Intent-Based Coordination of Robotic Autonomous Systems for Persistent Reconnaissance

M.A. Korthals Altes

Master of Science Thesis

TU Delft Delft University of Technology

Delft Center for Systems and Control

Intent-Based Coordination of Robotic Autonomous Systems for Persistent Reconnaissance

Master of Science Thesis

For the degree of Master of Science in Systems and Control at Delft University of Technology

M.A. Korthals Altes

June 4, 2021

Supervisor: Prof.dr.ir. T. Keviczky

Faculty of Mechanical, Maritime and Materials Engineering (3mE) · Delft University of Technology

Koninklijke Landmacht

The work in this thesis was supported by the Royal Dutch Army. Their cooperation is hereby gratefully acknowledged.

TU Delft Delft University of Technology

DCSC

Abstract

The introduction of Robotic & Autonomous Systems (RAS) in modern combat seems inevitable, with clear advantages like reduced risk and extensification of personnel. To scope this research, persistent reconnaissance with heterogeneous Unmanned Aerial Vehicles (UAVs) is selected, being one of the more prominent applications. Despite continuous efforts developing advanced hardware and algorithms, real-world implementations are still lacking. The root cause seems to be that state-of-the-art algorithms deal insufficiently with the high dynamics and uncertainty in a military environment.

Currently, the military uses intent-based Command & Control (C2) to deal with precisely these challenges, as they are inherently tied to combat. Therefore, a conversion of the communicative principles of C2 towards a mathematical approach applicable to RAS seems promising, of which intent-based coordination is the result. To be able to deal with the high dynamics and uncertainty, three requirements are formulated. First, *flexibility* is needed to revise the solution locally. Secondly, *robustness* against unreliable communications is necessary, and thirdly, *scalability* is required to ensure the performance can also be maintained for larger Areas of Interest (AOIs) and larger teams of UAVs.

The Single-Agent Reconnaissance Problem (SARP) and Multi-Agent Reconnaissance Problem (MARP) are formulated as a compact combination between the visitation frequency and coverage level approach for persistent reconnaissance. Based on advancements made on teamwork and organizations for Multi-Robot Systems (MRSs), a coordinative method is formulated. This coordinative method partitions an AOI for the MARP into smaller disjoint subsets, such that separate SARPs can be solved independently by each UAV. The key contribution of this research is that this coordinative method functions based on intent, enabling the required flexibility, robustness, and scalability. It does so by constructing a hierarchy of supervisors that perform distributed cooperation on overlapping subsets. This distributed problem is solved using the novel Complex Concurrent Bounding (CCB), which is an adjusted version of Concurrent Forward-Bounding (ConcFB) for Distributed Constraint Optimization Problems (DCOPs) with complex local problems. Additionally, a lower bound is generated to benchmark the obtained solutions, based on the pricing step of branch & price, by applying column generation to a reformulated version of the MARP.

Intent-based coordination shows flexibility against perturbations of the AOI. Especially when changes are spread out, it is not necessary to revise the solution as a whole immediately. Furthermore, if the cooperation is preemptively terminated due to failing communication, robustness is observed against the resulting suboptimal subsets. Especially for higher levels in the hierarchy, the suboptimal solutions can partially be corrected by lower levels. Lastly, the method shows a sublinear growth in computation time for increasingly larger problem instances. As such, intent-based coordination provides an exciting approach to maintain the performance of RAS even in more challenging environments.

Table of Contents

List of Figures

List of Tables

Preface

I have always been fascinated by how large groups of people can organize themselves to work towards a common goal. Especially the massive military campaigns undertaken during World War 2 are a true logistical and communicative feat. During my years at the Royal Dutch Army, I have seen quite the opposite of this ability in Robotic & Autonomous Systems (RAS), which are often very singular or impractical. This left me wondering: is it not possible that we as humans have developed a way to organize and cooperate that is not yet available for robotic systems? Of course, such a general question is hard to answer, and if an answer exists, it comes in many forms. Nonetheless, I have made my attempt for one application with the hope of making true robotic teamwork one step closer.

I would like to thank my supervisor Prof. dr. ir. T. Keviczky for his assistance during the writing of this thesis, and especially for his focus on making my research concrete and mathematically sound. Furthermore, I want to thank Defensity College for enabling me to work on this research.

Delft, University of Technology — M.A. Korthals Altes
June 4, 2021

"Never tell people *how* to do things. Tell them *what* to do, and they will surprise you with their ingenuity."

— *General George S. Patton, U.S. Army*

Chapter 1

Introduction

1-1 The complexity of military autonomy

The Royal Dutch Army pursues the continuous development of Robotic & Autonomous Systems (RAS) [1]. The advantages of such systems include reduced risk, extensification of personnel, enhanced Situational Awareness (SA), and integration with a new Concept of Operations (CONOPS).

Despite the significant investments by NATO partners in RAS and the advancements made on autonomous systems in various civil industries, implementations in a combat environment are still lacking. Though literature provides substantial propositions for advanced algorithms for complex and intelligent behavior like swarming, experiments within a military environment remain dedicated to validating the ruggedness, mechanical reliability, and other conventional benchmarks for new systems. The root cause of this slow implementation seems to be the gap between the scientific assumptions and the reality of combat. Subsequently, the proposed algorithms cannot be effective, prohibiting their implementation to any further stage than simulations or controlled experiments.

Specifically, a combat environment is highly dynamic and uncertain, which means that information is either unavailable, unreliable, or incomplete and needs to be processed quickly. These findings originate from the crushing defeat of the Prussians against Napoleon at Jena and Auerstedt (1806). Afterward, Gerhard von Scharnhorst (1755-1813) concluded that the Prussians had falsely assumed that victory could be gained by mathematical principles. Instead, he deemed war inherently chaotic and unpredictable. As such, the iron Prussian discipline left great opportunities unexploited, as there was no room for local initiative [2]. This laid the groundwork for a more flexible approach to decision-making in a combat environment, which gradually evolved into intent-based Command & Control (C2).

The essence is that through all levels in the command hierarchy, it is completely clear what each respective commander's intent encompasses. The result is that also local initiatives contribute to the global mission objective, such that sudden advantages can be exploited quickly without the need to justify actions up to the highest echelons. The question is if and how this type of C2 can be converted to a mathematical approach for RAS.

1-2 Problem statement

There are many RAS capabilities, but one of the main applications is a network of (heterogeneous) Unmanned Aerial Vehicles (UAVs) performing persistent Intelligence, Surveillance & Reconnaissance (ISR). Persistent reconnaissance means that an Area of Interest (AOI) is continuously monitored for the presence of targets. Though much research is done on solutions for various ISR missions, they are not explicitly designed for a demanding combat environment. This environment can be defined by two main characteristics:

1. High dynamics, meaning that the situation can change suddenly and dramatically, such that new solutions are required quickly.

2. High uncertainty, as communication links are time-varying, message delivery is asynchronous and fallible, bandwidth is scarce, and Emission Control (EMCON) is generally applied to reduce the possibility of jamming and triangulation.

Based on the challenges these present, the following research question is formulated:

Can a solution method be developed for persistent reconnaissance that produces consistently good results in a highly dynamic environment with unreliable communication?

1-3 Contributions

Fundamentally, it is assumed that persistent reconnaissance can be performed by solving the Multi-Agent Reconnaissance Problem (MARP) to optimality, which is not tractable given the challenges of a military environment. To obtain a good solution nonetheless, this thesis attempts to convert the principles of intent-based C2 to a mathematical approach, named intent-based coordination. This coordinative method aims to split the MARP into smaller Single-Agent Reconnaissance Problems (SARPs), which are solved separately. In Figure 1.1, this difference between UAVs optimizing their paths jointly and individually in disjoint subsets of the AOI is shown.

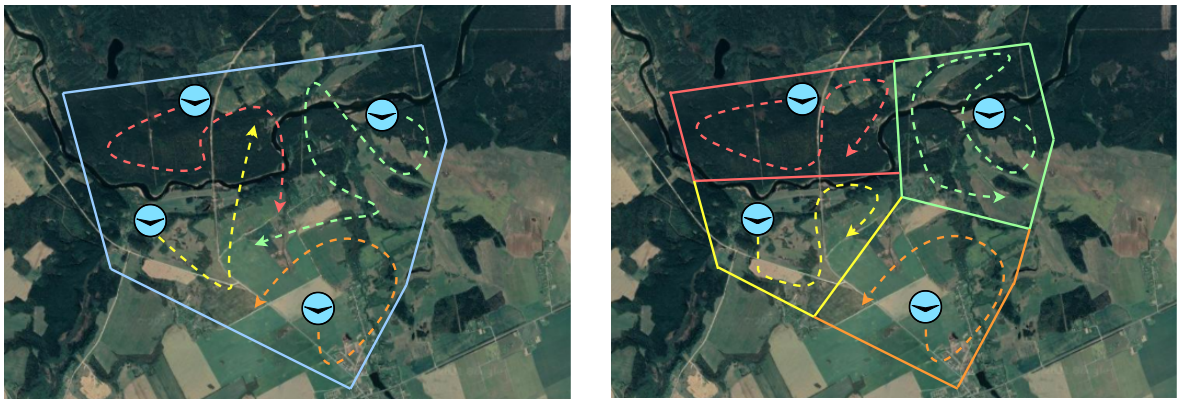


Figure 1.1: An example of splitting the Area of Interest (AOI) of a Multi-Agent Reconnaissance Problem (MARP) into smaller, disjoint Single-Agent Reconnaissance Problems (SARPs)

The overarching structure is displayed in Figure 1.2. Given the AOI, intent-based coordination creates subsets for multiple SARPs. The combined result should resemble the optimal solution of the MARP, which can be assessed using specific benchmarking methods. Therefore, the main contributions of this thesis can be listed as such:

- Highlighting the fundamental issues of using conventional solution approaches for various types of reconnaissance missions in a realistic combat environment (Chapter 2).
- Formulating the SARP and MARP as compact models, combining both the frequency and coverage level approach for persistent reconnaissance (Chapter 3).
- To generate tight lower bounds, a relaxed reformulation of the MARP is solved using column generation, which includes solving the Elementary Shortest Path Problem (ESPP) frequently. Due to the MARP structure, a cycle distance must be included, next to some other problem-specific adjustments to improve forward labeling [3] (Chapter 4).
- An interpretation of the conversion of intent-based C2 by describing a distributed, hierarchical framework for intent-based coordination (Section 5-2).
- The implementation of Fuzzy C-Means (FCM) [4] with an added post-processing interpolation method for weighed clustering of relevant sector features to reduce the problem complexity and accommodate sensor heterogeneity (Section 5-3).
- Formulating a task allocation problem that subdivides the clusters among agents, acting as a top-down heuristic to create subsets. This allocation includes a measure of task utility and a novel quadratic task dependency constraint to accommodate the limited capacities (Section 5-4). The formulation is extended to a cooperative formulation suitable for the distributed, hierarchical framework (Section 5-5-2).
- To solve the distributed cooperative formulation, the Concurrent Forward-Bounding (ConcFB) [5] algorithm is adjusted to accommodate complex local problems, leading to Complex Concurrent Bounding (CCB) (Section 5-5-5).
- A thorough analysis that includes the parameters and component performance and specific quantitative assessments for a military environment. (Chapter 6).

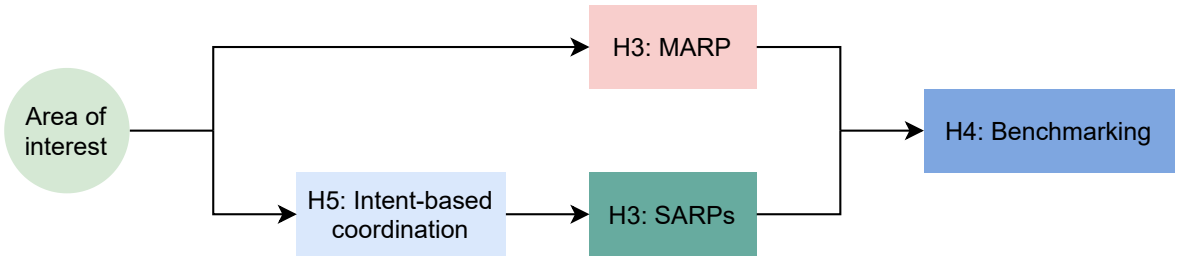


Figure 1.2: A general depiction of the solution approach. Instead of solving the MARP to optimality, the AOI is partitioned into smaller subsets such that smaller SARPs can be solved separately. A benchmarking method is used to compare the results.

1-4 Structure

To clarify the structure of this thesis, a brief introduction of each chapter is provided, with an emphasis on the contributions.

Chapter 2: Persistent Reconnaissance State of the Art A classification is made of the different kinds of ISR missions, clarifying their specific utility and central questions. The solution methods of these problems are discussed in general, with an emphasis on cooperation. Furthermore, the relation between coordination and teamwork for Multi-Robot Systems (MRSs) is explained, providing an argument for its use.

Chapter 3: Modelling Reconnaissance A rigorous, compact mathematical formulation is provided for the underlying persistent reconnaissance problem in the form of the SARP and MARP. This novel formulation builds on the various approaches highlighted in Chapter 2, and is kept compact and without a time horizon to allow more consistent analysis.

Chapter 4: Benchmarking This chapter includes both methods for generating upper and lower bounds. The lower bound is the most thorough, as it builds on the more advanced process of column generation during the pricing step of branch & price.

Chapter 5: Intent-Based Coordination The methodology of intent-based coordination is explained, highlighting all components, including clustering, task allocation, and distributed cooperation. Each of these components has novel contributions, of which the proposal of CCB is the most prominent.

Chapter 6: Performance Analysis As this thesis consists of the complex interplay of many components, the analysis is split into three parts. First, the effects of the main parameters are quantified. Secondly, the performance of all separate components is analysed. Thirdly, the results are presented for a military environment. Each part of the analysis includes a summary of the main findings.

Chapter 7-8: Conclusion & Discussion Lastly, the overarching findings and a discussion with extensive suggestions for future work are presented in Chapter 7 and Chapter 8, respectively.

Chapter 2

Persistent Reconnaissance State of the Art

This chapter provides a classification of Intelligence, Surveillance & Reconnaissance (ISR). Furthermore, an overview is given of the different solution approaches. Lastly, the shortcomings of these methods in a military environment will be explained. A broader analysis is provided in the preceding literature review [6], on which this chapter is based.

2-1 Classification of reconnaissance

Persistent reconnaissance can be considered a type of ISR. The definition used here for ISR is that an agent is physically moving in at least two dimensions to obtain information. The type of required information and the method to obtain it define the precise mission it is performing. The main missions identified are searching, tracking, targeting, coverage, and persistent reconnaissance. This classification builds upon the work of Nigam [7], which focuses mainly on the concept of persistency. It should be noted that this classification is not exhaustive and that used definitions can vary throughout literature. For example, patrolling is excluded as passing a perimeter is a one-dimensional problem. A complete overview of relevant literature is displayed in Figure 2.1, split per mission type and subdivided by main research direction. As the formulation for persistent reconnaissance formulated in Chapter 3 builds upon various elements from these mission types, each type is explained briefly using a carefully constructed central research question and a concise mathematical formulation, summarizing the work from an extensive literature review.

Searching

Given an area with specific targets and limited information. How can the position of these targets be estimated to find them in a minimum amount of time?

This problem can be formulated as a discrete-time Partially Observable Markov Decision Process (POMDP), using the tuple $\langle S, A, T, R, \omega, O, \gamma \rangle$. The environment is in some state

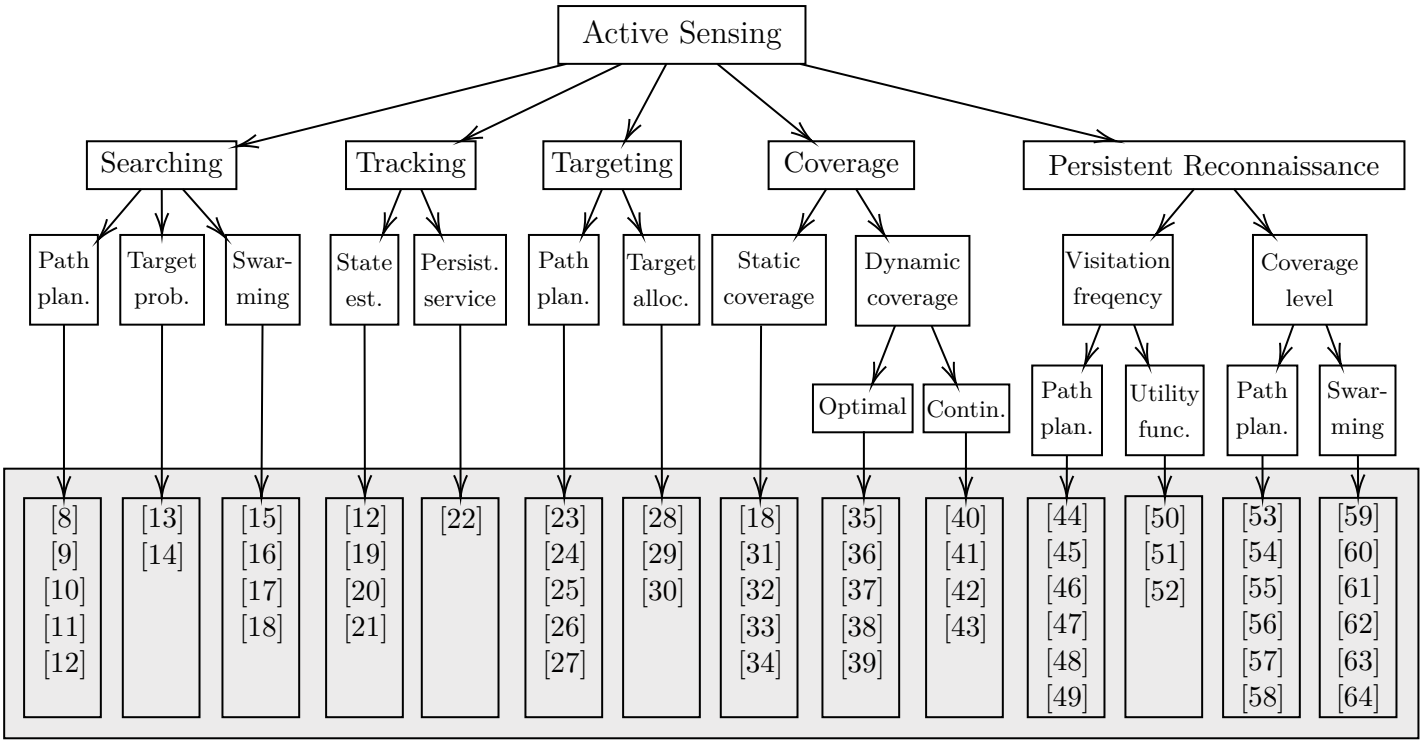


Figure 2.1: An overview of the different ISR missions applied in a military context, along with the most important sources.

