutions seem to increase with the number of tasks, while CBBA's solution value seems to increase only with the number of agents. This suggests that our algorithm is able to find solutions that utilise most of the available tasks while CBBA seems to get trapped in local optima with a few tasks per agent. Our utility function is non-monotone, and the value of the optimal solution -loosely speaking- scales linearly with the number of tasks. Hence, it is only consistent that our algorithm also scales linearly with the number of tasks, since its solution value is bounded to the optimal by the constant factor $\frac{1}{e}$. While CBBA is, in essence, a Greedy algorithm, which only has constant factor guarantees for monotone submodular functions and, by contrast, it may exhibit arbitrarily poor performance with non-monotone utility functions. Which is possibly the reason why CBBA has a poorer performance. Another noteworthy feature is that the gap between the rounded solution and the relaxation seems to be remain constant around the same order of magnitude, suggesting that will become less and less relevant as the problem size increases. Finally, this also suggest that the value of the solution derives mostly from the relaxation rather than by the process of rounding, as it is expected from the theoretical analysis of the previous chapter.

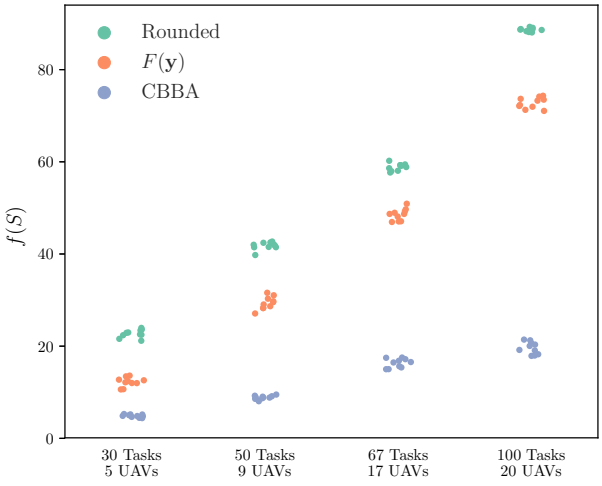

Figure 5.2: Comparison of the solution values.

Finally, we present the running time of our algorithm compared with CBBA in figure 5.3 with $\epsilon = 0.0078$. These experiments have been run using a Python 2.7 implementation of both algorithms in a Mid 2013 MacBook Air with an Intel i7 1.7 GHz processor and 8Gb of RAM, running OSX 10.11.6, running both algorithms sequentially in a single core without any parallelism. As we can see, our algorithm is about three orders of magnitude slower than CBBA. On the positive side, we see that the observed slopes of asymptotic complexity are quite similar for both algorithms, suggesting that the main drawback is the constant. We believe this is because, in essence, our algorithm takes $O(\frac{1}{\epsilon^2})$ more opera-

tions than CBBA in local computational costs. Here is where the real trade-off our algorithm lies, we have a slower running time, in exchange for guaranteed performance for non-montone submodular utilities. The algorithms that we present here are a first step towards decentralised non-monotone submodular models for multi-robot systems but there is still a long way to go in terms of algorithmic engineering to optimise them and turn them into a practical reality for real-world scenarios.

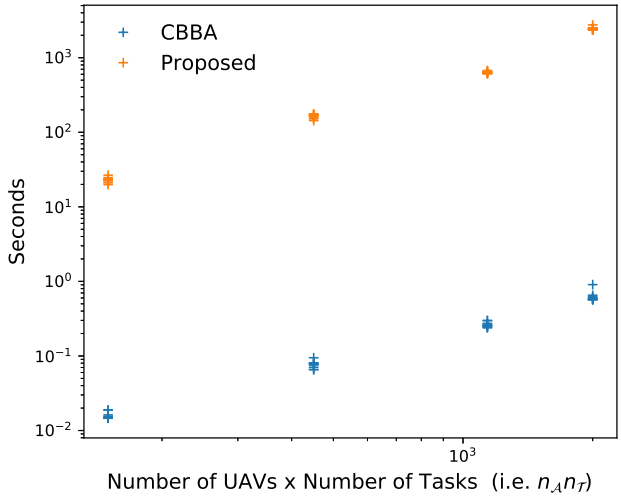

Figure 5.3: Comparison of running times.

5.4 Discussion

Since the utility function model that we have presented is quite flexible and can accommodate a wide range of scenarios. This is because there are few restrictions imposed on the range of weights and parameters that describe the model, and these are quite natural. Let us describe them in more detail:

- **Task-Agent value**, composed by:
  - q_j **Task importance**: *any positive number.*

    It has a clear and intuitive understanding.
  - m_{aj} **Task-Agent match fitness**: *any positive number.*

    We have chosen the dot product between two vectors so that we could generate a wide variety of values, but it can be any other quantity that describes the adequacy of the UAV to carry out a given task. For example, in a real-world application, the UAVs could be graded for each task in discrete levels by an expert operator, e.g., Perfect, Adequate, or Inadequate, with corresponding values -say 1, $\frac{1}{2}$, and 0 respectively- depending on the adequacy

of the sensor payloads that the UAV carries. Thus, enabling flexibility in adapting the model we present to the real world.

- **Probability of Survival**, composed by:
  - P_0 **Probability of detection**: *Any valid probability.*

    It is simply the probability of being detected at a task without having conducted any tasks previously. It has a clear and intuitive understanding.
  - α **Awareness paramenter**: *Any value* $\alpha \geq 1$ *s.t.* $\frac{P_{0_a}}{1-\alpha_a n P_{0_a}}$ *is a valid probability.*

    Basically, this parameter allows us to decide how fast is the enemy becomes aware of the presence of our UAVs. If we set it to 1 the probability of detection increases linearly with the number of tasks, and as we increase α, the probability of detection grows faster than linear. The only constraints that are imposed on it are, simply, those that keep the numbers in the range of valid probabilities. And the choice between a linear or superlinear increase in the detection probability spans the natural range of behaviour that would be desirable in the real world.

- c_{ij} **Cross-Task penalty**: *any positive number.*

  This parameter allows us to impose any arbitrary penalty between pairs of tasks. We have selected the product of the tasks' importances to discourage concentration of important tasks in order to increase the robustness of the solution. But we can also use it to model any other negative interaction between tasks, for example to avoid radio transmissions from interfering each other in the same frequency.

Therefore we believe that the utility function is flexible and can be adapted to different scenarios where a team of heterogeneous UAVs are used to carry out a set of heterogeneous tasks in a hostile environment.

In light of the numerical experiments that we present, we believe that the constant factor approximation ratios of our algorithm are not just of theoretical interest. Indeed, our algorithm is able to address scenarios that the current state of the art CBBA [19] could not. We believe this is of great importance, because it validates that a new family of utility functions -*non-monotone submodular*- opens to use in decentralised task allocation by researchers and practitioners. CBBA's great success -[19] has 250+ citations as of late 2017- was largely because it enabled decentralised task allocation with constant-factor approximation for non-negative monotone submodular functions. This was in spite of the fact that the monotonicity assumption may be too restrictive in many real world situations -such as the simple survival probability model here.

Nevertheless our algorithm has a limitation in its computational complexity. Hence, the most pressing future work line would be to find ways to keep its approximation guarantees while reducing the computational burden. As discussed in the previous chapter, there are several things that could improve the local computational cost of our algorithm. The first is to sample more selectively,

as we have explained in the previous chapter's section on Local Computational Cost, rather than naively sampling every task-agent pair we could devise more intelligent ways that lead the algorithm to make the same decisions with lower computational costs. The second is to remove sampling all together, by using utility functions that have an efficient way to compute the multilinear extension. The third and final stream would be a fundamental shift, away from relaxation and round approaches, that rely in many small integration steps, towards more combinatorial algorithms while keeping the approximation ratios. This, however, would require fundamentally new algorithms. We believe nevertheless, that the algorithms that we present here prove that it is indeed possible to find such algorithms.

On another note, we believe that a fruitful avenue for future work is to build non-negative non-monotone submodular utility function models that are of practical interest. Let us here briefly explain a few possibilities:

- **Coverage with diversity**. When trying to explore an area the notion of information and coverage is naturally submodular, i.e. as more sensors or UAVs are deployed the marginal utility of adding more diminishes. These models have been used with success in practice: [53, 87, 88]. However, these use monotone submodular models for information gathering that have been shown to be afflicted by excessive concentration, or lack of diversity [65]. To remedy this several submodular models have been proposed [23, 65, 94], naturally these functions are non-monotone, because they penalise excessive concentration. These works are in the context of document and image summarisation, and therefore, we believe that there is a prosperous avenue of research in studying and adapting these functions to search, surveillance, or exploration missions, now that the algorithms presented here enable their application in decentralised task allocation.

- **Tasks allocation under risk of agent failure**. We believe that a non-monotone model is only natural when considering a task allocation scenario where the agents can fail at their execution of their tasks. One example is the model presented here, where the agents (UAVs) can be detected and thus fail their mission. Another example is with a multi-robot team executing complicated manipulation functions, say in rough terrain, where completing each task is risky because the robot may suffer some difficulty, such as getting stuck, and thus fail the mission. In this situation, a single robot may well be the best suited to carry out all the required tasks, but a solution where an individual robot carries all the tasks would be undesirable because of the high risk of failure that it would involve. Monotone submodular functions are structurally ill-suited to model such scenarios, because, by definition, they do not contemplate a reduction in value of an excessively large number of tasks. Therefore, the algorithms presented here give ability to researchers to devise more suitable non-monotone submodular models, opening a fertile line of research.