$s \in S$, which includes the position of each agent and any relevant information used to predict the position of targets. To find the target, the agent can take a repositioning action $a \in A$. The environment then transitions to state s' using the conditional transition probabilities $T(s'|s, a, o)$, depending on the performed action a and the observation $o \in \Omega$, which follows from the conditional observation probabilities $O(o|s, a)$. The reward for observing a target is described as $r \in R(o)$. The optimization goal is to maximize the expected future discounted reward $E\left[\sum_{t=0}^{\infty} \gamma^t r_t\right]$.

Generally, this POMDP can be solved with dynamic programming, which is computationally intensive. Furthermore, the state and observation transition probabilities T and O need to be defined, which is no trivial task. For instance, there is a difference between static, dynamic, evasive, or cooperative targets [8]. As displayed in Figure 2.1, the main research directions are planning an optimal path given the predicted locations of targets, improving the predictions itself, and implementing swarming principles.

Tracking

Given noisy and incomplete states of moving targets, how can the agents' trajectories be planned to track the targets while collision avoidance is maintained?

As with searching, a discrete-time POMDP describes the problem well by using the tuple $\langle S, A, T, R, \omega, O, \gamma \rangle$. The state $s \in S$ now includes both the positions of agents and targets. Therefore, the reward $r \in R(s, a, o)$ is based on the probability of observing a target in a

specific position, given its current (or past) states. The reward $r \in R(s, a, o)$ remains the same, such that the expected future discounted reward $E\left[\sum_{t=0}^{\infty} \gamma^t r_t\right]$ needs to be maximized. In addition, states can be penalized if collisions occur.

Especially the problem of multi-target tracking and obstacle avoidance in challenging environments like urban terrain are topics of continuous research. As Figure 2.1 shows, much research is also dedicated to accurate state estimation, for which reviews by Selvaraj et al. [65] and Liu et al. [66] provide more background. Furthermore, some papers combine searching and tracking, which follows naturally from the combined mission of finding a target first and tracking it afterward [12, 17]. Lastly, research is also dedicated to maintaining the ability to track targets through continuous service, even if agents need to return due to fuel shortages.

Targeting

Given the positions of (heterogeneous) targets, which targets need to be visited by which agent, and in what order?

This problem closely resembles the uncapacitated Heterogeneous Fleet Vehicle Routing Problem (HFVRP). Specifically, given $i, j \in N$ targets and $k \in P$ agents, a binary decision variable x_{ijk} indicates if agent k services target j after target i. The cost associated with this is given by c_{ijk}. The problem can then be formulated as:

$$\min_{x_{ijk}} \quad \sum_{i \in N} \sum_{j \in N} \sum_{k \in P} x_{ijk} c_{ijk} \tag{2.1}$$

$$\text{s.t.} \quad \sum_{k \in P} \sum_{i \in N} x_{ijk} = \sum_{k \in P} \sum_{i \in N} x_{jik} = 1 \qquad \forall j \in N \tag{2.2}$$

$$\sum_{i \in N} x_{ijk} = \sum_{i \in N} x_{jik} \qquad \forall j \in N,\ k \in P \tag{2.3}$$

$$\textit{Subtour elimination constraints} \qquad \forall k \in P \tag{2.4}$$

Equation (2.1) prescribes that the cost associated with servicing multiple (heterogeneous) targets with a set of (heterogeneous) agents is minimized. Constraints (2.2) ensures that only one agent services each target. Constraint (2.3) preserves the path continuity, and Constraints (2.4) are left out for conciseness, but any existing set of subtour elimination constraints for the Vehicle Routing Problem (VRP) can be applied.

As the HFVRP is NP-hard, the research is largely dedicated to formulating efficient algorithms. Therefore, the research is mainly split between finding effective routes passing all targets or allocating the targets to the best-suited agent.

Static coverage

Given a set of agents, how can they be located to maximize the coverage of an area of interest against minimal placement costs?

Though this question does not directly satisfy the definition of ISR, some applications repeatedly solve the static coverage problem to deal with a dynamic problem. The problem can

be solved by generating a Voronoi partition based on a probability density function defining the area of interest [31]. Given a grid $A \in R^2$ with sectors $s \in A$, the density function is defined as $\phi(s)$. Degrading sensor performance of agent $k \in P$ is based on the Euclidean distance from the agent to point $s \in A$, described with $f(||s - p_k||)$. The resulting coverage effectiveness is therefore defined by the closest agent, such that the points are partitioned in a Voronoi diagram $V_k = \{s \in A \mid ||s - p_k|| \leq ||s - p_{k'}||,\ \forall k' \neq k \in P\}$. The goal is to find the optimal placement of each agent p_k, such that the following equation is minimized:

$$\min_{p_k} \quad \int_G \min_{k \in P} f(||s - p_k||)\phi(s)ds \tag{2.5}$$

The problem can be extended by minimizing the cost of moving to a specific position or the cost of including an agent. Some rudimentary dynamics are implemented by including these costs and repeatedly solving the static problem [40]. However, the core principle remains that agents can only cover while static, such that only relocation is possible, and no coverage is provided while on the move.

Dynamic coverage

Given a set of agents, how can their paths be planned to cover the complete area of interest at minimal cost?

A simple version of this problem can be described using the Multiple Traveling Salesmen Problem (MTSP). Given a set of sectors $i, j \in N$ that needs to be visited by $k \in P$ agents, a binary decision variable x_{ij} is set to one if sector j is visited after sector i, with the source sector named s. The associated costs are given described by c_{ij}, which usually represents time. The formulation is then given as:

$$\min_{x_{ij}} \quad \sum_{i \in N} \sum_{j \in N} x_{ij} c_{ij} \tag{2.6}$$

$$\text{s.t.} \quad \sum_{i \in N} x_{ij} = \sum_{i \in N} x_{ji} = 1 \qquad \forall j \in N \tag{2.7}$$

$$\sum_{j \in N} x_{sj} = \sum_{i \in N} x_{is} \leq |P| \tag{2.8}$$

$$\textit{Subtour elimination constraints} \tag{2.9}$$

The objective function (2.6) minimizes the total cost for all paths. Constraints (2.7) ensures that each sector has one incoming and one departing arc. Constraint (2.8) ensures that no more agents depart and return at the source sector than available. The subtour elimination constraints (2.9) are left out for conciseness.

The important difference with static coverage is that the agents can cover the area on the move and that the coverage should be achieved over time, in contrast to coverage at a specific moment. Difficulties mainly arise when covering complex polygons, when agents have different starting points, or when physical constraints like a turning radius are present. Generally, the focus is not on heterogeneity but on an optimal path, which is a key difference with targeting.

Relevant surveys include those by Choset et al. [67], and Cabreira et al. [68]. As displayed in Figure 2.1, research is mainly dedicated to optimally solving a variant of the formulated MTSP or implementing computationally efficient continuous control.

Persistent reconnaissance by visitation frequency

Given a set of agents, how can their paths be planned to minimize the maximum uncertainty in the area of interest over indefinite time?

An MTSP with adjusted subtour elimination constraints can describe this problem [44], such that the total cycle time is included during optimization. A set of $i, j \in N$ sectors and a set of $k \in P$ agents is given, with a binary decision variable x_{ij} if sector j is visited after $i \in N$. Furthermore, the arrival time at sector $i \in N$ is set to u_i, and the time left to the arrival at the source sector $s \in N$ is set to v_i. These values are used to construct the maximum cycle time z. The formulation is then given as:

$$\min_{x_{ij},u_i,v_i,z} \quad z \tag{2.10}$$

$$\text{s.t.} \quad \sum_{i \in N} x_{ij} = \sum_{i \in N} x_{ji} = 1 \qquad \forall j \in N \tag{2.11}$$

$$\sum_{j \in N} x_{sj} = \sum_{i \in N} x_{is} \leq |P| \tag{2.12}$$

$$u_i + v_i \leq z \qquad \forall i \in N \tag{2.13}$$

$$u_i + c_{ij} \leq u_j + M(1 - x_{ij}) \qquad \forall i, j \in N \tag{2.14}$$

$$v_j \leq v_i - c_{ij} + M(1 - x_{ij}) \qquad \forall i, j \in N \tag{2.15}$$

$$c_{si} \leq u_i \leq M - c_{is} \qquad \forall i \in N \tag{2.16}$$

$$c_{is} \leq v_i \leq M - c_{si} \qquad \forall i \in N \tag{2.17}$$

Equation (2.10) minimizes the maximum cycle time. Constraint (2.11) ensures that one agent services each sector. Constraint (2.12) limits the number of available agents from the source sector. The cycle time is constructed by Constraints (2.13), (2.14), and (2.15), using the concept of big-M constraints, which is at least as large as the summation of the $|N|$ largest travel times. Constraints (2.16) and (2.17) define the continuous domains of the time variables.

The key difference with dynamic coverage is that the goal is not to cover the area optimally once but recurrently. In this formulation, the maximum uncertainty is represented by the maximum intervisit time. Alternatively, prioritized locations can justify weighing the intervisit time as a measure of uncertainty. Nigam [7] extensively discusses the validity of this formulation approach, as he states that persistency always requires the formulation of a recurrent problem without time horizons. Given it is a complex problem, most research is dedicated to developing better models and solution methods, while some pursue the concept of utility functions.

Persistent reconnaissance by coverage level

Given a set of agents, how can their paths be planned to maximize the coverage of an area of interest within the planning horizon?

In contrast to the frequency approach, this problem can be formulated more clearly using a discrete-time POMDP, using the tuple $\langle S, A, T, R, \omega, O, \gamma \rangle$. The environment is in some state $s \in S$, which describes the position of the agents and the coverage level of the area of interest. The agent can take an action $a \in A$, after which the environment transitions to state s' depending on the conditional transition probabilities $T(s'|s, a, o)$. This transition can depend on various kinds of information, like area priorities, the time passed, or the observation of targets. Therefore it is also based on the observation $o \in \Omega$, generated by the conditional observation probabilities $O(o|s, a)$. In this case, the reward $r \in R(s)$ is dependent on the state, as it represents the coverage level. The goal of persistent coverage then is to maximize the expected future discounted reward $E\left[\sum_{t=0}^{\infty} \gamma^t r_t\right]$.

Though closely related to searching, the difference lies in the persistent character. Increasing the coverage level generates a deterministic reward for persistent coverage, as opposed to searching where the reward is a maximization of the probability of finding the target. In contrast, the observations during persistent coverage only influence the state, as it can affect the required coverage by transition probabilities T. An autonomous vacuum cleaner can therefore exemplify persistent coverage. Dust accumulates differently because of various causes, such that the vacuum cleaner should reposition continuously based on the current state and the expected future changes of dust levels. The principle difference with the approach from Nigam [7], is that recurrence is not optimized for directly. However, this is not necessarily worse because if the underlying model changes, a recurrent solution might become suboptimal. Therefore the frequency approach optimally covers a fixed environment indefinitely, while the coverage approach indefinitely adapts to a changing environment. As a solution method, Figure 2.1 shows that the main directions are optimal path planning, or swarming based on optimal control or simple behavior rules.

Summary

A first observation is that some models are better described using a POMDP, and others using a Mixed Integer Linear Program (MILP). The difference lies in the dependence on the transition probabilities that describe how an environment changes when performing specific actions. Especially important is when the targets react to the taken actions or if the possible actions are dependent on a particular state of the environment. For persistent reconnaissance, targets are not assumed reactive, and any path is allowed. In this case, a MILP is a more convenient approach, which can be solved using existing solvers.

Furthermore, there is a clear distinction between optimization problems over an indefinite horizon for a static problem and those that are optimized over a limited horizon recurrently. Based on the statements in Section 1-1, the assumption of a static problem cannot hold. Possibly, it can be solved recurrently, as with static coverage. However, given the persistent frequency approach is complex, this becomes computationally intractable quickly. On the other side, a qualitative assessment of the performance is much more complicated for a fixed time horizon, as it can only be performed during a time-extended simulation, and bounds are hard to provide.

2-2 Solution methods

Instead of discussing the quality of all different solution methods, the different approaches are classified in general, and their suitability for a military environment is discussed. A distinction is made between centralized and distributed approaches, where the latter is subdivided by cooperation type.

Centralized algorithms

Solution methods consist of exact algorithms, heuristics, and meta-heuristics. The algorithm choice depends on the problem, but a MILP is generally solved using branch & bound, branch & price, or branch & cut methods. A POMDP can be solved well using dynamic programming. Exact methods are not scalable due to the dependence on the underlying problem complexity. Furthermore, though effective heuristics can be implemented for some problems, they are very problem-specific and are not always available. Some examples are listed in this chapter that are fast, but these are pattern-based and thus not sufficiently adaptive to deal with a dynamic environment [42, 43]. Meta-heuristics are more broadly applicable [69], but often benefit significantly from domain knowledge and suffer from tractability for increasingly larger problems nonetheless.

Furthermore, as described in Section 1-2, only partial information through unreliable links is available. This eliminates the centralized method as a feasible approach, as this requires all information to be trustworthy and known in one place. In an unreliable network, such a task is cumbersome, if not impossible. Even if this was successful, it cannot be trusted that the solution would reach the actual system that needs to perform a task on time.

Distributed algorithms

Opposite to centralized is the distributed approach. Two types of distribution are distinguished. Firstly, distribution can exploit parallelism (like the ones arising from decomposition), to enhance performance and scalability. In practice, this often implies decentralization of an algorithm, in which multiple non-cooperating computation sources are active in parallel, and a central source performs synchronization. The second class arises from Multi-Agent systems (MASs), specifically Multi-Robot Systems (MRSs). Here, the distribution is natural as information is present locally, and no central agent is available to compute the global problem. This makes Distributed Constraint Optimization Problems (DCOPs) a suitable framework for describing this type of MRS. Solution approaches in this category include swarming algorithms, control, and utility functions.

Regardless of the type of algorithm, some form of cooperation is required to improve the overall solution value, as the agents share a common goal. For this application, cooperation can be categorized into three types:

- **Passive**: Each agent optimizes its problem individually, but agents communicate their local context, such that collective behavior emerges from reacting to the state of others. An example is using local optimization methods while sharing information [9].

- **Implicit**: Each agent optimizes its problem individually based on future moves of other agents, which is communicated along with the local context. This includes optimization [8] and utility functions [59], but consists mainly of swarming [16, 17].

- **Explicit**: All agents cooperatively optimize the problem to obtain a globally optimal solution by passing messages depending on the algorithm choice. This is generally solved using (exact) optimization methods [29, 34, 57, 58].

For all types of cooperation, communication is necessary to maintain performance. Therefore the available solution methods cannot deal with the challenges defined in Section 1-2, as there is no guaranteed robustness against communication failures.

2-3 Towards distributed coordination

This thesis aims to convert the principles of military intent-based Command & Control (C2), to overcome the issues that arise from using contemporary methods, as discussed in Section 2-2. The question is, does literature provide similar approaches? Does intent-based C2 fall within any existing framework for similar applications? And can lessons be learned or expected performance inferred?

First, the concept of coordination is clarified to ensure a valid link can be made with existing approaches. Then methods are discussed that try to tackle similar challenges, and lastly, the link to military intent-based C2 is made.

Terminology

In this thesis, coordination is defined as changing the parameters for agent behavior *a-priori*, separate from cooperation. This does not imply that coordination cannot be enforced on-line. However, it does mean that at any given moment agents cooperate, given the most recent set of coordinated parameters. As an example, consider the player roles within a football team. Players cooperate given their assigned roles, but after a mid-game player substitution, a coach often changes the roles itself (= coordination). In theory, the coach can even be replaced by a player voting system on the new roles. This exemplifies that coordination is not necessarily tied to a central authority but merely sets the parameters for the cooperative problem.

It should be noted that there is no consensus in the literature on this terminology. Farinelli et al. proposed a comparable taxonomy [70], in which cooperation is considered a broader term indicating whether agents operate together to achieve a common goal. The exact implementation of cooperation is viewed as a type of coordination. *Aware, not coordinated* systems correspond with *passive* cooperation. *Weak* coordination corresponds with *implicit* cooperation, and *strong* coordination with *explicit* cooperation. Furthermore, it also differs from the definition of Parker [71], in which coordinative interaction implies a network of agents that are aware of each other but do not share the same common goal.

The effect of organizations

One subject linked to mitigating the effects of a dynamic and unreliable environment is organizing MAS. For instance, Abbas et al. [72] analyse the merit of an organization-centered versus an agent-centered MAS (corresponding to distributed cooperation). They state that both the fully centralized or distributed method fall short for large, complex, and heterogeneous systems. These approaches are designed such that the global objective follows from individual actions, but for increasingly complex systems, this inevitably leads to unpredictable and uncertain outcomes. Instead, (dynamic) organizations should limit interactions, reduce uncertainty, and manage high-level goals no single agent is aware of.

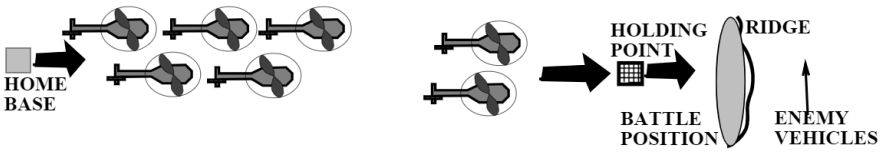
A complete overview of organizations for MAS is provided by Horling et al. [73]. For example, they discuss the concept of holarchies, which are a form of hierarchies with varying autonomy allowed for subordinates. In a holarchy, the chain of command generally goes up, but this is no strict interpretation. Holarchies are suitable for heterogeneous capabilities and show quick adaptation to dynamically changing conditions, but suffer from overall performance unpredictability. Another organization discussed is the team, which consists of cooperating non-selfish agents that pursue a shared objective. The main benefit is the explicit capability to reason about the effects of interactions, such that unforeseen conditions can be dealt with. The drawback is its increased communication. Other organizations include hierarchies, coalitions, congregations, societies, federations, markets, and matrix organizations, often deemed restrictive or overly complex.

Concept of teamwork

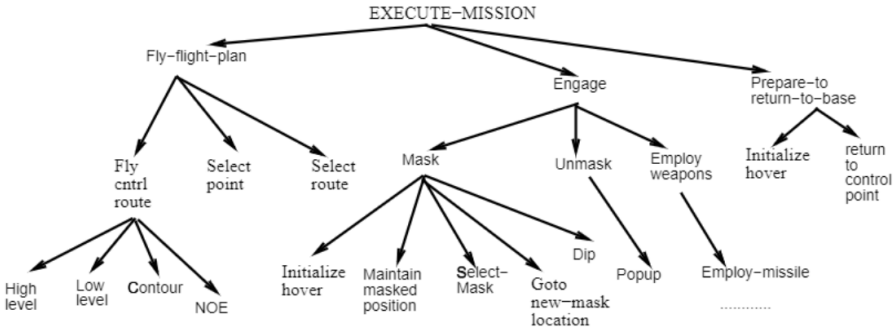
As discussed by Horling et al. [73], the team requires special attention as it is designed explicitly for MRSs in dynamic and unreliable environments. Though there are some key differences with the problems considered in teamwork, the concepts are closely related.

In general, the problem requiring teamwork is composing a plan of successive tasks to reach an objective. Therefore, it often draws from the well-known theories of Hierarchical Task Networks (HTNs) [74], or STRIPS [75], which is extended to MA-STRIPS for a MAS [76]. The multi-agent case is reviewed by Torreño et al. as Multi-Agent planning (MAP) [77]. The challenge for MAP is that some tasks are performed individually, but others jointly. Though MAP compromises a more general form of planning, the concept of teamwork is applied to MRSs. A typical application considered is RoboCup, but also military environments are frequently analysed. An example of an attack helicopter mission used by Tambe [78] is displayed in Figure 2.2.

Recently, Geihs [79] reviewed the current engineering challenges for teamwork and discussed two main frameworks: ALICA and STEAM. STEAM is a *joint intentions* implementation of the SharedPlans hierarchical decomposition [78]. A joint intention implies that a team mutually believes they are committed to completing a team action. Individuals can only desist if a new mutual belief is created that a joint task is completed, unattainable, or irrelevant [80]. The SharedPlans theory differs, as it describes how to achieve the common goal instead of only fixing it. This is achieved by constructing a mutual belief in the *recipe* before performing the separate individual or cooperative sub-tasks [81]. The core assumption for STEAM is that the possibility of unexpected failures or opportunities requires a flexible



(a) Example attack mission



(b) A pilot's task hierarchy

Figure 2.2: An example attack helicopter mission used in [78] to discuss teamwork for MRS.

approach. Inflexibility, which occurs with precomputed or slow implementations, can instead lead to severe failures. Though flexible, Geihs states that the requirement for establishing intentions makes it susceptible to communication failures. A comparable framework, called ALICA, is made by Skubch [82], drawing heavily from STEAM. The core difference is that an agent can start to act before a joint intention is established. Instead, estimations are used to guide decisions, and conflicts are detected and resolved afterward. The common thread through these approaches is that systems can continue without explicitly cooperating every step. In general, they all work towards a common goal.

Classifying intent-based C2

To summarize, for MRSs in dynamic and unreliable environments, two main research directions are found in the literature. First, organizing the network of agents should reduce uncertainty, manage high-level goals and increase the reaction speed. Secondly, teamwork is specifically designed to reason about the decisions of other systems, such that flexible response to unforeseen events remains possible. By comparison, military intent-based C2 can be considered a compound organization consisting of a top-down holarchy, combined with teamwork within each group. As communication remains critical for teamwork, the combined approach should reduce this drawback for larger networks. Following the given definition of coordination, both the organization structure, the amount of autonomy allowed at each level in the holarchy, and the implementation of teamwork can be considered coordination. All these decisions are made prior and define how agents cooperate and make decisions locally. Therefore, intent-based C2 can be classified as intent-based coordination for MRSs.

Chapter 3

Modelling Reconnaissance

In this chapter, the formulations are given for the Single- and Multi-Agent Reconnaissance Problem. These formulations provide a compact model, combining both the frequency and coverage level approach for persistent reconnaissance. The models are constructed to be solved recursively and do not include a time horizon for qualitative assessment purposes.

3-1 Single-Agent Reconnaissance Problem

The Single-Agent Reconnaissance Problem (SARP) is formulated over an indefinite horizon, which requires a recurrent path. A measure for risk increase is added to incorporate the frequency approach of persistent reconnaissance. For the coverage approach, also the acute risk values are included. This means it is worthwhile to visit specific sectors first, which might increase the cycle distance.

The problem space can be defined as a complete graph $G = (\mathcal{V}, \mathcal{E})$, with vertices $\mathcal{V}$ consisting of a set $\mathcal{N}$ of equally sized sectors and the source vertex s, and the undirected edges $\mathcal{E}$ describe all paths. Each sector $i \in \mathcal{N}$ has a current risk r_i^t at time t, and a risk increase $\dot{r}_i^t$. Though it will be shown that the risk increase is not linear, it is fixed to the current value each time the problem is solved. A weight ρ is used to balance the penalty on the risk increase and current risk. The decision variable x_{ij} describes if vertex j is visited after vertex $i \in \mathcal{V}$. The distance traveled between vertices is given by c_{ij}, modelled as the Euclidean distance (2-norm). The variable u_i describes the distance traveled from the source s to vertex i, and v_i describes the distance traveled after vertex i back to the source vertex s. The variable z then represents the cycle distance.

Objective function (3.1) minimizes the risk weighed by the distance after which it is visited, combined with the risk increase weighed by the full cycle distance. Constraint (3.2) forces the in- and outflow to be equal for each sector. Constraint (3.3) ensures that each sector is visited exactly once. Constraint (3.4) and Constraint (3.5) describe the minimum distance from the source and the minimum distance back to the source, respectively. These constraints function as a big-M constraint, where in this case $M = |\mathcal{N}| \cdot \max(c_{ij})$. Then, Constraint (3.6) combines these to a minimum cycle distance. Lastly, Constraint (3.7) and (3.8) set the domain of the decision variables. The complete formulation then becomes:

$$\min_{x_{ij},u_i,v_i,z} \quad J = \rho \cdot \sum_{i\in\mathcal{N}} r_i^t u_i + (1-\rho)\cdot \sum_{i\in\mathcal{N}} \dot{r}_i^t z \tag{3.1}$$

$$\text{s.t.} \quad \sum_{j\neq i\in\mathcal{V}} x_{ji} = \sum_{j\neq i\in\mathcal{V}} x_{ij} \qquad \forall i\in\mathcal{N} \tag{3.2}$$

$$\sum_{j\in\mathcal{V}} x_{ij} = 1 \qquad \forall i\in\mathcal{N} \tag{3.3}$$

$$u_i + c_{ij} \leq u_j + |\mathcal{N}|\cdot\max(c_{ij})\cdot(1-x_{ij}) \qquad \forall i\in\mathcal{V},\ j\in\mathcal{N} \tag{3.4}$$

$$v_j \leq v_i - c_{ij} + |\mathcal{N}|\cdot\max(c_{ij})\cdot(1-x_{ij}) \qquad \forall i\in\mathcal{N},\ j\in\mathcal{V} \tag{3.5}$$

$$u_i + v_i \leq z \qquad \forall i\in\mathcal{N} \tag{3.6}$$

$$x_{ij}\in\{0,1\} \qquad \forall i,j\in\mathcal{N} \tag{3.7}$$

$$u_i, v_i \geq 0 \qquad \forall i\in\mathcal{N} \tag{3.8}$$

3-2 Multi-Agent Reconnaissance Problem

The Multi-Agent Reconnaissance Problem (MARP) is an extension of the SARP, which closely resembles the extension of the Traveling Salesman Problem (TSP) towards the Vehicle Routing Problem (VRP). The MARP aims to solve the problem for a set of agents $\mathcal{P}$ combined. Each agent has a specific speed v_k and flight time e_k $\forall k\in\mathcal{P}$, as the problem allows heterogeneous agents. The decision variables are extended in the agent dimension to x_{ijk}, u_{ik}, v_{ik}, and z_k. Furthermore, the problem is now described as a multigraph $G=(\mathcal{V},\mathcal{E})$, with the vertices $\mathcal{V}$ including all sources s_k $\forall k\in\mathcal{P}$, and the edges being composed of subsets belonging to each agent, such that $\mathcal{E}=\bigcup_{k\in\mathcal{P}}\mathcal{E}_k$. Then, all vertices connected to any edge of agent k can be collected as $\mathcal{V}_k=\{i \mid \exists\{i,j\}\in\mathcal{E}_k\ \forall j\in\mathcal{V}\}$. This edge-induced subgraph $G_k[\mathcal{E}_k]$ is always complete. This implies not all agents have access to all sectors, but if it does, an edge exists to reach it from any other included sector. A Boolean A_{ik} indicates if $\exists i\in\mathcal{V}_k$, which is used to limit the decision variables. The complete formulation is then given as:

$$\min_{x_{ijk},u_{ik},v_{ik},z_k} \quad J = \rho\cdot\sum_{k\in\mathcal{P}}\frac{\min_{k\in\mathcal{P}}(v_k)}{v_k}\sum_{i\in\mathcal{N}} r_i^t u_{ik} + (1-\rho)\cdot\sum_{k\in\mathcal{P}}\frac{\min_{k\in\mathcal{P}}(e_k)}{e_k}\sum_{i\in\mathcal{N}}\dot{r}_i^t z_k \tag{3.9}$$

$$\text{s.t.} \quad \sum_{j\in\mathcal{V}} x_{ijk} \leq A_{ik} \qquad \forall i\in\mathcal{V},\ k\in\mathcal{P} \tag{3.10}$$

$$\sum_{j\neq i\in\mathcal{V}} x_{jik} = \sum_{j\neq i\in\mathcal{V}} x_{ijk} \qquad \forall i\in\mathcal{N},\ k\in\mathcal{P} \tag{3.11}$$

$$\sum_{k\in\mathcal{P}}\sum_{j\in\mathcal{V}} x_{ijk} = 1 \qquad \forall i\in\mathcal{N} \tag{3.12}$$

$$u_{ik} + c_{ij} \leq u_{jk} + |\mathcal{N}|\cdot\max(c_{ij})\cdot(1-x_{ijk}) \qquad \forall i\in\mathcal{V},\ j\in\mathcal{N},\ k\in\mathcal{P} \tag{3.13}$$

$$v_{jk} \leq v_{ik} - c_{ij} + |\mathcal{N}|\cdot\max(c_{ij})\cdot(1-x_{ijk}) \qquad \forall i\in\mathcal{N},\ j\in\mathcal{V},\ k\in\mathcal{P} \tag{3.14}$$

$$u_{ik} + v_{ik} \leq z_k \qquad \forall i\in\mathcal{N},\ k\in\mathcal{P} \tag{3.15}$$

$$x_{ijk}\in\{0,1\} \qquad \forall i,j\in\mathcal{N},\ k\in\mathcal{P} \tag{3.16}$$

$$u_{ik}, v_{ik} \geq 0 \qquad \forall i\in\mathcal{N},\ k\in\mathcal{P} \tag{3.17}$$

Objective (3.9) penalizes covering higher risk after larger distances, balanced with a penalty on longer cycle distances for higher risk increase. The resulting penalty depends on the agents' relative speed and flight time. As recharging is not explicitly accounted for, it is assumed that agents with a longer flight time can better sustain the cycle distance. For acute risk, high velocity is essential. Logically, a quicker agent can cover the same distance in a shorter time. Constraint (3.10) is added to explicitly limit the sectors that each agent can visit. It follows from the edge-induced subgraph $G_k[\mathcal{E}_k]$, which can be interpreted as the sectors allocated to an agent. This notion is important, as it is used in the remainder of this thesis. Constraint (3.11) forces the in- and outflow to be equal for each sector. Constraint (3.12) ensures that each sector is visited exactly once by an agent. Constraint (3.13), (3.14), and (3.15) set the distance to arrival, return and the complete cycle, respectively. Constraint (3.16) and (3.17) define the variable domains.

3-3 Risk evolution

Both the SARP and MARP are based on covering risk and risk increase optimally. These values are determined at any time instance t, based on predefined sector priorities M_i, dispersion of risk γ_i^t, agent presence $u_i^t \ \forall i \in \mathcal{N}$, and constants S and D. Combined, the model for risk evolution can be described with a set of equations. First, the sector uncertainty δ_i^{t+1} is calculated, which is defined within a $[0, 1]$ interval:

$$\delta_i^{t+1} = \min\{(\delta_i^t + \gamma_i^t + S \cdot M_i - u_i^t)^+,\ 1\} \qquad \forall i \in \mathcal{N} \tag{3.18}$$

The next uncertainty value is determined as the current uncertainty, possibly increased by dispersion γ_i^t from neighboring sectors, the static increase $S \cdot P_i$ and decreased by the presence indicator $u_i^t = 1$ if agents are in the sector. Dispersion from neighboring sectors is given as:

$$\gamma_i^t = D \cdot M_i \cdot (\bar{\delta_i}^t - \delta_i^t)^+ \qquad \forall i \in \mathcal{N} \tag{3.19}$$

Here, $\bar{\delta_i}^t$ represents the average uncertainty of the surrounding sectors, and M_i is the sector traversability, which depends on the terrain. The constant D is used to tune the effect of dispersion altogether. Finally, the sector risk at the next time instance is given as:

$$r_i^{t+1} = P_i \cdot \delta_i^{t+1} \qquad \forall i \in \mathcal{N} \tag{3.20}$$

In this equation, the sector uncertainty leads to a specific value for risk depending on the sector priority. The risk increase is then defined as the slope of this function, approximated using the central difference theorem:

$$\dot{r}_i^{t+1} = \frac{r_i^{t+2} - r_i^t}{2} = P_i \frac{\delta_i^{t+2} - \delta_i^t}{2} \qquad \forall i \in \mathcal{N} \tag{3.21}$$

On average, $\dot{r}_i^t$, will be much smaller than r_i^t. To balance the risk and risk increase terms in Objective (3.1) and (3.9), the risk increase term is chosen to be approximately D times larger. This implies that $\rho = \frac{D}{D+1}$ and $1 - \rho = \frac{1}{D+1}$. Based on practical experience, the constants $D = 0.01$ and $S = 0.1$ are implemented.

Chapter 4

Benchmarking

As stated in Section 2-2, calculating the optimal solution to the Multi-Agent Reconnaissance Problem (MARP) is a time-consuming operation, justifying the development of intent-based coordination for separate Single-Agent Reconnaissance Problems (SARPs). Therefore, a method is needed to compare the obtained solutions to the optimal solution of the MARP. In this chapter, methods are proposed to retrieve both lower and upper bounds.