- **Enemy Network Jamming**. Jamming enemy networks is an important part of modern warfare. These operations usually require a coordinated action a set of distributed assets behind enemy lines, and thus it is obvious use case for a team of small UAVs or robots. The goal is usually to partition a radio communications network [76]. Therefore, a graph cut

model, or a combination thereof, is a very natural way to formulate the problem [33]. Graph cut functions are non-monotone submodular, and so, our algorithm is well suited to solve problems that embody them. Hence, we think that an interesting research line opens in this direction.

- **Aerial Firefighting**. Aerial firefighting is a vital tool to combat large forest fires, and it is among the most dangerous missions that a civilian pilot can undertake [1]. It is another area where, in the future, UAVs could deliver immense benefits, removing humans from danger and operating in conditions where is currently not possible with a human pilot, e.g.: at night time, or with limited visibility. One example of such a type of UAV is described in the patent application by J. Moore [74]. These missions are usually carried out in groups of heterogeneous aircraft because there is an advantage in the persistent application of firefighting chemicals and in a geographically distributed attack. Therefore, it is an ideal application for task allocation in a team of UAVs. When an aircraft attacks more than one fire point the pressure in the tank drops as the chemical is delivered, and the effectiveness is reduced. Furthermore, the attack is altogether futile if insufficient firefighting chemical is delivered in a fire front. Therefore, a non-monotone function with diminishing returns seems a viable way to model the problem, providing another application area where our algorithm could be effective.

These are only a few future research lines, but we believe they serve to illustrate the broad range of possibilities that non-monotone submodular models open.

Chapter 6

A Combinatorial Auction Framework For Decentralised Task Allocation

In the previous chapters we have presented algorithms to solve the task allocation with submodular objective functions with approximation guarantees, here we take a different point of view and surrender approximation guarantees in exchange for a more general problem and more flexible framework. As we have seen, the task allocation problem is NP-Hard in all but its simplest incarnations [35], [52]. However, there are tools in the Operations Research literature that enable the centralised solution of particular instances of practical interest. For example, Vehicle Routing problems with hundreds of tasks are solved daily by delivery businesses [93]. This is because there are a host of techniques, e.g. Metaheuristics, Mixed Integer Programming, or Constraint Programming, that over many decades of algorithmic research and engineering have been adapted and tailored for specific problems, reaching the point where they can solve large instances efficiently.

However, the centralised solution of the problem involves having to communicate all the agents and environment data to a centralised entity. This may not be the most appropriate approach for some scenarios because the central entity removes resilience by introducing single point of failure, or the bandwidth to communicate all the information to the central entity may not be available. In light of this, the present chapter focuses on developing a framework that enables the transfer of successful centralised solution approaches from the Operations Research domain to decentralised task allocation. That is, our framework enables the use of Metaheuristics, Mixed Integer Programming, or any other of the techniques that have had much practical success, in the setting where each agent has its own utility function that is not known by the rest of the agents in the team.

In this chapter, therefore, our contribution is a preliminary discussion of a flexible and general decentralised task allocation framework that enables the use

of tailored solvers in the setting where agents do not have access to each other's utility functions. We do not assume that the problem is neither a maximisation nor a minimisation, and we do not assume anything on the objective function other than its non-negativity. Of course, this flexibility and generality comes at a price: the formulation reduces to an NP-Hard problem. However, in practice the flexibility in the tailoring of specific solvers can be very yield very efficient algorithms. Thus, the computational complexity of our framework depends on the specific solution technique used. On the other hand, we show that the communication complexity is linear in the number of tasks and in the number of agents. Finally, to validate our framework, we show some promising numerical results with routing cost functions of practical interest.

6.1 Problem Definition

A widely accepted taxonomy that maps the nature of each problem to a well known combinatorial problem was introduced in [35] and was extended to support task dependencies in [52]. These two works provide a map of the task allocation problem space. In this taxonomy, the framework we propose aims to tackle problems with: both Single-Task robots (ST) and Multi-Task robots (MT); Single-Robot tasks (SR); and both Instantaneous Assignment (IA) and Time Extended Assingnment (TA). With respect to the task dependencies taxonomy in [52] the framework proposed in this work can accommodate: No Dependencies, In-Schedule Dependencies (ID) and some instances of Cross-Schedule Dependencies (XD). Adapting the classical definition, from [27], we can formalise the general Task Allocation Problem as follows:
Given a set of tasks $\mathcal{T}$, a set of agents $\mathcal{A}$, and a function for each agent $a \in \mathcal{A}$ specifying the utility of completing each subset of tasks $c_a : 2^{\mathcal{T}} \to \mathbb{R}^+$, find a spanning and non-overlapping allocation, $\mathcal{S}^* \in \mathcal{A}^{\mathcal{T}}$, that minimises/maximises a global objective function $\mathcal{J} : \mathcal{A}^{\mathcal{T}} \to \mathbb{R}^+$.

This is the most general formulation of the task allocation problem, with the only constraint that all tasks must be allocated. Note the difference with the previous chapters, where we could have some tasks that were not allocated. Here we require that all tasks are allocated, but without any other additional constraints. This means that all the task allocation problems that require all the tasks to be allocated can be reduced to our framework. However, additional constraints, such as scheduling, or task grouping constraints, are not supported. This problem definition is specially suitable to capture routing problems that are important in multi-robot or multi-uav missions.

Due to the nature of our framework, it is advantageous to formulate the problem as a Mixed Integer Programming Problem. To this end, we define the objective function $J : (\mathbb{R}^+ \times \{0,1\})^{|2^{\mathcal{T}}| \cdot |\mathcal{A}|} \to \mathbb{R}^+$, that maps the performance metrics of each bundle of each agent and the allocation variables to a positive real number quantifying the allocation cost or score. Each agent $a \in \mathcal{A}$ has its own utility function $c^a : 2^{\mathcal{T}} \to \mathbb{R}^+$. Different agents might have different payloads and/or capabilities as well as different information about the tasks, and there is no requirement for this information to be shared: c^a is an entirely local and it is based only on the information available to each individual agent. We also define a binary decision variable x_b^a, that has a value of 1 if bundle $b \in 2^{\mathcal{T}}$ is allocated to agent $a \in \mathcal{A}$ or 0 if otherwise. Now, we can set out the Mixed

Integer Formulation of our problem:

$$\begin{aligned}
&\underset{\text{over all } x_b^a}{\text{optimise}} && J((c^a(b), x_b^a), ...),\ \forall a \in \mathcal{A},\ \forall b \subseteq \mathcal{T} \\
&\text{subject to:} && \\
& && y \cap z = \emptyset,\ \forall y, z \in \mathcal{S},\ y \neq z \\
& && \bigcup_{b \in \mathcal{S}} b = \mathcal{T} \\
& && x_b^a \in \{0, 1\},\ \forall a \in \mathcal{A}, \forall b \subseteq \mathcal{T} \\
& && \mathcal{S} = \{h | a \in \mathcal{A}, h \subseteq \mathcal{T}, x_h^a = 1\}
\end{aligned}$$

where $\mathcal{S}$ represents the set of allocated bundles, while the first constraint means that no two allocated bundles can have overlapping tasks, and the second constraint means that the union of all allocated bundles must be equal to the tasks set $\mathcal{T}$. In other words, all tasks should be allocated and each exclusively to one agent.

To illustrate this formulation, an example of a problem that could be casted into our framework could be the Multiple Traveling Salesman Problem, mTSP, with the tasks being cities or waypoints, the agents performance metrics being the distance traveled to visit a given bundle of cities, the objective function as the sum of the distances travelled in the allocated bundles, and the optimisation being a minimisation. The advantage of our framework is that it would enable the use of fast and efficient TSP solvers in the agents, i.e. run locally, while the communication exchange between them is be polynomial.

6.1.1 Background

In this chapter, we aim to find task allocation framework that remove the submodularity restriction on the objective function that CBBA and the Smooth Continuous Greedy algorithm that we have presented in the previous chapters require, while we keep the advantages of a decentralised model. To understand the rationale behind our framework, let us first look at CBBA [19]. At its essence, CBBA leverages a simple auction model: agents add tasks to their bundles and send single task bids with the marginal value of each of the tasks to the auctioneer. The auctioneer collects these bids, and simply awards the task to the agent whose marginal gain is the highest. Given such simple auctioneer mechanism, it can then be naturally translated into a set of consensus rules and make the algorithm decentralised. The key lesson from its design is: find a framework with a very efficient auctioneer role and then substitute it by a polynomial time consensus rule to achieve natural decentralisation. Unfortunately, combinatorial auctions are characterised, precisely, by a very hard auctioneer role, in fact the general Winner Determination Problem is NP-Hard [100]. To get around this, in this research we use a framework called the Progressive Adaptive User Selection Environment (PAUSE) conceived for government auction of telecom licenses [50]. It was intended as a way to simplify the auctioneer role in combinatorial auctions, with the aim that all companies involved could easily verify the decisions being taken and could understand them, guaranteeing the transparency and fairness of the final decision.

Naturally, the computational burden cannot be avoided. The computational load is not removed but rather transferred from the auctioneer to the bidder,

i.e. the agents. This is accomplished by requiring them to submit a composite bid that encompasses not only their own bundles but also those of other agents, in such a way that all the tasks are assigned, and each task is assigned to only one agent. Hence, the role of the auctioneer is simply to keep a record of the composite bids submitted and award the tasks to those agents contained in the composite bid whose payoff is the highest. With such an efficient role for the auctioneer, the decentralisation of the algorithm is natural with consensus rules.

Let us now give an intuitive description of the PAUSE mechanism following that of Land et al [59]. A PAUSE auction for m items is a multi-stage auction of m stages. In each phase, each agent must send to the auctioneer a composite bid that covers all the items. In the first stage, each agent submits a composite bid composed only of single item bids of its own (it does not have access to the other agents utility functions). In the second stage, the auctioneer broadcasts the bids from the first round and each agent submits a composite bid based only on a combination of the single item bids (submitted by all agents at stage 1) and two item bids of their own. In the third stage, the auctioneer broadcasts the bids from the second round and each agent submits a composite bid based on a combination of one item and two item bids (submitted by all agents at stage 1 and 2) and their own three items bids. The process continues and, in the general nth stage, each agent submits a composite bid that is a combination of the previously (1, 2 ... $n-1$)-items bids, submitted in the previous rounds by all the agents, and the agents own $n$item bid. In each round, each agent can either submit a bid, if it improves the current best composite bid by a design threshold, or just wait and listen to the other agents if it cannot improve it. Note that with this mechanism, if an agent improves the bid in the final round, it will be because it is submitting a composite bid that contains only a single bid that spans all the items.

With this mechanism, the job of the auctioneer simply is: to record all the bids with their bundle valuations and broadcast them to the agents, so that the agents can use them in their following composite bids. The winners then, are simply determined by the agents contained in the composite bid with the highest score. This mechanism was devised to improve the sense of fairness and transparency in government licensing by providing a very efficient auctioneer role, i.e. one whose decisions are efficiently verified. It is this that makes the auctioneer naturally decentralisable through consensus rules.

Leveraging the key ideas in CBBA and PAUSE, in this chapter we outline a preliminary proposal of a decentralised framework which finds good allocations for a broad set of unconstrained task allocation problems. We remark that we present *framework* rather than a specific *algorithm*, because it can implemented using many different techniques from the Operations Research arsenal.

6.2 Decentralised Task Allocation Framework

Now, we describe the proposed framework formally. But first, we shall introduce some notation used in the exposition. Recall that we defined a function $c^a : 2^{\mathcal{T}} \to \mathbb{R}^+$, thus $c^a(b) \in \mathbb{R}^+$ is the utility (or cost) of agent $a \in \mathcal{A}$ executing the bundle of tasks $b \in 2^{\mathcal{T}}$. Examples of this utility functions could be simple like flying time or more elaborate ones such as information collected, or a combination of agent dependent rewards and costs, among other things.

A bid B_b^a is a tuple $(b, a, c^a(b)) \in 2^{\mathcal{T}} \times \mathcal{A} \times \mathbb{R}^+$ that contains, respectively, the set of tasks involved $b \subseteq \mathcal{T}$, the agent who is responsible $a \in \mathcal{A}$, and its score or utility $c^a(b) \in \mathbb{R}^+$. The evaluation of the utility (cost) function to create bids is fully local and is produced by each individual agent and shared as part of a composite bid. For example, consider the simple case where the performance index were the fligh t time and the tasks waypoints to visit. Then, a bid $(b, a, c^a(b)) \in 2^{\mathcal{T}} \times \mathcal{A} \times \mathbb{R}^+$ would be computed locally by agent $a \in \mathcal{A}$. Agent a, in this case, would go about this by calculating that the time (in, say, seconds) needed to visit each task contained in b from its position is t_b (that is, it would evaluate $t_b = c^a(b)$ in our notation). Subsequently, in order to inform other agents about its capability to perform the tasks in $b \subseteq \mathcal{T}$ (i.e. visit the waypoints in b) it would aggregate this information with the task set b and its own identifier a, producing the bid (b, a, t_b). Hence, all the other agents in the set $\mathcal{A} \setminus \{a\}$ would "know" about the performance of agent a visiting the tasks in b is that it takes a time given by $t_b \in \mathbb{R}^+$, rather than being able to evaluate a's utility function themselves.

Now, a composite bid, CB_k^a is the solution to the bidding problem at stage k by agent $a \in \mathcal{A}$, and it is the set of bids that spans all the tasks and does not contain overlapping bids. CB_k^a is calculated optimising the objective function $J : (\mathbb{R}^+ \times \{0, 1\})^{|2^{\mathcal{T}}| \cdot |\mathcal{A}|} \to \mathbb{R}^+$. That is, finding the best combination of decision variables x_b^a from the bids that the agent has received and its own bids of k or fewer tasks. A composite bid is what agents exchange among themselves, and it is the result of each agent solving the allocation problem at each stage with the information available at that point in time to each of them.

The information that an agent $a \in \mathcal{A}$ keeps about the other agents is kept in a set S_k^a that contains all the bids in the composite bids previously exchanged by other agents in the stages $1, 2, ..., k-1$, more formally, $S_k^a = S_{k-1}^a \cup (\bigcup_{\substack{i \in \mathcal{A} \\ i \neq a}} CB_{k-1}^i)$ with $S_0^a = \emptyset$. This set S_k^a is used in conjunction with the agent's own bids to find the subsequent solution to the allocation problem to produce the composite bid CB_k^a that is exchanged with fellow agents.

A set D_k^a is kept by agent $a \in \mathcal{A}$, containing the bids exchanged previously by other agents and any of its own that have a size smaller or equal to the corresponding bid stage, i.e. $D_k^a = S_k^a \cup \{(b, a, c^a(b)) | \forall b \subseteq \mathcal{T}, |b| \leq k\}$. Intuitively D_k^a is the pool from which agent $a \in \mathcal{A}$ draws bids (i.e. sets) to find the optimal allocation resulting in the the composite bid of stage k.

With all this, the bidding problem for agent $i \in \mathcal{A}$ at stage k is: given a set of bids D_k^i find a non overlapping set of bids (i.e. a composite bid) that contains each task in $\mathcal{T}$ exactly once and optimises the objective function of the allocation. This problem can be formulated as:

$$\begin{aligned} \text{optimize} \quad & J((c^a(b), x_b^a), ...) \ \forall (b, a, c^a(b)) \in D_k^i \\ \text{subject to:} \quad & \\ & y \cap z = \emptyset, \ \forall y, z \in \mathcal{S}, \ y \neq z \\ & \bigcup_{b \in \mathcal{S}} b = \mathcal{T} \\ & x_b^a \in \{0, 1\} \ \forall (b, a, c^a(b)) \in D_k^i \\ & \mathcal{S} = \{h | (h, a, c^a(h)) \in D_k^i, x_h^a = 1\} \end{aligned}$$

Having defined the bidding problem, we can formally define a composite bid as a set CB_k^i that contains all the bids that are selected in the solution of the bidding problem that each agent i solves in the stage k, i.e. $CB_k^i = \{(b, a, c^a(b))|(b, a, c^a(b)) \in D_k^i, x_b^a = 1\}$, where the x_b^a variables are the optimisers of the bidding problem above. We will assume that agent $i \in \mathcal{A}$ solves the bidding problem in round k by calling the routine $\texttt{ComputeBid}\big(S_k^i, c^i, k\big)$. In practice this routine must be adapted to each individual problem with its performance metrics and objective function. Here is where the great advantage of our framework lies: the solution method to the bidding problem is tailored to a specific instance, and thus it can leverage Metaheuristics, MIP Solvers, CP Solvers, or any other algorithmic tools that performs well for the problem at hand.

With these definitions we can now outline the full framework. Initially the agents have a copy of the set $\mathcal{T}$ containing all the tasks that must be allocated and the set $\mathcal{A}$ of all the participating agents in the network. The framework proceeds in $k \in \{1, 2, ..., |\mathcal{T}|\}$ stages. In the first round each agent i has not received any information from the other agents hence S_1^i is empty and since $k = 1$, D_1^i contains only bids of agent i itself, thus there can only be$|\mathcal{T}|$ bundles and consequently there can be only one possible solution to the bidding problem: CB_1^i contains single task bids for each of the tasks in $\mathcal{T}$. At stage $k = 2$ agent's i bid-set S_2^i contains all the single task bids exchanged at the end of round $k = 1$ for each of the tasks in $\mathcal{T}$ by each of the agents in $\mathcal{A}$ and finds the best allocation among them and 2-task bids of its own. The process continues until $k = |\mathcal{T}|$ when finally the best allocation is the composite bid with the best objective. The routine for an agent $i \in \mathcal{A}$ is outlined in Algorithm 13.

6.3 Discussion

We have developed this algorithm bearing two main objectives in mind: to be decentralised with a communication overhead comparable with the state of the art; and to deal with problems that do not meet the assumptions in the objective functions, such as submodularity, that current state of the art algorithms require. The key philosophy has been to trade constant factor approximation guarantees in exchange for flexibility. In light of the practical success of many centralised approaches to NP-Hard problems, we have designed our framework in such a way that it can accommodate any solution method for the bidding problem. In other words, our framework provides a way to transfer successful approaches to solving NP-Hard problems to the decentralised setting.

Now, let us make some remarks on the locality of the valuation functions and its implications. Bundle valuations by each agent do not change over the execution of the algorithm. It is assumed that once an agent has shared a bid $(b, a, c^a(b))$ in a composite bid, the valuation $c^a(b)$ that agent a makes of bundle b does not change in subsequent rounds. This is a reasonable assumption as it only requires to have differentiated time scales for the allocation procedure and for the mission execution. This is usually the case in practice because the task execution implies that the agent must travel to different spatial location taking much longer than the allocation procedure.