4-1 Lower bound

Fundamentally, the optimal solution for the MARP can be interpreted as an allocation of sectors to agents, which are ordered in a sequence for each agent separately. Theoretically, the optimal solution can therefore be obtained by limiting the agent-specific vertices $\mathcal{V}_k \; \forall k \in \mathcal{P}$ *prior* to solving the MARP, such that it decomposes to independent SARPs. Therefore, the lower bound is rooted in the central question:

How can the decrease in solution quality by limiting the accessibility of agents to a specific subset of the sectors be assessed?

Again, the relation with the Vehicle Routing Problem (VRP) is evident. Given an optimal allocation of destinies, the VRP decomposes in separate Traveling Salesman Problems (TSPs), such that the optimal solution to the VRP will be obtained. This boils down to sensitivity analysis for integer programming: given that an agent is allocated the optimal set of vertices $\mathcal{V}_k$, adding any new vertex would not change the solution, rendering it insensitive to any additional allocations.

As Güzelsoy and Ralphs [83] summarized concisely, duality for a Mixed Integer Linear Program (MILP) can be described using a *value function*, which returns the optimal solution for any right-hand side. Finding such a function is at least as complex as solving the primal problem, which brings no benefit. An alternative is generating weak dual functions, which might still be used to analyse the effect of changing input data. As they point out, the most promising approach is to generate these functions as a byproduct of primal solution methods like branch & cut. However, the results are preliminary and more development is still necessary. Nonetheless, it is already applied in branch & price for the linearized case, which is known to produce tight lower bounds.

4-1-1 Reformulation

Branch & price functions by pricing the benefit of adding sectors to the subset of an agent for a linearly relaxed case, after which branching is used to remove any fractional allocations gradually. The pricing stage consists of repeatedly performing column generation and solving the linearly relaxed, reformulated version of the original problem. This reformulation is based on the set covering problem and answers the question: what sequences from $r \in \Omega$, with Ω being all existing sequences, need to be assigned to which agent $k \in \mathcal{P}$ to cover the sectors $\mathcal{N}$ optimally? The reformulation of the MARP is written as:

$$\min_{x_{rk}} \quad \sum_{k\in\mathcal{P}} \sum_{r\in\Omega} J_{rk} x_{rk} \tag{4.1}$$

$$\text{s.t.} \quad \sum_{k\in\mathcal{P}} \sum_{r\in\Omega} a_{ir} x_{rk} = 1 \qquad \forall i \in \mathcal{N} \tag{4.2}$$

$$\sum_{r\in\Omega} x_{rk} \leq 1 \qquad \forall k \in \mathcal{P} \tag{4.3}$$

$$x_{rk} \in \{0,1\} \qquad \forall r \in \Omega,\ k \in \mathcal{P} \tag{4.4}$$

Objective (4.1) assigns the optimal sequence to each agent, based on the cost J_{rk} for agent $k \in \mathcal{P}$ executing sequence $r \in \Omega$. This value is easily calculated using the objective function of the MARP. Constraint (4.2) ensures that each sector is visited once, represented by the constant a_{ir} describing if sector $i \in \mathcal{N}$ is included in sequence $r \in \Omega$. Constraint (4.3) and (4.4) limit the number of allocations per sensor to either zero or one. For column generation, this problem is labeled the Master Problem (MP).

Composing the sheer amount of sequences in Ω becomes intractable quickly and prohibits solving even a single relaxation of the problem. Therefore, this problem is changed to the Restricted Master Problem (RMP), such that $\Omega \Rightarrow \Omega^r$, where Ω^r is the restricted set of sequences. Theoretically, this limited set would only have to include the optimal set of $|\mathcal{P}|$ sequences. However, these sequences are, of course, still unknown. The pricing procedure can find those sequences for the relaxed RMP, in which Constraint (4.4) becomes $x_{rk} \geq 0$. If pricing does not produce any new sequences for the relaxed RMP, it is optimal. If the solution is integral, also no new sequences exist for the RMP. Then, it can be concluded that Ω_r includes the optimal sequences from Ω, which implies that the solution is also optimal for the MP. Pricing the relaxed RMP is done by a column generation procedure, called the Sub Problem (SP). Effectively, during this procedure, the sensitivity analysis is performed for the linearized problem using duals. This translates to finding a sequence with a negative reduced cost:

$$\hat{J}_{rk} = J_{rk} - \sum_{i\in N} a_{ir}\lambda_i - \lambda_k < 0 \qquad \forall r \in \Omega,\ k \in \mathcal{P} \tag{4.5}$$

Here, λ_i and λ_k are the duals obtained by solving the relaxed RMP, and J_{rk} is the objective value of the sequence for sensor $k \in \mathcal{P}$. If a sequence with negative reduced cost is found, it is included in Ω^r, and the procedure is repeated. If not, the solution to the relaxed RMP is optimal. As a lower bound is pursued, a fractional solution is acceptable. Instead, if the optimal solution is required, one can extend this pricing step with a branching scheme.

The SP is an optimization problem itself, called the Elementary Shortest Path Problem (ESPP). If no more negative reduced sequences exist, there are no more beneficial allocations for any sensor, meaning:

$$\min_{r\in\Omega,\ k\in\mathcal{P}}\{\hat{J}_{rk}\}\geq 0 \tag{4.6}$$

This gives rise to $|\mathcal{P}|$ separate SPs, as for each $k\in\mathcal{P}$ the ESPP is formulated as:

$$\min_{x_{ij},u_i,v_i,z}\quad \hat{J}_{rk}=\rho\cdot\frac{\min_{k\in\mathcal{P}}(v_k)}{v_k}\sum_{i\in N}r_i^t u_i+(1-\rho)\cdot\frac{\min_{k\in\mathcal{P}}(e_k)}{e_k}\sum_{i\in\mathcal{N}}\dot{r}_i^t z-\sum_{i,j\in\mathcal{V}}x_{ij}\lambda_i-\lambda_k \tag{4.7}$$

$$\text{s.t.}\quad \sum_{j\in\mathcal{V}}x_{ij}\leq 1 \qquad \forall i\in N \tag{4.8}$$

$$\sum_{j\neq i\in\mathcal{V}}x_{ji}=\sum_{j\neq i\in\mathcal{V}}x_{ij} \qquad \forall i\in N \tag{4.9}$$

$$u_i+c_{ij}\leq u_j+|\mathcal{N}|\cdot\max(c_{ij})\cdot(1-x_{ij}) \qquad \forall i\in\mathcal{V},\ j\in\mathcal{N} \tag{4.10}$$

$$v_j\leq v_i-c_{ij}+|\mathcal{N}|\cdot\max(c_{ij})\cdot(1-x_{ij}) \qquad \forall i\in\mathcal{N},\ j\in\mathcal{V} \tag{4.11}$$

$$u_i+v_i\leq z \qquad \forall i\in\mathcal{N} \tag{4.12}$$

$$x_{ij}\in\{0,1\} \qquad \forall i,j\in\mathcal{N} \tag{4.13}$$

$$u_i,v_i\geq 0 \qquad \forall i\in\mathcal{N} \tag{4.14}$$

This SP is comparable to the SARP, with two important changes. First, the Objective (4.7) is adjusted, including the relative speed and flight time, and the duals obtained from solving the relaxed RMP. Furthermore, Constraint (4.8) now states that every sector can be visited at most once instead of exactly once. This means that a sequence of any length can be formed to achieve the negative reduced cost. The remainder of the constraints is not changed, with Constraint (4.9) preserving the flow over each sector, and Constraints (4.10), (4.11), and (4.12) setting the preceding, succeeding and cycle distance respectively. Constraint (4.13), and (4.14) define the domain.

Though it is a complex problem to solve as it is NP-hard [84], the problem does not have to be solved to optimality every time. Any solution that results in $\hat{J}_{rk}<0$ can directly be added to Ω^r . However, to ensure that the optimum for the relaxed RMP has been found, it has to be solved to optimality at least once.

4-1-2 Column generation

As described in the previous section, column generation is the procedure applied to perform pricing, such that the relaxed RMP can be solved to optimality. Columns are generated if they lead to a negative reduced cost. To this end, the SP needs to be solved. Here, the methods will be discussed to solve this problem. First, the heuristics are described, which are used to obtain *any* negative reduced cost solution. Second, also the forward labeling method is discussed, which is used to solve the SP to optimality.

Heuristics

Three heuristics are proposed to find sequences with negative reduced cost quickly: greedy, insertion, and reshuffling. The heuristics are applied for each agent $k \in \mathcal{P}$ separately, with the corresponding speed v_k, flight time e_k, and source sector $s_k \in \mathcal{N}$. The heuristics are adaptions of the most commonly known variants, displayed concisely by Nilsson [85].

Greedy The first heuristic is the simplest. The idea is that a sequence of each possible length is created by appending a current sequence greedily and saving the intermediate result in a list. Upon termination, the best sequence is returned. It is added to the restricted set of sequences Ω^r for the RMP if it has a negative reduced cost. Otherwise, it is discarded. The pseudo-code is displayed in Algorithm 1.

Algorithm 1: Greedy sequence for agent $k \in \mathcal{P}$ and sectors $i \in \mathcal{N}$

```
sequence ← s_k
sequence_list ← ∅

while sectors unvisited do
    foreach unvisited sector do
        Collect potential reduced cost if sector is appended to the sequence
    sequence.add(min cost(sector))
    sequence_list.add(sequence)

best_sequence ← min cost(sequence_list)
return best_sequence
```

Insertion In the second heuristic, more possibilities are considered. Instead of only adding sectors at the end of the current sequence, it is possible to insert the new sector in any position. In Algorithm 2, the adjusted pseudo-code is shown.

Algorithm 2: Insertion sequence for agent $k \in \mathcal{P}$ and sectors $i \in \mathcal{N}$

```
sequence ← s_k
sequence_list ← ∅

while sectors unvisited do
    foreach unvisited sector do
        foreach position in sequence do
            Collect potential reduced cost if sector is inserted at position in the sequence
        new_sequence(sector) ← sequence ∪ min cost(position)
    sequence ← min cost(new_sequence)
    sequence_list.add(sequence)

best_sequence ← min cost(sequence_list)
return best_sequence
```

Reshuffling As an extra addition to the insertion heuristic, the reshuffling heuristic also allows reordering a sequence after a new sector has been inserted. The idea is that iteratively an edge $\{i, j\} \in \mathcal{E}$ between two sectors $i, j \in \mathcal{N}$ is selected that seems profitable to add. Then, the set of all selected edges are connected to form a sequence. Therefore, each time a new sequence is added, it grows in length and can be shuffled in order. The functioning is displayed in Algorithm 3.

Algorithm 3: Reshuffling sequence for agent $k \in \mathcal{P}$, sectors $i \in \mathcal{N}$ and edges $\{i, j\} \in \mathcal{E}$

```
edge_list ← ∅
sequence_list ← ∅

while edges available do
    foreach available edge do
        Calculate approximated added reduced cost Ĵ_ijk

    if Any added negative reduced cost edge found then
        Save minimum cost edge {i, j} = {out, in}                    /* process edge */
        Remove all edges to j
        Remove all edges from i
        Remove the reverse edge {j, i}.
        edge_list.add(edge)

        Connect sector j if edges {i, j}, {j, j'} ∀i, j' ∈ V included  /* build sequence */
        if no source edge {s_k, j} ∀j ∈ V included then
            Add min cost source edge to sector i of any included edge {i, j}
        while unconnected edges do
            Add min cost edge {i, j} for any included edges {i', i}, {j, j'}  ∀i', j' ∈ V
        sequence ← connected edges
        Calculate actual reduced cost of sequence
    else
        Break search
    sequence_list.add(sequence)

best_sequence ← min cost(sequence_list)
return best_sequence
```

Algorithm 3 builds upon the concept of approximated added reduced cost. This approximation aims to find beneficial edges, by making the following calculation:

$$\hat{J}_{ijk} = \rho \cdot \frac{\min_{k \in \mathcal{P}}(v_k)}{v_k} \sum_{i \in N} r_j \hat{u}_j + (1 - \rho) \cdot \frac{\min_{k \in \mathcal{P}}(e_k)}{e_k} \sum_{i \in \mathcal{N}} \dot{r}_i \hat{z} - \lambda_j \qquad \forall i, j \in \mathcal{V}, k \in \mathcal{P} \quad (4.15)$$

This approximation includes the duals, and a lower bound on the distance $\hat{u}_j = c_{s_k,j} + c_{ij}$ and cycle distance $\hat{z} = \hat{u}_j + c_{j,s_j}$. If $\hat{J}_{ijk} \geq 0$, including the edge $\{i, j\}$ will never lower the actual reduced cost. Conversely, $\hat{J}_{ijk} < 0$ provides no guarantee that adding it will improve the solution, but it does give some indication on which edges provide greater benefits, which should lead to a sensible solution.

Forward labeling

If the heuristics are unable to find a solution to the SP with negative reduced cost, an exact method is needed to ensure no more sequences with negative reduced cost exist.

Background The Shortest Path Problem (SPP) can be solved to optimality using dynamic programming, for which Desrocher proposed a label correction algorithm [86]. Feillet et al. [3] adjusted this for the ESPP, such that it does not allow cycles. This is important, as the SP regularly has negative cost edges. In general, this algorithm functions by extending labels successively while applying dominance rules to remove labels guaranteed to be suboptimal. This exhaustive method terminates when no label can be extended anymore, returning the optimal solution. Righini et al. propose two improvements over this algorithm [87]. First, they introduce bi-directional labeling. Secondly, they include bounding to discard unpromising states and stop the extension of non-dominated labels to allow efficient joints of labels. It is shown that bi-directional labeling has a significant advantage over mono-directional forward labeling. In addition, Righini et al. [88] compare this method to the decremental state-space relaxation, which is found to perform better, especially when the constraints of the SP are less tight. They indicate that there is a trade-off to be made between computation time and lower bound tightness, which is in large dependent on whether a form of exact or relaxed pricing is implemented. This trade-off is further examined in the survey by Pugliese et al. [89]. Lastly, pricing for the Multi-Depot Heterogeneous Vehicle Routing Problem (MDHVRP) is discussed by Bettinelli et al. [90]. They propose an aggregated label in which the cost structure is included for multiple depots. The heterogeneous application considered here draws from this implementation and is built using the clear overview provided by Feillet [91].

Concept During forward labeling, a label $L = \langle i, \mathcal{W}, \mathcal{U}_k, d_k, c_k, c_k^z, \hat{c}_k, \hat{c}^{lb} \rangle$ is created with $i \in \mathcal{V}$ being the current vertex, d_k the cumulative sequence distance for each agent to the current vertex, $\mathcal{W}$ is a set consisting of all vertices not yet visited, $\mathcal{U}_k$ the linearly ordered set describing the sequence for each agent thus far, and c_k, c_k^z and $\hat{c}_k$ describe the cost, cycle cost, and reduced cost for each agent $k \in \mathcal{P}$, respectively. The variable $\hat{c}^{lb}$ is a lower bound on the reduced cost obtained by extending the label. During each iteration, the label with the minimum lower bound $\hat{c}^{lb}$ is selected. By pursuing this label, a best-first strategy is implemented. The complete pseudo-code is provided in Algorithm 4. In line 14, it shows the search is broken off as soon as the first sequence with negative reduced cost has been found. If this line is removed, the algorithm will always continue to optimality.

When a label L is extended to L', it is checked first if it is beneficial, by calculating if the dual step cost $\hat{c}_j^{step} < 0$. This cost represents the added reduced cost that is obtained when making the extension. As the subgraph $G_k[\mathcal{E}_k]$ is known to be complete (see Section 3-2) and the triangle inequality holds as the Euclidean distance is used, an extension that increases the reduced cost can never be part of an optimal negative reduced cost sequence. First, the step cost from sector $L\langle i \rangle$ to sector $L'\langle j \rangle$ is calculated as follows:

$$c_{L\langle j \rangle,k}^{step} = \rho \cdot \frac{\min_{k \in \mathcal{P}}(v_k)}{v_k} r_j \cdot L'\langle d_k \rangle + L'\langle c_k^z \rangle - L\langle c_k^z \rangle \qquad \forall j \in \mathcal{V}, k \in \mathcal{P} \tag{4.16}$$

Then, the dual step cost to sector j is given as:

$$\hat{c}_{L\langle j \rangle,k}^{step} = c_{L\langle j \rangle,k}^{step} - \lambda_j \qquad \forall j \in \mathcal{V}, k \in \mathcal{P} \tag{4.17}$$

Essentially, this describes the change in the Objective value (4.7) from the SP (see Section 4-1-1). This added cost uses the difference in cycle cost c_k^z after extension to L'.