The valuation of the bids by each agent $a \in \mathcal{A}$, namely, the functions $c^a : 2^{\mathcal{T}} \to \mathbb{R}^+$, is fully local. That is, given a bundle of tasks $b \in 2^{\mathcal{T}}$, only agent a can

Algorithm 13: Combinatorial Auction Task Allocation Framework

Input : Set of agents $\mathcal{A}$; set of tasks $\mathcal{T}$; local agent i, local utility function $c^i : 2^{\mathcal{T}} \rightarrow \mathbb{R}^+$; and access to comms to send and receive bids.

Output: A non-overlapping allocation CB^*

for $a \in \mathcal{A}$ *if* $a \neq i$ **do**
  $CB_0^a \leftarrow \emptyset$
$S_0^i \leftarrow \emptyset$
for $k \in \{1, 2, ..., |\mathcal{T}|\}$ **do**
  `// receive bids from the previous round`
  for $a \in \mathcal{A}$ **do**
    if $a \neq i$ *and* $k > 1$ **then**
      $CB_{k-1}^i \leftarrow$ `ReceivedBid`$(a, k-1)$
  $S_k^i \leftarrow S_{k-1}^i \cup \bigcup_{\substack{\forall i \in \mathcal{A} \\ a \neq i}} CB_{k-1}^a$
  `// compute the new bid and send it`
  $CB_k^i \leftarrow$ `ComputeBid`$\big(S_k^i, c^i, k\big)$
  `SendBid`$\big(CB_k^i\big)$
`// receive the bids from the last round`
for $a \in \mathcal{A}$ *if* $a \neq i$ **do**
  $CB_{|\mathcal{T}|}^a \leftarrow$ `ReceivedBid`$(a, |\mathcal{T}|)$
`// return the best available`
$CB^* \leftarrow \text{OPT}_{a \in \mathcal{A}} CB_{|\mathcal{T}|}^a$
Return: CB^*

compute the valuation $c^a(b)$ of the bundle b, all the other agents in $\mathcal{A} \setminus \{a\}$ only know the valuation given by agent a in the bid, not the valuation function, and cannot infer the valuation for other bundles that have not been previously shared by a. This locality is fundamental to enable cooperation of both heterogeneous agents with different capabilities and different levels of situational awareness. This is because each agent can value locally in their function $c^a : 2^{\mathcal{T}} \rightarrow \mathbb{R}^+$ whether they have enough information to perform a task, whether they have the payload needed, whether they can fly fast enough to get to the task in time etc. This enables the network controller to trade off between delaying the start of the allocation procedure to spend more time exchanging situational awareness in order to have more accurate valuations and tolerating some level of situational awareness discrepancy in exchange of an earlier start of the allocation procedure.

Now, we discuss briefly the computational complexity and the communication complexity of our framework. Then, we present preliminary numerical experiments comparing it against CBBA with three representative objective function that are common in multi-robot routing problems: MinSum, MinMax, and MinAve.

6.3.1 Communication Complexity

In this framwork it is assumed that all the agents are in a connected network and, hence, are able to receive the composite bids of all the other agents. Let D_m be the network diameter, in our framework each agent exchanges $|\mathcal{T}|$ composite bids of size $O(\mathcal{T})$, therefore the number of messages exchanged is $O(|\mathcal{A}| \cdot |\mathcal{T}| \cdot D_m)$. Each message's size is proportional to the number of tasks $|\mathcal{T}|$. Thus, its communication complexity scales asymptotically equal to that of CBBA. We do not describe which specific protocol the agents use to exchange bids, because this is application dependent, and the framework is open to accommodate any variant of a wave or mesh protocol. The reader can see [34, 68, 92] for further insight into the topic.

6.3.2 Computational Complexity

The computational complexity of the framework is determined by the bidding problem which is, naturally, NP-Hard because it can be reduced to an instance of the task allocation problem. And solving the task allocation problem has been shown to be NP-Hard [35], [27]. Thus, solving this problem is what takes most of the computational effort. In the formulation we have outlined in the previous section the problem has, because of its generality, an exponential number of variables and constraints. In reality this is not necessarily the case because depending on the specific utility functions it can be reformulated as an adapted version of a well researched problem, where efficient formulations or algorithms are available. For practical problems, there is a myriad of methods that solve NP-Hard problems fast enough to be useful, some examples are: traveling salesman, set packing, or scheduling problems (to name but a few). If the methods to solve these problems can be adapted by including an extra constraint on the bundle (set) size to match the stage, and by incorporating the solutions of the other agents, then our framework can perform as efficiently as them. This is the key idea in this chapter: we have given up theoretical guarantees in exchange for practical flexibility. Therefore, the framework's complexity depends

on the specific method used to solve the bidding problem, and we believe that, for practical problems, there are algorithmic tools that do have a satisfactory performance in practice.

6.3.3 Numerical Results

In order to explore the performance of our algorithm in this section we carry out some preliminary tests with interesting objective functions. We have chosen the widely studied field of multi-robot routing, that is, how to find optimal paths for a group of robots. Given a set of agents and locations to visit (tasks), there are three main objective functions that can be considered the eigenvectors of the performance metrics used in routing problems:

- **MinSum**: minimises the total sum of the costs (distances) over all agents;
- **MinMax**: minimises the maximum cost (distance) incurred by the agents;
- **MinAve**: minimises the average cost (distance) incurred by the agents from start to the visit of each location.

As baselines to compare the performance of our algorithm, we take a global optimal solution, to assess in relative terms how good is the solution of our framework, and CBBA because it is the state of the art in decentralised task allocation and has the same asymptotic communication scalability. However, as we have seen CBBA relies on submodularity for its convergence, therefore it may not converge when the objective function is not submodular, to prevent this, here we use the warped extension provided by Johnson et al [47]. We compute the optimal solutions using the commercial MIP Solver Gurobi [39]. And, to solve the bidding problem in each agent we also used Gurobi. We conducted a Monte Carlo simulation with 400 runs for each task number and each objective function to obtain preliminary trends in the solution performance as the size of the problem increases. To this end, we created a random scenario each time by placing 4 agents in uniformly random locations of a 1000m by 1000m area, and we placed the tasks uniformly at random within it. The costs of the agents were the distances travelled by the agents. Once an scenario was solved with the three algorithms, we normalised the objective values of CBBA and our framework with respect to the optimal as follows:

$$sc_{\text{CBBA}} = \frac{CBBA - OPT}{OPT} 100(\%) \tag{6.1}$$

$$sc_{\text{proposed}} = \frac{PROPOSED - OPT}{OPT} 100(\%) \tag{6.2}$$

where OPT, PROPOSED, and CBBA represent the objective scores attained for the optimal, our framework, and CBBA respectively.

We show a summary of the normalised results for each of the objectives in figures 6.1, 6.2 and 6.3 and the means of the relative scores for each case is shown in table 6.1. While the number of tasks is small only from 5 to 9, some conclusions can be extracted regarding the underlying trends. In the MinSum metric, the our framework gave in each and everyone of the cases a relative score of 0%, i.e., it found the optimal solution every time, whereas for the same metric CBBA's performance was gradually degrading as the task number increased. In

the cases of the MinMax and MinAve metric, our framework most of the time found an optimal solution, and while its performance degraded gradually as the task number increased, it did so to a much lesser extent than CBBA. Indeed, in all three objectives we can see that our framework provides significantly better results. This is probably due to the fact that the agents in our framework are able to consider a richer description synergies between the bundles by obtaining good solutions to the bidding problem, whereas in CBBA agent only consider the marginal values.

6.3.4 Conclusions and Future Work

We have presented a preliminary decentralised algorithm for the Multi Robot Task Allocation Problem with communication costs comparable to those of the state of the art such as CBBA [19]. The advantage of our framework is that it provides a way to leverage existing algorithmic techniques in a decentralised setting. This opens the use of a lot of the algorithmic technology available in a centralised setting -Mixed Integer Programming, Constraint Programming, Metaheuristics, etc- where agents can only access their own objective functions, and not that of their peers. The numerical results that we have presented are preliminary but demonstrate that its performance improve over the state of the art (CBBA [19]) with representative routing objective functions: MinSum, MinMax and MinAve. We remark that we present here a *framework* rather than an individual *algorithm* because its key advantage lies in the flexibility in the formulation and the solution of the bidding problem with tailored methods. This flexibility comes at a price: we cannot establish any formal approximation guarantees, nor a polynomial running time. However, solution methods that have performed well in practice with other NP-Hard problems could be adapted to provide a practical computation running time. For example, one could use an off-the-shelf solver to solve the bidding problem with an approximation gap or a time limit. Therefore, given that this approach has performed well for many real-world problems and continues to be a fertile field of in the Operations Research community, we believe that it could be viable alternative in to tackle the decentralised solution of a problem that does not satisfy any conditions (such as submodularity) that enables constant factor approximation algorithms. There are two strands of future work that could yield interesting results. The first is to explore the refinement of the solution once all the rounds have been completed. For example, one could re-run the whole bidding process several times after it has been completed, allowing agents to use in the solution of their bidding problems the bids that have been exchanged in previous rounds. Another idea is, once a whole run with stages $1, ..., |\mathcal{T}|$ has been conducted, to run rounds with a random maximum bundle size allowing agents to use all the information exchanged previously. The second line of future work, would be the systematic study of how different kinds of real-world problems (routing, scheduling, etc.) perform in our framework, and how the current best centralised methods available for their solution can be applied to our framework.

		Task Number				
		5	6	7	8	9
PROPOSED	MinSum	0.	0.	0.	0.	0.
	MinMax	0.	0.08	0.08	0.17	0.34
	MinAve	0.	0.04	0.04	0.09	0.2
CBBA	MinSum	1.94	2.95	3.45	3.68	4.35
	MinMax	9.84	9.65	12.9	12.31	13.36
	MinAve	4.15	3.45	5.01	5.77	5.83

Table 6.1: Mean Relative Scores (%) of our framework and CBBA wrt the optimal.

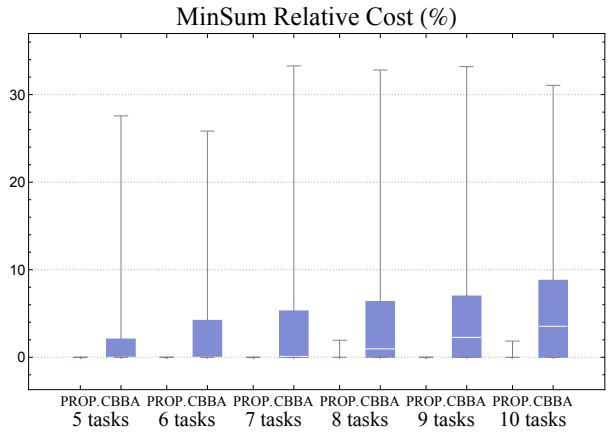

Figure 6.1: Comparison of the distribution of the costs of CBBA using warping functions with the proposed algorithm for the MinSum metric.

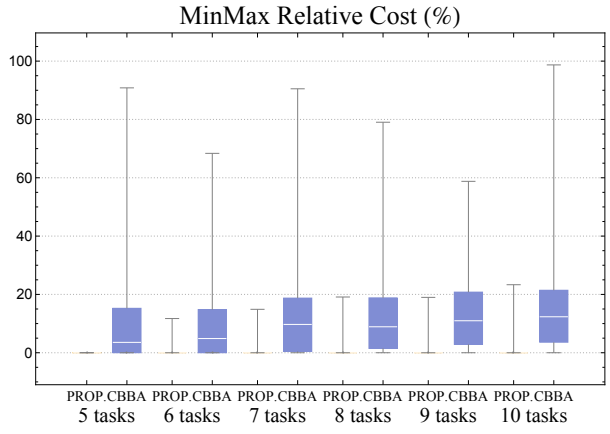

Figure 6.2: Comparison of the distribution of the costs of CBBA using warping functions with the proposed algorithm for the MinMax metric.

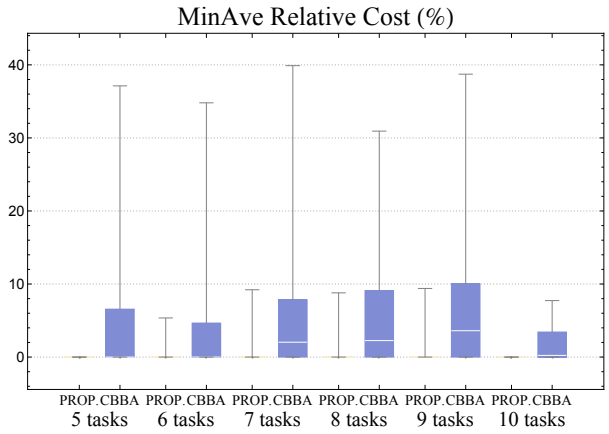

Figure 6.3: Comparison of the distribution of the costs of CBBA using warping functions with the proposed algorithm for the MinAve metric.

Chapter 7

Conclusions and Future Work

In this chapter we collect the conclusions and future research lines that we have stated in each of the preceding chapters.

7.1 Submodular Maximisation

In Chapter 3, we have presented a $\frac{1}{e}-\epsilon$-approximation algorithm for general non-negative submodular function maximisation that requires $O(\frac{nr^2}{\epsilon^4}\left(\frac{\bar{d}+d}{\bar{d}}\right)^2\log^2(\frac{n}{\epsilon}))$ value oracle calls. This is the fastest $\frac{1}{e}$-approximation algorithm currently available, which enables the use of general (non-monotone) matroid-constrained submodular maximisation for many applications for which existing algorithms were implausibly slow. We think this is of great significance, even beyond Task Allocation problems, because there has been a recent surge of interest for applying submodular maximisation in fields where large problem instances are paramount, such as Machine Learning [10, 64, 72], particularly in the field of summarisation where non-monotone submodular functions are natural [23, 94]. Our algorithm is slower than the one presented for the monotone case in [6] by $O\left(r\left(\frac{\bar{d}+d}{\bar{d}}\right)^2\right)$ due to the inability to sample the marginal values of the multilinear extension up to an additive and multiplicative bound. If we could, then we would reduce the additional value oracle calls required to achieve an algorithm with the same running time, we believe this might be possible.

A future avenue of research would be to combine our work with the very interesting results in [13], where an efficient algorithm is proposed to allow the trade-off of value oracle calls and matroid independence calls for non-negative monotone submodular functions, to enable query trade-off for general non-negative submodular functions. Another interesting path is to combine the more continuous-like measured continuous greedy update step that we present here with the acceleration techniques for strong submodular functions presented in [101] to produce an adaptive step algorithm. This way, in each step we could use a large δ that extended to the boundary of the region of validity of the set B, instead of taking a δ that is small enough to satisfy the worst case. Finally, an obvious improvement on the algorithms presented here would be to combine the

ideas from the Lazy Greedy Algorithm [71] to adaptively change the decrement of the threshold in the Decreasing-Threshold procedure.

7.2 Decentralised Submodular Task Allocation

In Chapter 4 we have presented the a decentralised Task Allocation algorithm that provides approximation guarantees for non-negative monotone ($1 - \frac{1}{e} \approx 63\%$) and non-monotone ($\frac{1}{e} \approx 37\%$) submodular functions whilst relying, only, on local utility function calls and neighbour to neighbour communications. We have given a full formal analysis on the approximation guarantees, communications and computational complexity. This is the first decentralised algorithm, i.e. that assumes only *local* or *private* access to each agent's utility function, that is able to provide a constant factor approximation for the non-monotone case. We believe this to be of relevance because it enables the use of non-monotone functions that are characteristic of many practical situations. This is because non-monotonicity captures the natural situation when agents *bite off more than they can chew* and allocating too many tasks to a given agent ends up destroying utility. This is a situation with which previous decentralised algorithms, such as CBBA [19], could not cope with.

The local running time is not very good at $O\left(\frac{n_{\mathcal{T}}^4}{\epsilon^3}\left(\frac{\bar{d}+\underline{d}}{\underline{d}}\right)^2 \log(n_{\mathcal{T}})\right)$ however, we believe that this is a very loose upper bound for two reasons. First the number of samples required to estimate the marginal value has potential to be reduced by using more savvy sampling strategies. Such as for example adaptively sampling only those tasks that have potential to be selected in a given round. That is, rather than starting from scratch at each iteration, information from previous rounds could be used to inform which tasks to sample. The second reason why we believe the number of samples can be reduced is because we have used an overly pessimistic upper bound in the number of iterations of the while loop, $O(n_{\mathcal{T}})$, inside the decreasing threshold procedure. We believe this number will be in the order of $O\left(\min\left(n_{\mathcal{A}}\frac{1}{\epsilon}\log\left(\frac{n_{\mathcal{T}}}{\epsilon}\right), n_{\mathcal{T}}\right)\right)$ due to the speed at which the threshold decreases, but we have been unable to prove it and so it remains a conjecture. Therefore we believe that the most immediate work would be to prove or disprove it. Furthermore, in our work we have set the number of samples in each iteration to be $O\left(\frac{n_{\mathcal{T}}^2}{\epsilon^2}\left(\frac{\bar{d}+\underline{d}}{\underline{d}}\right)^2 \log(n_{\mathcal{T}})\right)$ this is a consequence of Lemma 3.3.2 and it supports the analysis for general non-negative submodular functions. However, as shown in the work of [6] to use a decreasing-threshold approach in the monotone case we only need to sample $O(\frac{n_{\mathcal{T}} \log n_{\mathcal{T}}}{\epsilon})$. Which if combined with the conjecture on the tighter analysis on the number of iterations of the while loop above, would yield a local complexity of $O\left(\frac{n_{\mathcal{A}} n_{\mathcal{T}}}{\epsilon^3}\log(n_{\mathcal{T}})^2\right)$ value oracle calls which is asymptotically better than CBBA's $O(n_{\mathcal{T}}^2)$ when $n_{\mathcal{A}} \ll n_{\mathcal{T}}$. It is also interesting to point out that for monotone functions $\underline{d} = 0$, and so the term $\frac{\bar{d}+\underline{d}}{\underline{d}}$ cancels out.