Algorithm 4: Forward labeling for agent $k \in \mathcal{P}$ and sectors $i \in \mathcal{V}$

1 Initialize *labels* with label L at current sector i
2 $best\_sequence \leftarrow \emptyset$
3 $\hat{c}^{best} = 0$

4 **while** *labels* **do**
5 Select $\min(\hat{c}^{lb})$ as current label L;
foreach *sector* $j \in \mathcal{W}$ *of current label* L **do**
6 Extend L to new label L' to sector j at t, with $\mathcal{W}$ and c_k^z
7 Calculate dual step cost $\hat{c}_j^{step}$

8 **if** j *is a sink or* $\hat{c}_j^{step} < 0$ **then** `/* Check if beneficial step */`
9 Extend new label L' with $\mathcal{U}_k$, c_k, $\hat{c}_k$, and $\hat{c}^{lb}$
10 **else**
11 Discard new label L'

12 **if** j *is a sink and any* $\hat{c}_k < \hat{c}^{best}$ **then** `/* Check if sequence found */`
13 $best\_sequence \leftarrow \min \text{cost}(u_k)$
14 Break search
15 **else**
16 Discard new label L'

17 **if** j *is not a sink and* $\hat{c}^{lb} < \hat{c}^{best}$ **then** `/* Check if beneficial label */`
18 **if** *agent dominated by other label* **then** (partially) discard new label L'
19 **if** *agent dominates other labels* **then** (partially) discard other labels
20 **else**
21 Discard new label L'
22 Discard current label L

23 **return** *best_sequence*

Extension As visible in Algorithm 4, the extension of the label is split into two parts, at lines 6 and 9. It is possible to do this at once, but this leads to unnecessary computations as labels can be discarded already based on the dual step cost. Nonetheless, all extensions are described here at once. For a new label $L' = \langle j, \mathcal{W}, \mathcal{U}_k, d_k, c_k, c_k^z, \hat{c}_k, \hat{c}^{lb} \rangle$, the extension can be described using the following formulas:

$$\mathcal{W} = L\langle \mathcal{W} \rangle \setminus L'\langle j \rangle \tag{4.18}$$

$$d_k = \begin{cases} c_{s_k, L'\langle j \rangle} & \text{if } L'\langle i \rangle \notin \mathcal{N} \\ L\langle d_k \rangle + c_{L\langle i \rangle, L'\langle j \rangle} & \text{otherwise} \end{cases} \qquad \forall k \in \mathcal{P} \tag{4.19}$$

$$\mathcal{U}_k = L\langle \mathcal{U}_k \rangle \cup \{L'\langle j \rangle, L'\langle d_k \rangle\} \qquad \forall k \in \mathcal{P} \tag{4.20}$$

$$c_k^z = (1-\rho) \cdot \frac{\min_{k \in \mathcal{P}}(e_k)}{e_k} \cdot (L'\langle d_k \rangle + c_{L'\langle j \rangle, s_k}) \cdot \sum_{i \notin V, \in \mathcal{N}} \dot{r}_i \qquad \forall k \in \mathcal{P} \qquad (4.21)$$

$$c_k = \begin{cases} \infty & \text{if } L'\langle j \rangle \notin \mathcal{V}_k \\ L\langle c \rangle + c^{step}_{L'\langle j \rangle} & \text{otherwise} \end{cases} \qquad \forall k \in \mathcal{P} \qquad (4.22)$$

$$\hat{c}_k = \begin{cases} -\lambda_k & \text{if } L'\langle i \rangle \notin \mathcal{N} \\ L\langle \hat{c} \rangle + \hat{c}^{step}_{L'\langle j \rangle} & \text{otherwise} \end{cases} \qquad \forall k \in \mathcal{P} \qquad (4.23)$$

$$\hat{c}^{lb} = \min_{k \in \mathcal{P}} \left\{ \min \left(0, \hat{c}^{step}_{L''\langle j'' \rangle, k} \right) \right\} \qquad (4.24)$$

Dominance In Algorithm 4, dominance rules are applied in lines 18 and 19. Those are critical in reducing the state space and speeding up the algorithm. Furthermore, bounding is implemented to discard a solution preemptively. First, labels are removed immediately if the dual step cost $\hat{c}^{step}_j \geq 0$ or if the lower bound $\hat{c}^{lb} \geq \hat{c}^{best}$, as it can not improve the best solution anymore.

If these criteria did not disqualify a label, dominance rules are applied to check both if any label dominates the new label or if the new label dominates any other label. The difficulty is that, in this case, the labels can also be partially dominated, as they contain different costs of multiple agents. For an agent $k \in \mathcal{P}$, the label L_1 is considered to dominate label L_2 if the following holds true:

1. $L_1\langle i \rangle = L_2\langle i \rangle$
2. $L_1\langle \mathcal{W} \rangle \supseteq L_2\langle \mathcal{W} \rangle$
3. $L_1\langle d_k \rangle \leq L_2\langle d_k \rangle$
4. $L_1\langle \hat{c} \rangle \leq L_2\langle \hat{c} \rangle$

Criterion 1 checks if the labels have the same current sector. Criterion 2 states that any expansion from 2 is also allowed for 1. Criterion 3 verifies that the distance of label 1 is the same size or smaller than that of label 2. The last criterion ensures that the negative reduced cost of label 1 is equal or smaller. Combined, label 1 is guaranteed to produce the same solution or better for agent $k \in \mathcal{P}$, meaning its values do not have to be extended in L_2. If the label cannot be extended anymore for any agent, it is removed completely.

Summary Initially, the heuristics are used to generate sequences. If unsuccessful, forward labeling is used, which can continue to optimality to ensure no sequences exist anymore. This implies the relaxed RMP is optimal. Performing forward labeling is computationally intensive, as the SP is NP-hard. Some main improvements are known but not implemented yet. The main directions are extending forward labeling to a bi-directional algorithm, which is expected to perform much better [87]. Furthermore, the concept of decremental state-space relaxation can improve the solution quality even more [88]. Additionally, relaxations can be applied by, for example, relaxing criterion 1 [90] or by removing sectors that are not expected to be included in the solution [89], which can improve the speed at the cost of less tight lower bounds.

4-2 Upper bound

Any feasible solution for the MARP can be considered an upper bound. Generally, a trivial method is used to generate a feasible solution, such that the proposed method can be compared to this 'naive' approach. Obviously, the trivial method should be outperformed to justify the use of a more comprehensive approach. For this problem, a feasible solution consists of allocating sectors to agents and finding a sequence for each agent visiting the allocated sectors. A trivial solution needs to be generated only for the allocation into separate SARPs, such that a direct comparison with the subsets from intent-based coordination is provided.

Trivial clustering

To create n trivial clusters from $\mathcal{N}$ sectors, n random sectors are selected as cluster centroids. Sequentially, a random neighboring sector is added to each cluster, which implies it is adjacent to at least one sector in the cluster. If all remaining sectors are not neighbors to any cluster, a random sector is selected and added to the closest cluster. As such, clusters are not necessarily completely connected. It is trivial as opposed to random, as it is reasonable to believe that it is more efficient to cover close or connected sectors than a cluster that is completely scattered.

Trivial Allocation

Instead of focusing on adjacency, the allocation is concerned with the capacity of the agents involved in the MARP, but no heterogeneity is taken into account. In each iteration, a random cluster is chosen. Then, this cluster is added to the agent with the lowest number of total sectors allocated, balancing the workload between the agents in general.

4-3 SARP solution

Both intent-based coordination and the trivial allocation generate subsets for separate SARPs. These methods can only be compared if the resulting SARPs are solved in an equal manner. It can be solved to optimality using existing solvers, but this is not tractable for larger problem instances. Instead, heuristics can be used comparable to those for a TSP, which are concisely discussed by Nilsson [85]. Finding the most efficient heuristics is a topic of research in itself, which is out of scope for this thesis. Therefore, it is chosen to modify the heuristics used for the ESPP in Section 4-1-2.

Greedy & Insertion heuristic

Algorithms 1 and 2 are easily modified by requiring that all sectors are visited and that the actual instead of the reduced cost is used. This simplifies the heuristic, as no list of sequences is built. Instead, one sequence is expanded such that the final result is immediately accepted as the solution. Due to the simplicity of the adjustments, they are not displayed separately.

Reshuffling heuristic

Furthermore, the reshuffling heuristic is modified. Again, no list of sequences is necessary as all sectors need to be visited. Therefore, this algorithm is adjusted such that all low-cost edges are selected first, after which the edges are connected greedily to create a single sequence. The resulting heuristic is displayed as Algorithm 5.

Algorithm 5: Reshuffling sequence for agent $k \in \mathcal{P}$, sectors $i \in \mathcal{V}$ and edges $\{i, j\} \in \mathcal{E}$

```
edge_list ← ∅

while edges available do
    foreach available edge do
        Calculate approximated added cost J_ijk
    Save minimum cost edge {i, j} = {out, in}          /* process edge */
    Remove all edges to j
    Remove all edges from i
    Remove the reverse edge {j, i}.
    edge_list.add(edge)

Connect sector j if edges {i, j}, {j, j'} ∀i, j' ∈ V included   /* build sequence */
while unconnected edges do
    Add min cost edge {i, j} for any included edges {i', i}, {j, j'}  ∀i', j' ∈ V
sequence ← connected edges
Calculate actual cost of sequence

return sequence
```

A solution to the SARP is then found by performing all three heuristics and selecting the minimum cost sequence produced by any of the heuristics. This solution is an upper bound to the optimal solution, which should prove to be sufficiently consistent to allow comparison between the trivial and coordinated allocation of sectors to the subsets of the agents.

Chapter 5

Intent-Based Coordination

It is assumed that persistent reconnaissance is performed well if the Multi-Agent Reconnaissance Problem (MARP) from Chapter 3 is solved to optimality. As stated in Chapter 4, the optimal MARP solution consists of each agent covering the *optimal set of sectors* in an *optimal sequence*. The intent-based coordination put forward here is based on this concept by attempting to allocate the *optimal set of sectors* during a coordinative process and leaving the *optimal sequence* as the solution to each respective Single-Agent Reconnaissance Problem (SARP). The challenge is to design the coordinative method such that it can deal with a dynamic and uncertain environment.

5-1 Solution structure

Requirements

Based on the analysis in Chapter 2, three requirements are deducted for a suitable solution method:

1. It needs to be **flexible**, meaning that the solution method can adjust the behavior of agents locally without revising the solution for the network as a whole.
2. It needs to be **robust** against limited and unreliable communication, meaning that reduced cooperation does not prohibit the generation of solutions, or impact the solution quality beyond acceptable levels.
3. It needs to be **scalable** in the number of agents and sectors, meaning that large problem instances will not lead to an intractable increase in computation time, maintaining any achieved flexibility and robustness.

Combined, intent-based coordination is designed to allocate sectors as closely as possible to each agent's *optimal set of sectors* while conforming to the three requirements outlined here. In Chapter 4, methods for lower and upper bounds have been presented, providing insight into how well this allocation is performed.

Components

The proposed solution method consists of multiple components. The links between them are displayed in Figure 5.1, to explain their functionality clearly. The baseline problem that needs to be solved is the Multi-Agent Reconnaissance Problem (MARP), shown in red. Instead of solving the MARP, intent-based coordination (light blue) allocates the sectors such that each agent can solve the SARP independently (dark green). Intent-based coordination consists of a distributed hierarchical structure of supervisors. Each supervisor allocates an area to either a subordinate supervisor or to the actual agents. It does so using two sequential processes. First, it clusters the sectors within its own allocated subset (orange). Then, it allocates them to its subordinates (purple). The distributed part of this coordinative process comes forth in the cooperation between peer supervisors (light green). Instead of solving the allocation separately, cooperation is allowed to improve the final allocation. Lastly, the separately obtained solutions for the SARPs are assessed using a benchmarking method (dark blue) to measure the quality compared to the (unknown) optimal solution of the MARP.

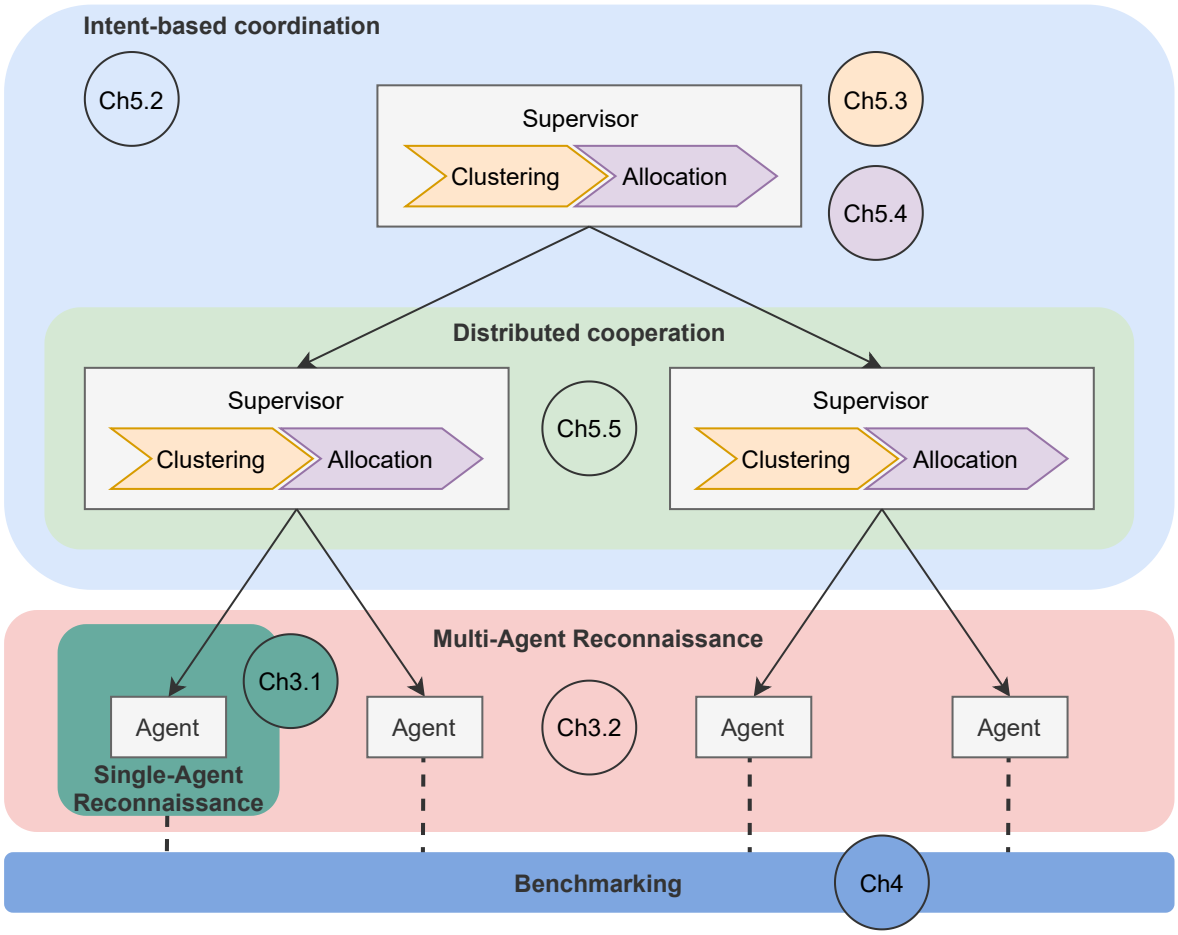


Figure 5.1: A graphical display of the components of the solution method, and the different links between them.

5-2 Application of intent

General concept

As outlined in the overview of the components, the coordination consists of a distributed hierarchical structure of supervisors, which all perform sequential clustering and allocation while cooperating with peer supervisors to improve the allocation. Intent, however, is not a specific component. Instead, it is a principle used in designing the clustering and (distributed) allocation methods.

As the military deals with the same highly dynamic and uncertain environment, the concept of intent is taken from the contemporary intent-based Command & Control (C2), which is developed to deal with these challenges specifically. The requirements for this form of C2 are equal to those described in Section 5-1. Flexibility is needed, as it is not always possible to wait for approval from higher echelons in the chain of command. Furthermore, scalability is mandatory as the process of command has to remain functional for larger combat groups. Lastly, the continuity of operations should be robust against unreliable communication.

Intent-based C2 revolves around the idea that a commander must have the freedom to act on local information at any level in the chain of command without explicit approval from its superior. The vital part of making this type of command work is a clear intent, which implies that commanders can make decisions locally that are still in line with the global objectives. This approach is opposite to rigid, micro-managed command, which resembles the centralized approach to solving the MARP. Therefore, intent outlines the overall objective and constraints but leaves the exact implementation to the subordinates.

This type of C2 can be mimicked by distributed, hierarchical intent-based coordination. Distributed in the sense that at each level peer supervisors cooperate towards a common objective, and hierarchical as a shared superior sets their constraints.

Though a hierarchy suggests a strictly top-down approach, it is important to understand that when information changes, the bottom levels of the hierarchy react first. This makes it a type of holarchy [73]. The intent is refined at higher levels in the hierarchy only for larger and more general changes.

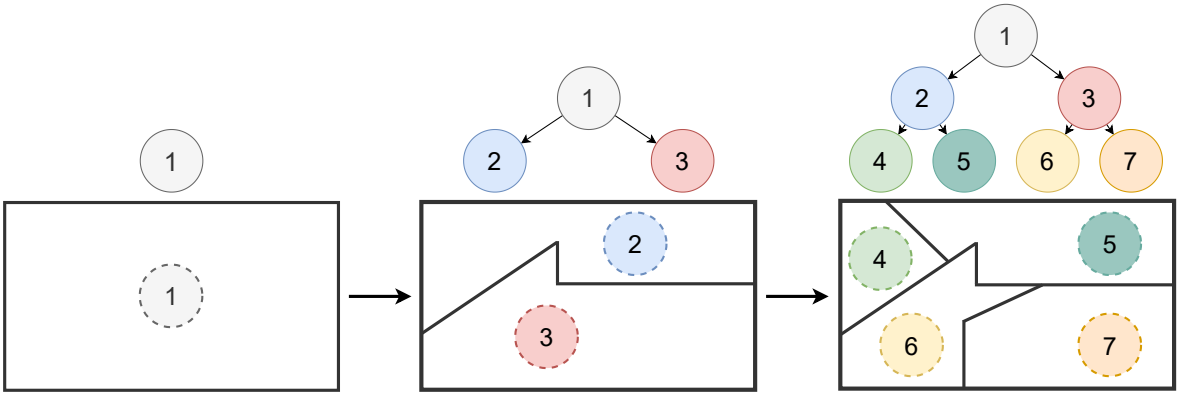


Figure 5.2: An example of strictly hierarchical coordination. Each supervisor partitions the subsets further for its own subordinates.