Furthermore, this analysis has been conducted quantifying the computational cost on the *value oracle* model because for general submodular functions no closed form of the multilinear extension exists. However, for many submodular functions of practical interest such a closed form does exist. In his PhD

thesis Iyer [44] shows how to calculate the multilinear extensions efficiently of the following functions: graph cuts, weighted sums of matroid ranks, set coverage functions, and facility location, among others. In this situation we say we have access to a *mulitlinear oracle* rather than to a *value oracle*, and it would reduce the local computational burden in each iteration from $O\left(\frac{n_{\mathcal{T}}^2}{\epsilon^2}\left(\frac{\bar{d}+d}{\bar{d}}\right)^2 \log(n_{\mathcal{T}})\right)$ value oracle calls to just 1 multilinear oracle call. So if the conjecture above were to be true, and we were to have access to a *multilinear oracle*, then our algorithm could be proven to be much more efficient than CBBA when $n_{\mathcal{A}} \ll n_{\mathcal{T}}$.

To summarise, our algorithm is the first to provide constant factor approximation guarantees for non-monotone submodular utilities. Its running time can, and should, be improved. However, we believe that it is an important milestone, and that it has a practical relevance. Indeed, in light of the numerical experiments that we present in Chapter 5, we believe that the constant factor approximation ratios of our algorithm are not just of theoretical interest. Sure enough, our algorithm is able to address scenarios of practical interest that the current state of the art CBBA [19] could not. We believe this is of great importance, because it validates that a new family of value functions -non-monotone sumbodular- opens to use in decentralised task allocation by researchers and practitioners. CBBA's great success -[19] has 250+ citations as of late 2017- was largely because it enabled decentralised task allocation with constant factor approximation for non-negative monotone submodular functions. This was in spite of the fact that the monotonicity assumption may be too restrictive in many real world situations -such as the simple survival probability model here. Therefore, we think that a fruitful avenue for future work is to build non-negative non-monotone submodular value function models that are of practical interest, let us explain briefly a few possibilities:

- **Coverage with diversity**. When trying to explore an area the notion of information and coverage is naturally submodular, i.e. as more sensors or UAVs are deployed the marginal utility of adding more diminishes. These models have been used with success in practice: [53, 87, 88]. However, these use monotone submodular models for information gathering that have been shown to be afflicted by excessive concentration, or lack of diversity [65]. To remedy this several submodular models have been proposed [23, 65, 94]. These functions are non-monotone submodular because they penalise excessive concentration and reward diversity. These works are in the context of document and image summarisation, and therefore, we believe that there is a prosperous avenue of research in studying and adapting these functions to search, surveillance, or exploration missions, now that the algorithms presented here enable their application in decentralised task allocation.

- **Tasks allocation under risk of agent failure**. We believe that a non-monotone model is only natural when considering a task allocation scenario where the agents can fail at their execution of their tasks. One example is the model presented here, where the agents (UAVs) can be detected and thus fail their mission. Another example is with a multi-robot team executing complicated manipulation functions, say in rough terrain, where completing each task is risky because the robot may suffer some

difficulty, such as getting stuck, and thus fail the mission. In this situation, a single robot may well be the best suited to carry out all the required tasks, but a solution where an individual robot carries all the tasks would be undesirable because of the high risk of failure that it would involve. Monotone submodular functions are structurally ill-suited to model such scenarios, because, by definition, they do not contemplate a reduction in value of an excessively large number of tasks. Therefore, the algorithms presented here give ability to researchers to devise more suitable non-monotone submodular models, opening a fertile line of research.

- **Enemy Network Jamming**. Jamming enemy networks is an important part of modern warfare. These operations usually require a coordinated action a set of distributed assets behind enemy lines, and thus it is obvious use case for a team of small UAVs or robots. The goal is usually to partition a radio communications network [76]. Therefore, a graph cut model, or a combination thereof, is a very natural way to model the problem [33]. Graph cut functions are non-monotone submodular, and so, our algorithm is well suited to solve problems that embody them. Hence, we think that an interesting research line opens in this direction.

- **Aerial Firefighting**. Aerial firefighting is a vital tool to combat large forest fires, and it is among the most dangerous missions that a civilian pilot can undertake [1]. It is another area where, in the future, UAVs could deliver immense benefits, removing humans from danger and operating in conditions where is currently not possible with a human pilot, e.g. at night time, or with limited visibility. One example of such a type of UAV is described in the patent application by J. Moore [74]. These missions are usually carried out in groups of heterogeneous aircraft because there is an advantage in the persistent application of firefighting agents and in its geographically distributed attack. Therefore, it is an ideal application for task allocation in a team of UAVs. When an aircraft attacks more than one fire point the pressure in the tank drops as the agent is delivered, and the effectiveness is reduced. Furthermore, the attack is altogether futile if insufficient firefighting agent is delivered in a fire front. Therefore, a non-monotone function with diminishing returns seems a viable way to model the problem, providing another application area where our algorithm could be effective.

7.3 A Combinatorial Auction Framework For Decentralised Task Allocation

Finally, in chapter 6 we have presented a decentralised algorithm for the Multi Robot Task Allocation Problem with communication costs comparable to those of the state of the art such as CBBA [19]. The advantage of our framework is that it provides a way to leverage existing algorithmic techniques in a decentralised setting. This opens the use of a lot of the algorithmic technology available in a centralised setting -Mixed Integer Programming, Constraint Programming, Metaheuristics, etc- where agents can only access their own objective functions, and not that of their peers. The numerical results that we have pre-

sented are preliminary but demonstrate that its performance improve over the state of the art (CBBA [19]) with representative routing objective functions: MinSum, MinMax and MinAve. We remark that we present here a *framework* rather than an individual *algorithm* because its key advantage lies in the flexibility in the formulation and the solution of the bidding problem with tailored methods. This flexibility comes at a price: we cannot establish any formal approximation guarantees, nor a polynomial running time. However, solution methods that have performed well in practice with other NP-Hard problems could be adapted to provide a practical computation running time. For example, one could use an off-the-shelf solver to solve the bidding problem with an approximation gap or a time limit. Therefore, given that this approach has performed well for many real-world problems and continues to be a fertile field of in the Operations Research community, we believe that it could be viable alternative in to tackle the decentralised solution of a problem that does not satisfy any conditions (such as submodularity) that enables constant factor approximation algorithms. There are two strands of future work that could yield interesting results. The first is to explore the refinement of the solution once all the rounds have been completed. For example, one could re-run the whole bidding process several times after it has been completed, allowing agents to use in the solution of their bidding problems the bids that have been exchanged in previous rounds. Another idea is, once a whole run with stages $1, ..., |\mathcal{T}|$ has been conducted, to run rounds with a random maximum bundle size allowing agents to use all the information exchanged previously. The second line of future work, would be the systematic study of how different kinds of real-world problems (routing, scheduling, etc.) perform in our framework, and how the current best centralised methods available for their solution can be applied to our framework.

Bibliography

[1] *Federal Aerial Firefighting: Assessing Safety and Effectiveness.* USDA Forest Service, and Bureau of Land Management, 2002.

[2] *Perdix Fact Sheet.* The Strategic Capabilities Office (US DoD), 2016.

[3] Technology Quaterly: A Sudden Light. *The Economist*, aug 2016.

[4] A.A. Ageev and M.I. Sviridenko. Pipage Rounding: A New Method of Constructing Algorithms with Proven Performance Guarantee. *Journal of Combinatorial Optimization*, 8(3):307–328, sep 2004.

[5] M. Argyle, D.W. Casbeer, and R. Beard. A Multi-Team Extension of the Consensus-Based Bundle Algorithm. In *Proceedings of the American Control Conference*, pages 5376–5381, 2011.

[6] Ashwinkumar Badanidiyuru and Jan Vondrák. Fast algorithms for maximizing submodular functions. In *Proceedings of the Twenty-Fifth Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 1497–1514. Society for Industrial and Applied Mathematics, 2014.

[7] Levent Bayindir and Erol Sahin. A review of studies in swarm robotics. *Turkish Journal of Electrical Engineering & Computer Sciences*, 15(2):115–147, 2007.

[8] Dimitri P. Bertsekas and David A. Castañon. Parallel synchronous and asynchronous implementations of the auction algorithm. *Parallel Computing*, 17(6-7):707–732, sep 1991.

[9] Luca F Bertuccelli, Han-lim Choi, Peter Cho, and Jonathan P How. Real-time Multi-UAV Task Assignment in Dynamic and Uncertain Environments. In *AIAA Guidance, Navigation, and Control Conference and Exhibit*, 2009.

[10] Jeffrey Bilmes and Wenruo Bai. Deep Submodular Functions. jan 2017.

[11] Jonathan M Borwein, Jon D Vanderwerff, and Others. *Convex functions: constructions, characterizations and counterexamples*, volume 109. Cambridge University Press Cambridge, 2010.

[12] Niv Buchbinder and Moran Feldman. Constrained Submodular Maximization via a Non-symmetric Technique. 2016.

[13] Niv Buchbinder, Moran Feldman, and Roy Schwartz. Comparing Apples and Oranges: Query Tradeoff in Submodular Maximization. oct 2014.

[14] R E Burkard, M Dell'Amico, and S Martello. *Assignment Problems*. SIAM e-books. Society for Industrial and Applied Mathematics (SIAM, 3600 Market Street, Floor 6, Philadelphia, PA 19104), 2009.

[15] Gruia Calinescu, Chandra Chekuri, Martin Pál, and Jan Vondrák. Maximizing a Monotone Submodular Function Subject to a Matroid Constraint. *SIAM Journal on Computing*, 40(6):1740–1766, jan 2011.

[16] Glenn Michael Callow, Markus Deittert, and John Paterson Bookless. Goal Based Planning System. *WPO Patent Office, BAE SYSTEMS PLC*, page WO/2013/030538, 2013.

[17] Chandra Chekuri, Jan Vondrák, and Rico Zenklusen. Submodular Function Maximization via the Multilinear Relaxation and Contention Resolution Schemes. *SIAM Journal on Computing*, 43(6):1831–1879, nov 2014.

[18] Yuxin Chen, Shervin Javdani, Amin Karbasi, J Andrew Bagnell, Siddhartha S Srinivasa, and Andreas Krause. Submodular Surrogates for Value of Information. In *AAAI*, pages 3511–3518, 2015.

[19] Han-lim Choi, Luc Brunet, Jonathan P How, and Senior Member. Consensus-Based Decentralized Auctions for Robust Task Allocation. *IEEE Transactions on Robotics*, 25(4):912–926, 2009.

[20] Han-Lim Choi, A.K. Whitten, and J.P. How. Decentralized task allocation for heterogeneous teams with cooperation constraints. In *American Control Conference*, pages 3057–3062, 2012.

[21] Jorge Cortés. Distributed algorithms for reaching consensus on general functions. *Automatica*, 44(3):726–737, mar 2008.

[22] G. P. Das, T. M. McGinnity, S. A. Coleman, and L. Behera. A fast distributed auction and consensus process using parallel task allocation and execution. In *2011 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 4716–4721. IEEE, sep 2011.

[23] Anirban Dasgupta, Ravi Kumar, and Sujith Ravi. Summarization Through Submodularity and Dispersion. In *ACL (1)*, pages 1014–1022, 2013.

[24] Prithviraj Dasgupta, Angélica Munoz-Meléndez, and K R Guruprasad. Multi-robot terrain coverage and task allocation for autonomous detection of landmines. In *SPIE Defense, Security, and Sensing*, pages 83590H—-83590H. International Society for Optics and Photonics, 2012.

[25] M Bernardine Dias. Traderbots: A new paradigm for robust and efficient multirobot coordination in dynamic environments. *Robotics Institute*, page 153, 2004.

[26] M Bernardine Dias and Anthony Stentz. A free market architecture for distributed control of a multirobot system. In *6th International Conference on Intelligent Autonomous Systems (IAS-6)*, pages 115–122, 2000.

[27] M.B. Dias, R. Zlot, N. Kalra, and a. Stentz. Market-Based Multirobot Coordination: A Survey and Analysis. *Proceedings of the IEEE*, 94(7):1257–1270, jul 2006.

[28] Devdatt P Dubhashi and Alessandro Panconesi. *Concentration of measure for the analysis of randomized algorithms*. Cambridge University Press, 2009.

[29] Shaddin Dughmi, Tim Roughgarden, and Mukund Sundararajan. Revenue submodularity. In *Proceedings of the 10th ACM conference on Electronic commerce*, pages 243–252. ACM, 2009.

[30] Alina Ene and Huy L Nguyen. Constrained submodular maximization: Beyond 1/e. In *Foundations of Computer Science (FOCS), 2016 IEEE 57th Annual Symposium on*, pages 248–257. IEEE, 2016.

[31] Uriel Feige. A threshold of ln n for approximating set cover. *Journal of the ACM (JACM)*, 45(4):634–652, 1998.

[32] M. Feldman, Joseph Naor, and R. Schwartz. A Unified Continuous Greedy Algorithm for Submodular Maximization. In *2011 IEEE 52nd Annual Symposium on Foundations of Computer Science*, pages 570–579. IEEE, oct 2011.

[33] Jixin Feng, Warren E Dixon, and John M Shea. Fast algorithms for jammer placement to partition a wireless network. In *Communications (ICC), 2017 IEEE International Conference on*, pages 1–6. IEEE, 2017.

[34] Wan Fokkink. *Distributed algorithms: an intuitive approach*. MIT Press, 2013.

[35] B. P. Gerkey and Maja J Mataric. A Formal Analysis and Taxonomy of Task Allocation in Multi-Robot Systems. *The International Journal of Robotics Research*, 23(9):939–954, sep 2004.

[36] Brian P Gerkey and Maja J Mataric. Sold!: Auction methods for multirobot coordination. *IEEE transactions on robotics and automation*, 18(5):758–768, 2002.

[37] Shayan Oveis Gharan and Jan Vondrák. Submodular maximization by simulated annealing. In *Proceedings of the twenty-second annual ACM-SIAM symposium on Discrete Algorithms*, pages 1098–1116. Society for Industrial and Applied Mathematics, 2011.

[38] Silvia Giannini, Antonio Petitti, Donato Di Paola, and Alessandro Rizzo. Asynchronous Max-Consensus Protocol With Time Delays: Convergence Results and Applications. *IEEE Transactions on Circuits and Systems I: Regular Papers*, 63(2):256–264, feb 2016.

[39] Inc. Gurobi Optimization. Gurobi Optimizer Reference Manual, 2014.

[40] Wassily Hoeffding. Probability inequalities for sums of bounded random variables. *Journal of the American statistical association*, 58(301):13–30, 1963.

[41] Simon Hunt, Qinggang Meng, and CJ Hinde. An extension of the consensus-based bundle algorithm for group dependant tasks with equipment dependencies. *Neural Information Processing*, pages 518–527, 2012.

[42] Sarah Ismail and Liang Sun. Decentralized hungarian-based approach for fast and scalable task allocation. In *Unmanned Aircraft Systems (ICUAS), 2017 International Conference on*, pages 23–28. IEEE, 2017.

[43] F. Iutzeler, P. Ciblat, and J. Jakubowicz. Analysis of Max-Consensus Algorithms in Wireless Channels. *IEEE Transactions on Signal Processing*, 60(11):6103–6107, nov 2012.

[44] RK Iyer. *Submodular Optimization and Machine Learning: Theoretical Results, Unifying and Scalable Algorithms, and Applications*. PhD Thesis, University of Washington, 2015.

[45] Stefanie Jegelka and Jeff Bilmes. Submodularity beyond submodular energies: coupling edges in graph cuts. In *Computer Vision and Pattern Recognition (CVPR), 2011 IEEE Conference on*, pages 1897–1904. IEEE, 2011.

[46] LB Johnson, SS Ponda, HL Choi, and JP How. Asynchronous Decentralized Task Allocation for Dynamic Environments. In *AIAA Infotech at Aerospace Conference*, number March, 2011.

[47] Luke Johnson, Han-lim Choi, Sameera Ponda, and Jonathan P How. Allowing Non-Submodular Score Functions in Distributed Task Allocation. In *Proceedings of the IEEE Conference on Decision and Control*, number 1, pages 4702–4708, 2012.

[48] Luke B Johnson, Han-Lim Choi, Sameera S Ponda, and Jonathan P How. Decentralized task allocation using local information consistency assumptions. *Journal of Aerospace Information Systems*, 2017.

[49] Nidhi Kalra, Dave Ferguson, and Anthony Stentz. Hoplites: A market-based framework for planned tight coordination in multirobot teams. In *Robotics and Automation, 2005. ICRA 2005. Proceedings of the 2005 IEEE International Conference on*, pages 1170–1177. IEEE, 2005.

[50] Frank Kelly and Richard Steinberg. A combinatorial auction with multiple winners for universal service. *Management Science*, 2000.

[51] David Kempe, Jon Kleinberg, and Éva Tardos. Maximizing the spread of influence through a social network. In *Proceedings of the ninth ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 137–146. ACM, 2003.

[52] G. a. Korsah, a. Stentz, and M. B. Dias. A comprehensive taxonomy for multi-robot task allocation. *The International Journal of Robotics Research*, 32(12):1495–1512, oct 2013.

[53] a. Krause, C. Guestrin, a. Gupta, and J. Kleinberg. Near-optimal sensor placements: maximizing information while minimizing communication cost. *2006 5th International Conference on Information Processing in Sensor Networks*, 2006.

[54] Andreas Krause and Daniel Golovin. Submodular function maximization. *Tractability: Practical Approaches to Hard Problems*, 3(19):8, 2012.

[55] Andreas Krause, Jure Leskovec, Carlos Guestrin, Jeanne VanBriesen, and Christos Faloutsos. Efficient sensor placement optimization for securing large water distribution networks. *Journal of Water Resources Planning and Management*, 134(6):516–526, 2008.

[56] Harold W Kuhn. The Hungarian method for the assignment problem. *Naval Research Logistics (NRL)*, 2(1-2):83–97, 1955.

[57] Indraneel S Kulkarni and Dario Pompili. Task allocation for networked autonomous underwater vehicles in critical missions. *IEEE Journal on Selected Areas in Communications*, 28(5), 2010.

[58] Michail G Lagoudakis, Evangelos Markakis, David Kempe, Pinar Keskinocak, Anton J Kleywegt, Sven Koenig, Craig A Tovey, Adam Meyerson, and Sonal Jain. Auction-Based Multi-Robot Routing. In *Robotics: Science and Systems*, volume 5, page 343C350. Rome, Italy, 2005.

[59] Ailsa Land, Susan Powell, and Richard Steinberg. Chapter 6 PAUSE : A Computationally Tractable Combinatorial Auction. In Peter Cramton, Yoav Shoham, and Richard Steinberg, editors, *Combinatorial Auctions*, chapter 6, pages 139–157. 2006.

[60] Benny Lehmann, Daniel Lehmann, and Noam Nisan. Combinatorial auctions with decreasing marginal utilities. In *Proceedings of the 3rd ACM conference on Electronic Commerce*, pages 18–28. ACM, 2001.