Quantifying intent

Before intent can be implemented in coordination, the principle needs to be made exact. For persistent reconnaissance, coordination means allocating the *optimal set of sectors* to each agent. This means that the total area to be searched is partitioned into increasingly smaller subsets in the hierarchy of supervisors. The objective of this partitioning step is equal for all supervisors, namely to allocate the right subsets to their subordinates. Based on this objective, an example of strictly hierarchical coordination is displayed in Figure 5.2.

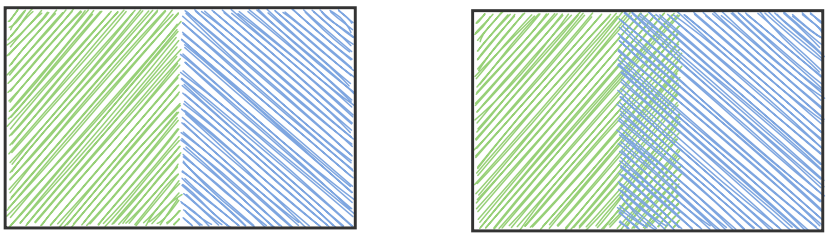


Figure 5.3: By letting the subsets overlap, room for cooperation is added for peer supervisors. The specific size and number of overlapping subsets represent the superior's intent.

As there is no variation in the objective, the intent is represented by the partitioning constraints. In this case, local room for refinement means that the subspace allocated to each supervisor is not final but fixed only to a certain degree. Through distributed cooperation, supervisors can then adjust their allocations. An example is displayed in Figure 5.3. The left case depicts a strictly hierarchical subset allocation, corresponding with Figure 5.2. The right case implements the definition of intent, such that the subsets overlap to indicate room for freedom. Effectively, a superior indicates within which bounds any subsequent partitioning can take place but leaves the exact implementation to its subordinates. The overlapping region then represents the area in which peer supervisors can cooperate distributedly. Intent is thus quantified by how much overlap is generated between the subsets of subordinates.

5-3 Clustering sectors

The clustering objective is to group sectors for more efficient allocation, with a minimal impact on the resulting solution quality. Furthermore, as clustering measures the resemblance between sectors, it can be used to implement intent. This implies that clustering should:

- Include a measure of partial overlap, such that intent can be modelled.
- Anticipate the capabilities of the available heterogeneous agents.
- Return connected polygonal clusters, allowing for efficient coverage.

The first objective can be realized by utilizing existing properties of the Fuzzy C-Means (FCM) clustering algorithm [4]. The second and third objectives are new contributions. In the following sections, the sector features will be outlined used to accommodate the heterogeneity, and the algorithm will be described. This consists of the implementation of FCM, and the new post-processing method to obtain more efficient connected clusters.

5-3-1 Sector features

The clustering process aims at making all sectors within each cluster as similar as possible while making sectors from different clusters a dissimilar as possible. The (dis)similarity is measured between two weighed feature vectors, in which the feature vector consists of values for each of the sector's features. For all sectors $i \in \mathcal{N}$ these can be summarized as:

- **Position**, given as $\{y_i, x_i\}$ coordinates.
- **Risk**, given as $r_i \in [0, P_i]$.
- **Risk increase**, given as $\dot{r}_i \in [0, \frac{1}{2}P_i]$.

For comparison, the features are normalized and collected in the feature vector:

$$X_i = \left[\frac{y_i}{\max_{i\in\mathcal{N}}(y_i, x_i)}, \frac{x_i}{\max_{i\in\mathcal{N}}(y_i, x_i)}, \frac{r_i}{\max_{i\in\mathcal{N}}(P_i)}, \frac{2\cdot\dot{r}_i}{\max_{i\in\mathcal{N}}(P_i)}\right] \tag{5.1}$$

The (dis)similarity is defined as the Euclidean distance between two weighed feature vectors, in which the weights w can be used to create clusters with a different emphasis:

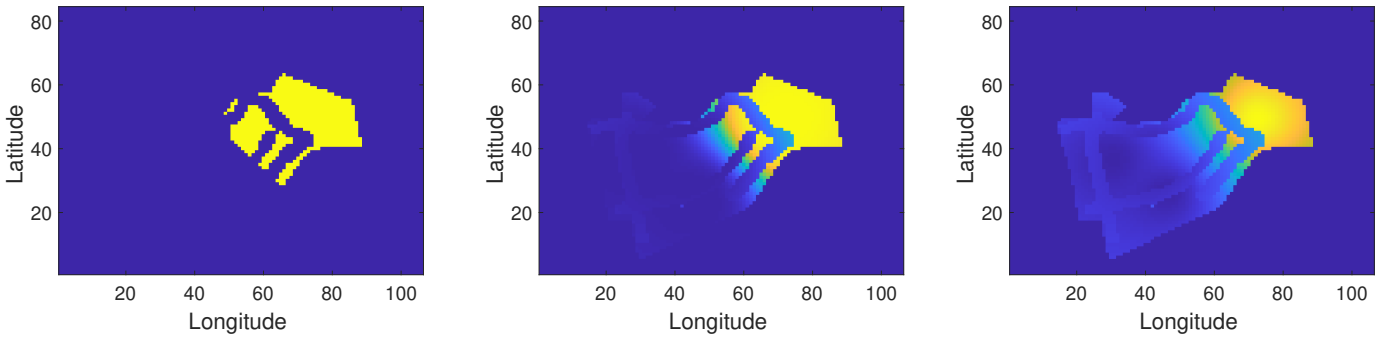
$$d(X_i, X_j) = ||w \cdot (X_i - X_j)||_2 \qquad \forall\ i, j \in \mathcal{N} \tag{5.2}$$

The weights w should enable capability-based clustering for heterogeneous agents. This implies that clusters can be created that are tailored to specific agents during allocation. The central assumption is that agent capabilities correspond with sector features, such that if agents differ more in certain capabilities, that feature must be weighed more heavily. Therefore, a direct link must be formulated between capabilities and features to allow capability-based clustering. The relations follow from the MARP (see Section 3-2), in which the agent capabilities are tied to the objective value. For the mentioned capabilities this means:

- **Speed**: The higher the speed v_k, the better agent $k \in \mathcal{P}$ can cover high risk sectors at longer distances. Therefore, the larger the speed difference between the available systems, the more emphasis is need on risk r_i $\forall i \in \mathcal{N}$.
- **Flight Time:** The higher the flight time e_k, the better the agent can cover high-risk increase over longer cycle distances. For larger differences in flight time, an emphasis is needed on the risk increase $\dot{r}_i$ $\forall i \in \mathcal{N}$.

5-3-2 Fuzzy C-Means algorithm

The FCM algorithm [4] is selected to perform the initial clustering for three reasons. First, it is a relatively straightforward clustering method comparable to k-means. This makes the results understandable and consistent. Secondly, in contrast to hard clustering, fuzzy clustering returns a continuous degree representing the certainty of a sector belonging to a cluster, which can be used directly to describe intent. Thirdly, the fuzzy degree shows more gradual growth or decay in degree if features change. This means that instead of sectors



(a) $m = 1.0$ **(b)** $m = 1.5$ **(c)** $m = 2.0$

Figure 5.4: A cluster created with FCM with different values for the fuzzifier m. With $m = 1$, the fuzzy clusters converge to hard clusters. For higher values of m, more sectors are included in the clusters with a small degree.

'flipping' to other clusters, the degree changes slowly. Lastly, c-means has a parameter called the fuzzifier $m \geq 1$, which is used to set the amount of fuzziness easily. This effect is shown in Figure 5.4. This value can be tuned to generate different amounts of cluster overlap, such that the amount of cooperation between supervisors is changed accordingly.

FCM can formally be described as an optimization problem. Given a set of clusters $\mathcal{M}$ with centroid C_j $\forall j \in \mathcal{M}$, and a set of sectors $\mathcal{N}$ with a feature vector X_i $\forall i \in \mathcal{N}$ the optimization problem is formulated as follows:

$$\min_{C_j, u_{ij}} \quad \sum_{i \in \mathcal{N}} \sum_{j \in \mathcal{M}} u_{ij}^m ||X_i - c_j||^2 \tag{5.3}$$

$$\text{s.t.} \quad \sum_{j \in \mathcal{M}} u_{ij} = 1 \qquad \forall i \in \mathcal{N} \tag{5.4}$$

$$C_j = \frac{\sum_{i \in \mathcal{N}} u_{ij}^m \cdot X_i}{\sum_{i \in \mathcal{N}} u_{ij}^m} \qquad \forall j \in \mathcal{M} \tag{5.5}$$

$$u_{ij} = \frac{1}{\sum_{k \in \mathcal{M}} \left(\frac{||X_i - C_j||}{||X_i - C_k||} \right)^{\frac{2}{m-1}}} \qquad \forall i \in \mathcal{N},\ \forall j \in \mathcal{M} \tag{5.6}$$

Objective function (5.3) minimizes the Euclidean distance to each cluster, weighed by the degree u_{ij}, which describes the membership of sector $i \in \mathcal{N}$ to cluster $j \in \mathcal{M}$. Constraint (5.4) dictates that the total degree of a sector is always 100%. Constraint (5.5) calculates the centroid for each cluster based on the degree of each sector. Constraint (5.6) sets the degree based on the Euclidean distances.

A solution is found iteratively by setting the fuzzifier m and defining a fixed amount of clusters $\mathcal{M}$. Then, the algorithm is initialized with initial centroids C_j. Iteratively, Constraints (5.6) and (5.5) are calculated until the improvement of the objective function (5.3) falls below a given threshold ϵ.

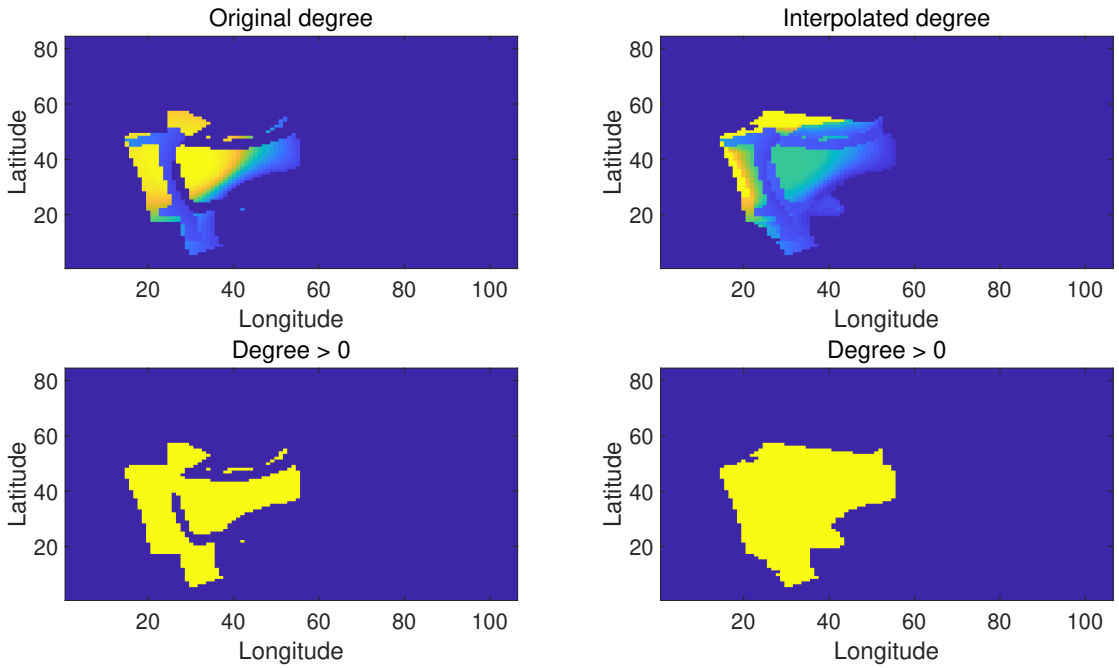


Figure 5.5: An example of the interpolation method to create polygonal clusters without losing low degrees, by capping the interpolated values at the n$^{\text{th}}$ percentile of the original degree.

5-3-3 Reducing cluster sparseness

FCM tends to make the clusters sparse, meaning that a cluster does not necessarily consist of connected sectors but physically scattered sectors. This effect becomes greater when the fuzzifier is set lower, as is also visible in Figure 5.4. As sparse clusters are assumed inefficient in practice, a post-processing method is proposed in which the initial clusters are interconnected to create connected, polygonal clusters. The approach consists of multiple steps:

1. Using FCM, calculate the degree u_{ij} of sector $i \in \mathcal{N}$ belonging to cluster $j \in \mathcal{M}$.
2. Retrieve the sectors with significant degrees, such that $u_{ij} \geq \beta$ (i.e. $\beta = 3\%$).
3. Each sector with $u_{ij} \geq \beta$ and all sectors that are on the edges of the convex hull are collected as $i' \in \mathcal{N}$. The remaining sectors are identified as $\hat{i} \in \mathcal{N}$.
4. The included degree $u_{i'j}$ is interpolated for all sectors to create $\hat{u}_{ij}$, where $\hat{u}_{i'j} = u_{i'j}$. Then, $\hat{u}_{\hat{i}j}$ is the interpolated degree for the remaining sectors.
5. Get the degree γ at n$^{\text{th}}$ percentile of u_{ij} (i.e. $n = 20\%$).
6. Limit the interpolated degree $\hat{u}_{\hat{i}j}$ to γ, such that $\hat{u}_{\hat{i}j} = \min\{\hat{u}_{\hat{i}j}, \gamma\}$. By doing this, 'valleys' of low degree are not interpolated to the high degree of its surrounding.
7. The degrees are normalized by $\hat{u}_{ij} = \hat{u}_{ij} / \sum_j \hat{u}_{ij}$, such that $\sum_j \hat{u}_{ij} = 1 \ \forall i \in \mathcal{N}$.

The result of these steps can be seen in Figure 5.5, where it is shown that the values are interpolated within the polygonal surrounding the sectors with a significant degree originally, but not so much that its low degree information is lost. Caution must be taken, as this effect changes with varying fuzziness.

5-4 Task Allocation

The objective of task allocation is to allocate the clusters such that each agent receives its *optimal set of sectors* as best as possible, as formulated in Section 3-2. Of course, this result is limited by the clustering quality. During task allocation, the objective of the MARP is not taken into account explicitly. Instead, the task allocation is a heuristical approach by valuing clusters for each agent and including capacity. The main contribution consists of implicitly linking the properties of the obtained clusters to the MARP objective and creating a corresponding mathematical formulation accordingly. The two main objectives taken into account are defined as:

1. Maximize the utility of agents taking up specific tasks, such that agents get allocated sectors in which they can eliminate risk more efficiently than others.
2. Minimize inefficiencies resulting from the allocation of more tasks than the capacity. Though some agents might be better at performing certain tasks, allocating too many tasks will eventually decrease performance.

5-4-1 Valuing task properties

Task properties

For each cluster $j \in \mathcal{M}$ representing a task, some generic properties can be calculated for comparison, with $\hat{u}_{ij}$ indicating the interpolated fuzzy degree of sector $i \in \mathcal{N}$ belonging to cluster $j \in \mathcal{M}$, and A^s the standard sector area:

$$\text{Centre of mass:} \quad F_j = \left\{ \bar{y}_j = \frac{\sum_{i \in \mathcal{N}} (\hat{u}_{ij} \cdot y_i)}{\sum_{i \in \mathcal{N}} \hat{u}_{ij}},\ \bar{x}_j = \frac{\sum_{i \in \mathcal{N}} (\hat{u}_{ij} \cdot x_i)}{\sum_{i \in \mathcal{N}} \hat{u}_{ij}} \right\} \tag{5.7}$$

$$\text{Total cluster area:} \quad A_j = \sum_{i \in \mathcal{N}} \hat{u}_{ij} \cdot A^s \tag{5.8}$$

$$\text{Total risk:} \quad R_j = \sum_{i \in \mathcal{N}} \hat{u}_{ij} \cdot r_i \tag{5.9}$$

$$\text{Average risk increase:} \quad \dot{R}_j = \frac{\sum_{i \in \mathcal{N}} (\hat{u}_{ij} \cdot \dot{r}_i)}{\sum_{i \in \mathcal{N}} \hat{u}_{ij}} \tag{5.10}$$

Task utility

The available capabilities for each agent are speed and flight time. The task utility for a specific agent is defined as a linear score for the suitability of the agents' capabilities matched

with the corresponding task properties. This means that long flight time agents score well on high-risk-increase tasks, and that fast agents score well on high-risk tasks. l^1_{jk} indicates the gain in quick elimination of risk by assigning agent $k \in \mathcal{P}$ to task $j \in \mathcal{M}$. The score l^2_{jk} indicates the gain of putting long flight time agents on clusters with high risk increase. The scores are then calculated as follows:

$$l^1_{jk} = \frac{v_k \cdot R_j}{\sum_{j \in \mathcal{M}} \max_{k \in \mathcal{P}} (v_k \cdot R_j)} \qquad \forall j \in \mathcal{M},\ k \in \mathcal{P} \tag{5.11}$$

$$l^2_{jk} = \frac{e_k \cdot \dot{R}_j}{\sum_{j \in \mathcal{M}} \max_{k \in \mathcal{P}} (e_k \cdot \dot{R}_j)} \qquad \forall j \in \mathcal{M},\ k \in \mathcal{P} \tag{5.12}$$

Using these scores, the utility a_{jk} of agent $k \in \mathcal{P}$ taking task $j \in \mathcal{M}$ is calculated as:

$$a_{jk} = \frac{1}{2}(l^1_{jk} + l^2_{jk}) \qquad \forall j \in \mathcal{M},\ k \in \mathcal{P} \tag{5.13}$$

Given this derivation, the total attainable utility is bounded by the range $[0, 1]$, and equals 1 if the maximum scoring agent is selected for each task. However, this is only the case if the same agent achieves the maximum score for both measures. If the measures conflict, allocating the maximum score of both measures is not possible at the same time.

Capacity

Capacity $\hat{b}_k \in [0, 1]$ is defined as a fraction of the total area an agent $k \in \mathcal{P}$ can service effectively. The task area $A_i\ \forall i \in \mathcal{M}$ is implemented as a fraction of the total area too, such that $A^f_i = A_i / \sum_{i \in \mathcal{M}} A_i$. It is important to note that the capacity is a relative measure dependent on the area size, type, and the number of agents. Furthermore, the metric task dependency D^f_{ij} is introduced, which indicates an approximated area that needs to be covered by combining tasks, as displayed in Figure 5.6. The normalized dependency between task i and j is given as:

$$D^f_{ij} = \frac{A_i A_j ||F_i - F_j||_2^2}{\sum_{i,j \in \mathcal{M}} A_i A_j ||F_i - F_j||_2^2} \qquad \forall i, j \in \mathcal{M} \tag{5.14}$$

A proof of the necessity of weighing by area size is included in Appendix C. Given $D^f_k \in [0, 1]$ and $A^f_k \in [0, 1]$ represent the sum of all included dependencies and task areas respectively, the capacity can be constrained for each agent $k \in \mathcal{P}$ using the following formula:

$$\frac{1}{2} D^f_k + \frac{1}{2} A^f_k \leq \hat{b}_k \tag{5.15}$$

5-4-2 Local formulation

The Mixed Integer Linear Program (MILP) formulation proposed here applies to the problem of a single supervisor allocating tasks to its subordinates. Effectively this makes it the local, non-cooperative problem of each supervisor.

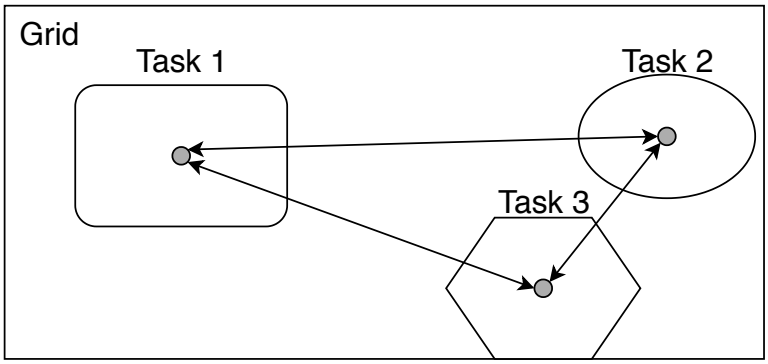


Figure 5.6: A schematic view of dependencies between tasks. If multiple tasks are selected, the distance between them induces inefficiencies and a strain on the capacity.

Let x_{ik} be a binary variable indicating if task $i \in \mathcal{M}$ is performed by subordinate $k \in \mathcal{P}$ (either supervisors or agents). Then the constant a_{ik} indicates the utility of subordinate k performing task i. The matrix D_{ij}^f gives a normalized, weighted squared distance (dependency) between the centers of mass from task i to task j. The constant $\hat{b}_k$ indicates the normalized capacity, meaning the fraction of the area the agent can service effectively. The constant A_i^f indicates the fraction of the task area compared to the grid area. Then, the variable β_k indicates the overused capacity for each subordinate, with e the inefficiency penalty. The formulation then follows as:

$$\max_{x_{ik},\beta_k} \quad \sum_{k\in\mathcal{P}} \left(\sum_{i\in\mathcal{M}} a_{ik}x_{ik} - e \cdot \beta_k \right) \tag{5.16}$$

$$\text{s.t.} \quad \frac{1}{2} \sum_{i\in\mathcal{M}} \left(\sum_{j\in\mathcal{M}} x_{ik} D_{ij}^f x_{jk} + A_i^f x_{ik} \right) - \beta_k \leq \hat{b}_k \qquad \forall k \in \mathcal{P} \tag{5.17}$$

$$\sum_{k\in\mathcal{P}} x_{ik} = 1 \qquad \forall i \in \mathcal{M} \tag{5.18}$$

$$x_{ik} \in \{0,1\} \qquad \forall i \in \mathcal{M},\ \forall k \in \mathcal{P} \tag{5.19}$$

$$\beta_k \geq 0 \qquad \forall k \in \mathcal{P} \tag{5.20}$$

Objective function (5.16) maximizes the utility of tasks allocated, by selecting the appropriate subordinate and preventing penalties for overused subordinate capacity. Constraint (5.17) sets the approximated fraction of the area that has to be serviced by a subordinate and relates it to its capacity. If all tasks are serviced by one subordinate, the first two terms add up to one. Constraint (5.18) ensures that all tasks are performed by exactly one subordinate. Constraint (5.19) and (5.20) set the domain for the decision variables.

After the allocation is performed, the allocated subsets are represented by the resulting degree u_{ik}, stating the total degree with which sector $i \in \mathcal{N}$ belongs to agent $k \in \mathcal{P}$. This is calculated using the interpolated degree $\hat{u}_{ij}$ of each cluster:

$$u_{ik} = \sum_{j\in\mathcal{M}} \hat{u}_{ij} x_{jk} \qquad \forall i \in \mathcal{N},\ k \in \mathcal{P} \tag{5.21}$$

The resulting problem can be classified as a Mixed Integer Quadratically Constrained Program (MIQCP), which is a very difficult problem, as both the non-convex integer variables and non-convex quadratic constraints are combined [92, 93]. If linearly relaxed, the problem reduces to a Quadratically Constrained Linear Program (QCLP), which can be solved efficiently by interior-point methods using Semidefinite Programming (SDP), if matrix D^f is positive semidefinite. By definition $D^f \geq 0$ and symmetrical, such that $x^T D^f x \geq 0$ does not hold for any nonzero vector x. This proves it is not positive semidefinite. However, in the case of binary decision variables, this can be enforced relatively easily using that diagonal dominance $D_{ii} \geq \sum_{i \neq j} D_{ij} \ \forall i \in N$ is a sufficient condition for positive semidefiniteness. Hence, as $x_i^2 = x_i$ holds for binary variables, D^f can be convexified as follows:

$$x^T D^f x = x^T \hat{D}^f x - \hat{d} x \tag{5.22}$$

Then, the values of $\hat{D}$ and $\hat{d}$ are chosen to enforce diagonal dominance:

$$\hat{D}_{ij}^f = D_{ij}^f \qquad \forall i \neq j \in N \tag{5.23}$$
$$\hat{D}_{ii}^f = \sum_{j \in N} D_{ij}^f \qquad \forall i \in N \tag{5.24}$$
$$\hat{d}_i^f = \sum_{j \in N} D_{ij}^f \qquad \forall i \in N \tag{5.25}$$

It should be noted that for $\hat{d}$ any value can be chosen that makes D^f positive definite, even if not necessarily diagonally dominant. Even more so, the selection of $\hat{d}$ is not trivial and has a significant impact on performance [94]. A much more efficient reformulation is, for example, achieved by applying the Quadratic Convex Reformulation (QCR) [95]. However, these more advanced methods are out of scope for this thesis, and preference is given for the more straight forward implementation of linearization.

The quadratic constraint can be eliminated by linearizing, using – as Mallach [96] noted – 'the standard linearization technique', which was gradually developed by Fortet [97], Hammer and Rudeanu [98], and Glover and Woolsey [99,100]. This linearization replaces the quadratic binary variable by a combined, continuous variable, using the following reformulation:

$$\sum_{i \in \mathcal{N}} \left(\sum_{j \in \mathcal{N}} D_{ij} x_{ijk} + A_j x_{ik} \right) - \beta_k = \hat{b}_k \qquad \forall k \in \mathcal{P} \tag{5.26}$$
$$x_{ijk} \leq x_{ik} \qquad \forall i, j \in \mathcal{N},\ \forall k \in \mathcal{P} \tag{5.27}$$
$$x_{jik} \leq x_{ik} \qquad \forall i, j \in \mathcal{N},\ \forall k \in \mathcal{P} \tag{5.28}$$
$$x_{ik} + x_{jk} - 1 \leq x_{ijk} \qquad \forall i, j \in \mathcal{N},\ \forall k \in \mathcal{P} \tag{5.29}$$
$$x_{ijk} \geq 0 \qquad \forall i, j \in \mathcal{N},\ \forall k \in \mathcal{P} \tag{5.30}$$

These constraints can replace constraint (5.17), as the equality $x_{ijk} = x_{ik} \cdot x_{jk}$ is implicitly enforced, though this linearization does not necessarily improve performance. More compact and novel linearizations have been developed, reducing the amount of constraints and variables, tightening the linear relaxation of the problem [96, 101, 102].

5-5 Distributed cooperation

Though each supervisor can partition its subset locally for its subordinates, the addition of overlapping subsets (intent) requires cooperation. Cooperation enables flexibility, as solutions can be altered within the freedom provided by a shared supervisor without its explicit approval. The approach is distributed to provide robust anytime properties, such that instead of having a single bottleneck, the solution improves gradually over time. Therefore, it is not the explicit goal to create a more efficient method compared to the centralized approach.

The main contribution for distributed cooperation is the integration of a complex local problem into Concurrent Forward-Bounding (ConcFB) [5], creating the new Complex Concurrent Bounding (CCB) algorithm. This involves compiling the local problem and sharing these over all concurrent Search Processes (SPs), removing the concept of Upper Bound (UB) requests, and adding a local search tree.

5-5-1 Concept of shared tasks

Before any cooperation can be performed, it needs to be defined on which tasks can be cooperated. This directly follows from the intent of the shared superior of a group of peer supervisors. An example is given in Figure 5.7. The left picture displays how two supervisors (red and blue) are allocated overlapping subsets. Both supervisors perform clustering for their subset. The consequence is that some clusters are partially located in the overlapping sections with the other supervisor's subset. This is displayed in the center image. The red supervisor has created the yellow and orange clusters, and the blue supervisor the green and turquoise clusters. Both the red and turquoise clusters are partially located in the subset of the other supervisor. This means they both belong to the shared tasks. The right image displays that both shared tasks are allocated to their respective peer supervisor after cooperation, which are now in their final, deconflicted subset. An important observation here is that the supervisors now have a subset *outside* the original overlapping parts.

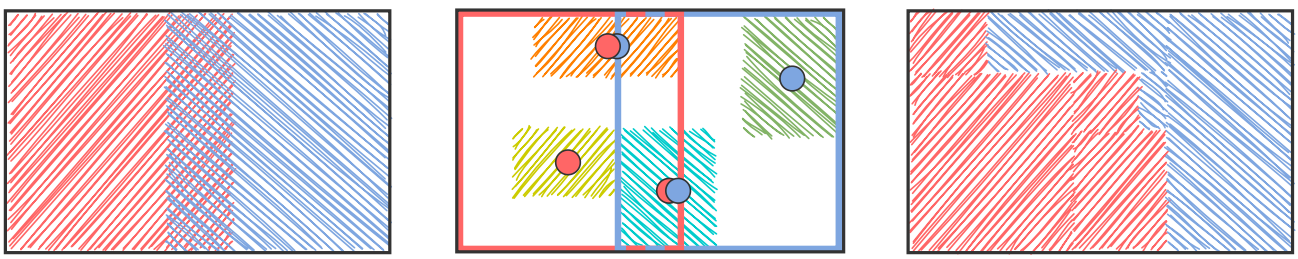


Figure 5.7: Left: two supervisors are allocated to overlapping subsets. Center: The yellow and green tasks are local, as they belong to only one subset. The orange and turquoise clusters are located partially in the overlapping subsets, meaning they are shared tasks. Right: The tasks are allocated to their respective peer supervisor after cooperation.

5-5-2 Cooperative formulation

The cooperative problem can be formalized by extending the local formulation from Section 5-4-2 to include the local and shared tasks. The problem is described for a group of $q \in \mathcal{C}$ peer supervisors, sharing the same superior q^0. The subordinates of each supervisor are defined as $k \in \mathcal{P}_q \ \forall q \in \mathcal{C}$, with $\mathcal{P} = \bigcup_{q \in \mathcal{C}} \mathcal{P}_q$. Furthermore, each supervisor is allocated a subset prior by the superior, described by the degree u_{iq} of sector $i \in \mathcal{N}$ belonging to supervisor $q \in \mathcal{C}$. Note that $\mathcal{C} = \mathcal{P}_{q^0}$, as the set of subordinates for the superior is simply the current group of supervisors. This also implies that $u_{iq} = u_{ik} \ \forall k \in \mathcal{P}_{q^0} = q \in \mathcal{C}$. Each supervisor clusters locally, generating a set $\mathcal{M}_q$ of the clusters it created. Combined, the set of all clusters is $\mathcal{M} = \bigcup_{q \in \mathcal{C}} \mathcal{M}_q$. Furthermore, each supervisor discounts the interpolated degrees $\hat{u}_{ij} \ \forall j \in \mathcal{M}_q$, using the subset u_{iq} it has been allocated prior:

$$\tilde{u}_{ij} = \hat{u}_{ij} \cdot u_{iq} \qquad \forall i \in \mathcal{N},\ j \in \mathcal{M}_q \tag{5.31}$$

The task properties formulated using Equation (5.7) - (5.10) are then changed to use the discounted degree instead of the interpolated degree directly. This ensures that throughout each level of the hierarchy, the total degree of each sector remains equal to 1.

Then, the formulation for a group of supervisors $\mathcal{C}$, which cooperatively need to allocate $\mathcal{M}$ tasks to $\mathcal{P}$ subordinates, is given as:

$$\max_{x_{ik}, \beta_k} \quad \sum_{q \in \mathcal{C}} \sum_{k \in \mathcal{P}_q} \left(\sum_{i \in \mathcal{M}} a_{ik} x_{ik} - e \cdot \beta_k \right) \tag{5.32}$$

$$\text{s.t.} \quad \text{Eq. (5.17)} \qquad \forall k \in \mathcal{P}_q,\ q \in \mathcal{C} \tag{5.33}$$

$$\sum_{q \in \mathcal{L}_i} \sum_{k \in \mathcal{P}_q} x_{ik} = 1 \qquad \forall i \in \mathcal{M} \tag{5.34}$$

$$\text{Eq. (5.19), (5.20)} \qquad \forall i \in \mathcal{M},\ k \in \mathcal{P}_q,\ q \in \mathcal{C} \tag{5.35}$$

Constraint (5.34) can then be split into two constraints to separate the local and shared tasks. To do so, for each task $j \in \mathcal{M}$ a set $\mathcal{L}_j$ is defined, which includes all supervisors $q \in \mathcal{C}$ that are allowed to perform the task:

$$\mathcal{L}_j = \{q \in \mathcal{C} \ : \ \sum_{i \in \mathcal{N}} \tilde{u}_{ij} \cdot u_{iq} > 0\} \qquad \forall j \in \mathcal{M} \tag{5.36}$$

Effectively, this states that a supervisor has access to a cluster if any sector in the cluster also has a positive degree in the subset of this supervisor. Note that the supervisor who has created a cluster always has access, implying that $\exists j \in \mathcal{M}_q \Rightarrow \exists q \in \mathcal{L}_j$. However, potentially also others can have access, as displayed in 5.7. Combined, the local task set consists of all tasks that are accessible *only* by the supervisor that created it:

$$\mathcal{M}_q^l = \{j \in \mathcal{M}_q \ : \ |\mathcal{L}_j| = 1\} \qquad q \in \mathcal{C} \tag{5.37}$$

Subsequently, all tasks that belong to multiple supervisors are collected in the shared task set $\mathcal{M}^s$. The reformulation of Constraint (5.34) is then given as:

$$\sum_{k \in \mathcal{P}_q} x_{ik} = 1 \qquad \forall i \in \mathcal{M}_q^l,\ q \in \mathcal{C} \tag{5.38}$$

$$\sum_{q \in \mathcal{L}_i} \sum_{k \in \mathcal{P}_q} x_{ik} = 1 \qquad \forall i \in \mathcal{M}^s \tag{5.39}$$

Using this formulation, the difference is made visible between the separate local tasks of each supervisor $q \in \mathcal{C}$, and the set of shared tasks $\mathcal{M}^s$. Though the sets are separated, they are not independent. Nonetheless, during cooperation only the assignments of the tasks in the shared set $\mathcal{M}^s$ have to be communicated, as the tasks in the local sets can be solved by each supervisor locally.