[61] Thomas Lemaire, Rachid Alami, and Simon Lacroix. A distributed tasks allocation scheme in multi-UAV context. In *Robotics and Automation, 2004. Proceedings. ICRA'04. 2004 IEEE International Conference on*, volume 4, pages 3622–3627. IEEE, 2004.

[62] Jan Karel Lenstra, David B Shmoys, and Éva Tardos. Approximation algorithms for scheduling unrelated parallel machines. *Mathematical programming*, 46(1):259–271, 1990.

[63] Jure Leskovec, Andreas Krause, Carlos Guestrin, Christos Faloutsos, Jeanne VanBriesen, and Natalie Glance. Cost-effective outbreak detection in networks. In *Proceedings of the 13th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 420–429. ACM, 2007.

[64] Hui Lin and Jeff Bilmes. Multi-document summarization via budgeted maximization of submodular functions. In *Human Language Technologies: The 2010 Annual Conference of the North American Chapter of the Association for Computational Linguistics*, pages 912–920. Association for Computational Linguistics, 2010.

[65] Hui Lin and Jeff Bilmes. A class of submodular functions for document summarization. In *Proceedings of the 49th Annual Meeting of the Association for Computational Linguistics: Human Language Technologies-Volume 1*, pages 510–520. Association for Computational Linguistics, 2011.

[66] Lantao Liu and Dylan A. Shell. An anytime assignment algorithm: From local task swapping to global optimality. *Autonomous Robots*, 35(4):271–286, jul 2013.

[67] Chunbo Luo, Paul Ward, Stephen Cameron, Gerard Parr, and Sally Mcclean. Communication Provision for a Team of Remotely Searching UAVs : A Mobile Relay Approach. In *IEEE Globecom Workshops*, pages 1544–1549, 2012.

[68] Nancy A Lynch. *Distributed algorithms*. Morgan Kaufmann, 1996.

[69] S Thomas McCormick. Submodular function minimization. *Handbooks in operations research and management science*, 12:321–391, 2005.

[70] Travis Mercker, DW Casbeer, Travis Millet, and Maruthi Akella. An extension of consensus-based auction algorithms for decentralized, time-constrained task assignment. *American Control ...*, pages 6324–6329, 2010.

[71] Michel Minoux. Accelerated greedy algorithms for maximizing submodular set functions. In *Optimization Techniques*, pages 234–243. Springer-Verlag, Berlin/Heidelberg, 1978.

[72] Baharan Mirzasoleiman, Ashwinkumar Badanidiyuru, and Amin Karbasi. Fast constrained submodular maximization: Personalized data summarization. In *ICLM'16: Proceedings of the 33rd International Conference on Machine Learning (ICML)*, 2016.

[73] Sangwoo Moon, Eunmi Oh, and David Hyunchul Shim. An Integral Framework of Task Assignment and Path Planning for Multiple Unmanned Aerial Vehicles in Dynamic Environments. *Journal of Intelligent & Robotic Systems*, 70(1-4):303–313, sep 2012.

[74] Jason Moore. *UAV Fire-fighting System*. US Patent Application, US 20130134254 A1, 2013.

[75] Alejandro R Mosteo, Luis Montano, and Michail G Lagoudakis. Guaranteed-performance multi-robot routing under limited communication range. *Distributed Autonomous Robotic Systems*, 8:491–502, 2009.

[76] Aristides Mpitziopoulos, Damianos Gavalas, Charalampos Konstantopoulos, and Grammati Pantziou. A survey on jamming attacks and countermeasures in WSNs. *IEEE Communications Surveys & Tutorials*, 11(4), 2009.

[77] G. L. Nemhauser, L. A. Wolsey, and M. L. Fisher. An analysis of approximations for maximizing submodular set functionsI. *Mathematical Programming*, 14(1):265–294, dec 1978.

[78] Trung Thanh Nguyen, Magnus Roos, and Jörg Rothe. A survey of approximability and inapproximability results for social welfare optimization in multiagent resource allocation. *Annals of Mathematics and Artificial Intelligence*, 68(1-3):65–90, jan 2013.

[79] Lynne E Parker. ALLIANCE: An architecture for fault tolerant multirobot cooperation. *IEEE transactions on robotics and automation*, 14(2):220–240, 1998.

[80] Vangelis T. Paschos. A survey of approximately optimal solutions to some covering and packing problems. *ACM Computing Surveys*, 29(2):171–209, jun 1997.

[81] Sameera Ponda, Josh Redding, HL Choi, Jonathan P How, M Vavrina, and J Vian. Decentralized planning for complex missions with dynamic communication constraints. In *American Control Conference*, 2010.

[82] Sameera S Ponda, Luke B Johnson, Alborz Geramifard, and Jonathan P How. Cooperative mission planning for multi-uav teams. In *Handbook of Unmanned Aerial Vehicles*, pages 1447–1490. Springer, 2015.

[83] SS Ponda, LB Johnson, and JP How. Distributed chance-constrained task allocation for autonomous multi-agent teams. *... Control Conference (ACC), ...*, pages 4528–4533, 2012.

[84] R Tyrrell Rockafellar. Lagrange multipliers and optimality. *SIAM review*, 35(2):183–238, 1993.

[85] A Schrijver. *Combinatorial Optimization: Polyhedra and Efficiency*. Algorithms and Combinatorics. Springer Berlin Heidelberg, 2003.

[86] David B Shmoys and Éva Tardos. An approximation algorithm for the generalized assignment problem. *Mathematical programming*, 62(1-3):461–474, 1993.

[87] A. Singh, A. Krause, C. Guestrin, and W. Kaiser. Efficient informative sensing using multiple robots. *Journal of Artificial Intelligence Research*, 2009.

[88] Amarjeet Singh, Andreas Krause, Carlos Guestrin, William J Kaiser, and Maxim A Batalin. Efficient Planning of Informative Paths for Multiple Robots. In *IJCAI*, volume 7, pages 2204–2211, 2007.

[89] Hyun Oh Song, Yong Jae Lee, Stefanie Jegelka, and Trevor Darrell. Weakly-supervised discovery of visual pattern configurations. In *Advances in Neural Information Processing Systems*, pages 1637–1645, 2014.

[90] Richard S Sutton and Andrew G Barto. *Reinforcement learning: An introduction*, volume 1. MIT press Cambridge, 1998.

[91] Zoya Svitkina and Lisa Fleischer. Submodular approximation: Sampling-based algorithms and lower bounds. *SIAM Journal on Computing*, 40(6):1715–1737, 2011.

[92] Gerard Tel. *Introduction to distributed algorithms*. Cambridge university press, 2000.

[93] P Toth and D Vigo. *Vehicle Routing: Problems, Methods, and Applications, Second Edition*. MOS-SIAM Series on Optimization. SIAM, 2014.

[94] Sebastian Tschiatschek, Rishabh K Iyer, Haochen Wei, and Jeff A Bilmes. Learning mixtures of submodular functions for image collection summarization. In *Advances in neural information processing systems*, pages 1413–1421, 2014.

[95] Johannes Van Der Horst, Jason Noble, and Adrian Tatnall. Robustness of market-based task allocation in a distributed satellite system. In *European Conference on Artificial Life*, pages 334–341. Springer, 2009.

[96] Jan Vondrak. Optimal approximation for the submodular welfare problem in the value oracle model. In *Proceedings of the fourtieth annual ACM symposium on Theory of computing - STOC 08*, page 67, New York, New York, USA, may 2008. ACM Press.

[97] Jan Vondrak. A note on concentration of submodular functions. may 2010.

[98] Jan Vondrák. Symmetry and approximability of submodular maximization problems. *SIAM Journal on Computing*, 42(1):265–304, 2013.

[99] Jan Vondrák, Chandra Chekuri, and Rico Zenklusen. Submodular function maximization via the multilinear relaxation and contention resolution schemes. In *Proceedings of the forty-third annual ACM symposium on Theory of computing*, pages 783–792. ACM, 2011.

[100] S De Vries and RV Vohra. Combinatorial auctions: A survey. *INFORMS Journal on computing*, 2003.

[101] Zengfu Wang, Bill Moran, Xuezhi Wang, and Quan Pan. An accelerated continuous greedy algorithm for maximizing strong submodular functions. *Journal of Combinatorial Optimization*, 30(4):1107–1124, 2015.

[102] Andrew Whitten. Decentralized Planning for Autonomous Agents Cooperating in Complex Missions, 2010.

[103] Wikipedia. *Article on Planet Labs, Inc.* Retrieved in mid 2017 from en.wikipedia.org/wiki/Planet_Labs.

[104] Zhi Yan, Nicolas Jouandeau, and Arab Ali. A Survey and Analysis of Multi-Robot Coordination. *International Journal of Advanced Robotic Systems*, 10:1, 2013.

[105] Michael M. Zavlanos, Leonid Spesivtsev, and George J. Pappas. A distributed auction algorithm for the assignment problem. *2008 47th IEEE Conference on Decision and Control*, pages 1212–1217, 2008.

[106] Kai Zhang, Emmanuel G. Collins, and Adrian Barbu. An Efficient Stochastic Clustering Auction for Heterogeneous Robotic Collaborative Teams. *Journal of Intelligent & Robotic Systems*, jan 2013.

[107] Kai Zhang, Emmanuel G. Collins, and Dongqing Shi. Centralized and distributed task allocation in multi-robot teams via a stochastic clustering auction. *ACM Transactions on Autonomous and Adaptive Systems*, 7(2):1–22, jul 2012.