5-5-3 Distributed Constraint Optimization Problems

The cooperative formulation can be solved using the Distributed Constraint Optimization Problem (DCOP) paradigm. This section will explain the general concept, with special attention to why this formulation can be classified as one with a complex local problem.

DCOPs originate from Distributed Artificial Intelligence (DAI) and describe a problem in which agents must distributedly cooperate to decide on a local variable while subjected to a set of constraints, such that the global cost is minimized. Local variables often represent physical behavior, more commonly known as a Multi-Robot System (MRS). Fioretto et al. [103] provide a detailed survey on the use of DCOPs for Multi-Agent systems (MASs). In line with their definition, a DCOP is defined as a tuple $S = \langle P, X, D, F, \alpha \rangle$, with P the set of agents, X the set of variables, D the set of domains for X, F the set of cost functions where each function $f \in F$ is coupled with a variable $x \in X$, and α is a function that assigns control of variables to agents, such that $\alpha : X \mapsto A$. The global cost function is then defined as $F_g(X) = \sum_{f \in F} f(x)$. The goal of the DCOP algorithm is then to find a solution that minimizes the global cost.

A full classification of the properties of DCOP algorithms is also provided by Fioretto et al. [103]. The desired properties can be summarized as asynchronous, any-time, fully distributed, and exact. It needs to be exact as the complete solution approach of intent-based coordination consists of many components. As such, introducing potentially unreliable results by an inexact method is undesirable, at least for the scope of this thesis. The other properties all increase the robustness against communication failures. Apart from these properties, it is also generally assumed that each agent controls only one variable. In this application, this is not true, which requires some adjustments for existing algorithms.

Towards a complex problem

In the cooperative formulation, each DCOP agent represents a supervisor, which controls the full assignment of all its local tasks to its subordinates and can cooperate on the assignment of multiple shared tasks. Effectively, each agent still needs to generate an optimal solution given the assignment of shared variables. Yokoo [104] defined this as a complex local problem, and Burke and Brown [105] summarize two general methods for dealing with this complexity.

The first solution is *compilation*, which produces a new variable for each agent, for which the domain is the set of all local solutions. The second solution is *decomposition*, in which a unique artificial agent manages each local variable.

With large complex problems, decomposition would lead to highly constrained, dense problems, which generally reduces parallelism and efficiency [104], as additional computational and communication overhead is introduced. Compilation, on the other hand, can become intractable quickly. In general, it cannot be expected that enumerating the solutions to any NP-hard problem is viable. Recognizing this, Burke et al. [106] proposed an improved compilation method. They state that for complex local problems, only some shared variables are constrained by other agents. This means that all local solutions with equal assignment for the shared variables are fully interchangeable. Even using this form of compilation, the domain of each agent can still become very large quickly. In fact, the size of the compiled domain D_q of any supervisor $q \in \mathcal{C}$ grows exponentially on the total number of tasks the supervisor can perform from the shared set:

$$|D_q| = 2^{(\sum_{i \in \mathcal{M}^s} [\exists q \in \mathcal{L}_i])} \qquad \forall q \in \mathcal{C} \tag{5.40}$$

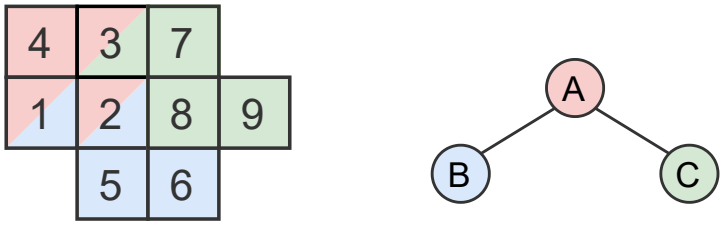
Nonetheless, compilation of the variable domain is chosen, as Burke and Brown [105] show that solving a local problem centrally enables more efficient exploitation of the problem structure, especially for larger local problems. However, they emphasize a need for techniques that efficiently handle large domain sizes to counter the inevitable increase in domain size.

Literature also provides some case studies for complex local problems, summarized also by [103]. Davin and Modi [107] apply basic compilation to ADOPT, which is not any-time. Khanna et al. [108] propose an algorithm specifically for dynamic problems. Portway and Durfee [109] develop adjustments for inexact algorithms, and Grinshpoun [110] implements clustering of variables in pseudo-trees during decomposition to lower communication overhead. Fioretto et al. [111] propose a more general framework to decompose a multi-variable DCOP in a global and multiple local problems, for which existing DCOP algorithms can be used. Their description resembles the improved compilation by Burke et al. [106], as *external* and *interface* variables both describe the shared tasks. From these works, it can be concluded that the improvement compilation method is a promising approach.

Defining the compiled domain

The compiled variable of an agent consists of all unique combinations of shared tasks it can assign. To illustrate this, consider the possible allocations of the tasks displayed in Figure 5.8 and Table 5.1, where three agents have (partial) access to 9 tasks. In this example, task 1 to 3 belong to the shared set, and task 4 to 9 belong to different local sets. As each task can be selected by a Boolean, the domain for agent A is given as $2^3 = 8$ compiled assignments, representing all unique combinations. For agent B this is $2^2 = 4$, and $2^1 = 2$ for agent C. However, when an agent decides on a possible assignment, the domain might not be completely available. The actual number of possible assignments depends on decisions already made by other agents, as constraint (5.34) dictates that all tasks are satisfied by one agent only.

The actual available domain, minus the unavailable assignments, is therefore dependent on the ordering of the agents. In Figure 5.9, an example is shown. In the left image, the ordering



(a) Tasks **(b)** Connectivity

Figure 5.8: In the left plot, figurative tasks are displayed as a grid. Three colored agents each have local and overlapping shared tasks. At the right, a connectivity graph is displayed indicating which agents have at least one overlapping task.

Table 5.1: A figurative example of the access of three agents to ten tasks.

Tasks	Shared			Local					
Agents	1	2	3	4	5	6	7	8	9
A	✘	✘	✘	✘					
B	✘	✘			✘	✘			
C			✘				✘	✘	✘

$A \Rightarrow B \Rightarrow C$ is displayed. Agent A can make eight combinations. After that, agents 2 and 3 have nothing to cooperate about. Depending on the chosen assignment of A, they are forced to assign the remaining tasks. In the right image, the ordering $C \Rightarrow B \Rightarrow A$ is displayed. After agent C decides whether to assign task 3, agent B still has four unique combinations to make, independent of the chosen assignment. The last agent always has a domain of size one independent of the ordering, as it has to allocate all remaining shared tasks. It can be expected that the chosen ordering affects the runtime, for which two options are outlined based on the connectivity graph, which is displayed in Figure 5.8. The left ordering from Figure 5.9 corresponds to ordering the agents on descending connectivity, meaning the first agent has a shared task with the most other agents. The right case corresponds to descending connectivity. The precise effect and best choice are yet to be determined.

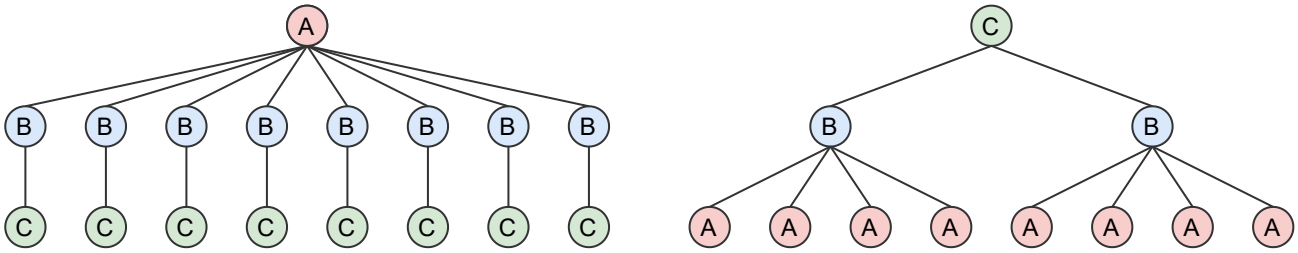


Figure 5.9: Example of the different domain sizes depending on the ordering of the agents.

5-5-4 Concurrent Forward Bounding

From the discussed algorithms by Fioretto et al. [103], ConcFB [5] is a suitable candidate as it has the desired properties, and shows improved performance over, for example, AFB [112]

and BnB-adopt [113]. Furthermore, it shows high concurrency and active workload balancing, which is useful when the computational effort is either costly or slow. Therefore, this algorithm is chosen as starting point to solve the cooperative formulation described in Section 5-5-2.

To explain the algorithm, consider a DCOP with four agents $\{a_1, \ldots, a_4\}$, where a_1 has a domain of four values, and all others a domain of two values. A complete ordering of the agents is made, after which a search tree can be constructed, as depicted in Figure 5.10. First, agent a_1 assigns a value and saves it as the Current Partial Assignment (CPA), then a_2 adds its assignment, and so forth. The values at the bottom of the tree represent solution values, and the red line is an example assignment.

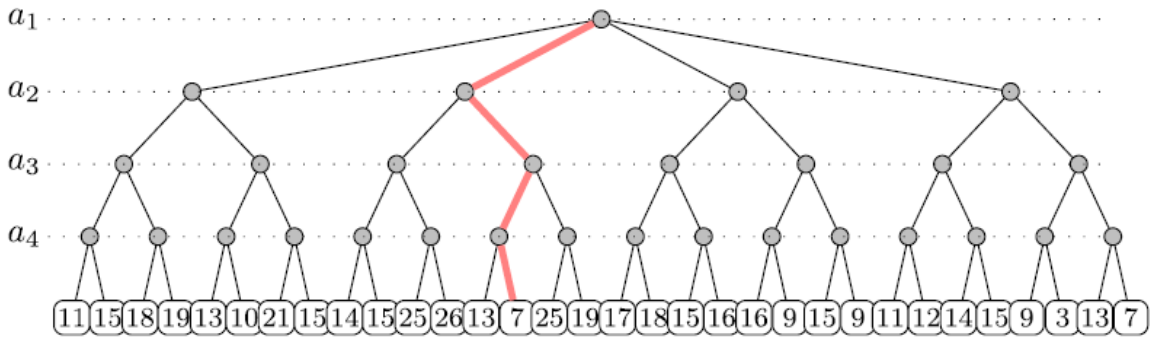


Figure 5.10: A DCOP decision tree, with objective values (bottom) and a solution (red line) [5].

By extending the CPA, a solution is obtained by a_4, after which a new CPA can be created by a_1. To speed up this slow and exhaustive search, Synchronous Forward Bounding (SFB) can be used. This process ensures that each time an agent receives the CPA, it requests an upper bound from all subsequent agents in the ordering and uses these results to compare the current branch to the best-known solution. If insufficient, the CPA is passed back to the preceding agent, which can assign another variable, or pass it back to its parent too. SFB is a synchronous process, as it only continues upon receiving all the requested upper bounds.

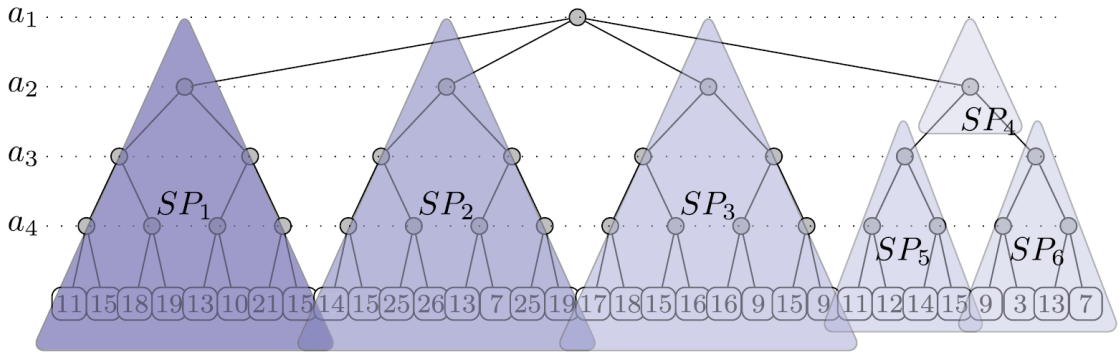


Figure 5.11: Example of disjoint SPs in ConcFB [5].

ConcFB introduces concurrency by spawning multiple disjoint Search Processes (SPs), which all perform SFB. This is depicted in Figure 5.11. In this case, agent a_1 initiates an SP for

each assignment, after which the SFB process is executed in parallel. The concurrency of the independent SPs makes the complete algorithm asynchronous. As can be seen, SPs can be initiated by different agents, like the splitting of SP 4 into 5 and 6 by agent a_2. This is called dynamic splitting. Furthermore, the ordering within each SP need not be equal, which allows for load balancing by shuffling the order heuristically or randomly. The first agent of each SP, however, cannot be reordered.

The algorithm consists of an overhead part dictating the workload balancing and dynamic splitting and a fundamental part dictating the steps within each SP. The algorithm is started by initiating the first agent with an empty CPA. Thereafter, propagating the CPA triggers the SP of the next agent. This SP utilizes SFB, as described in Algorithm 6. Possibly, the last agent finds a new best solution, which is then immediately shared over all concurrent SPs. If an agent depletes its domain, it sends a backtrack message to request a new CPA. As the first agent cannot do this, the algorithm terminates if the domain of the first agent is empty. For a more in-depth explanation, see [5].

Algorithm 6: Synchronous Forward Bounding (SFB)

```
1  CPA ← parent(CPA)
2  while domain not empty do
3  |  CPA.add(assignment)
4  |  if has child then
5  |  |  request(child_UBs)
6  |  |  await(child_UBs)
7  |  |  if CPA.cost + child_UBs > best_cost then
8  |  |  |  propagate(CPA)
9  |  |  |  await(backtrack)
10 |  else
11 |  |  if CPA.cost > best_cost then
12 |  |  |  best_cost ← CPA.cost
13 |  |  |  best_CPA ← CPA
14 backtrack(CPA)
15 return best_CPA
```

The functioning of the algorithm shows three main deficiencies. First, each subsequent agent independently sends its UB, which is less tight than a combined UB. Secondly, all the domain variables have to be explored, which can become intractable for the compiled domain as it grows exponentially on the number of shared tasks. Thirdly, no specific search strategy can be implemented, as there is no way of indicating which variables of the compiled domain are worthwhile checking first. This is in contrast to, for instance, the centralized branch & bound, in which relaxations play an important role in choosing the next branch to check. To address these deficiencies, Complex Concurrent Bounding (CCB) is proposed.

5-5-5 Complex Concurrent Bounding

In this section, the contributions leading to CCB are highlighted first. Then, an example is provided explaining the complete functionality of the local tree. Thereafter, the entire algorithm will be described in detail.

Contributions

Four adjustments are proposed for ConcFB. First, each DCOP agent now includes a local branch & bound tree, which allows for relaxing and pruning the compiled variable domain locally. Secondly, the UB requests are replaced by a local relaxation to retrieve tighter bounds at the loss of privacy guarantees. Thirdly, the relaxations are used to implement a best-first strategy to explore the compiled domains. Lastly, the lower bounds are shared over all SPs and the assignment of local tasks for each compiled solution.

Local search tree As shown in Equation (5.40), the compiled domain D_q of agent $q \in \mathcal{C}$ grows exponentially on the number of accessible shared tasks. With ConcFB, the complete domain has to be searched before a branch is pruned. As solving the local problem is a complex endeavor, the performance of ConcFB becomes questionable for increasingly larger numbers of shared tasks. Therefore, a local search tree is added such that the complete domain can be pruned before it is fully searched. Similar to branch-and-bound, each branch represents a binary choice on including a specific shared task. Lower bounds are then retrieved as the best known feasible solution, and upper bounds by relaxing the underlying problem. If the relaxations are sufficiently tight, branches can be pruned for which the $UB \leq LB$, possibly significantly improving performance.

To summarize, the local search tree consists of two main components:

- An Upper Bound (UB) for each branch by relaxing the cooperative formulation for all remaining agents in the complete ordering.
- A Lower Bound (LB) for each branch, by passing a feasible integer assignment to the next agent in the ordering and awaiting a broadcast for the corresponding solution. This is a key difference with branch-and-bound, where a lower bound can often be obtained by a heuristical transformation of the relaxed solution.

Removing UB requests To increase the tightness during relaxation, also the separate UB requests are addressed. In this application, a UB request would be replied by relaxing the local formulation from Section 5-4-2 linearly, and changing Constraint (5.18) to an inequality:

$$\sum_{k \in \mathcal{P}} x_{ik} \leq 1 \qquad \forall i \in \mathcal{M} \tag{5.41}$$

As each supervisor cannot know if any peer supervisor will perform the shared task it has access to, it cannot be forced to include it. Otherwise, capacity violation might occur for multiple agents performing the same task, invalidating it as an upper bound. Instead, it performs only the tasks that maximize the objective of the relaxed problem.

As in this DCOP application no privacy of information is required, a supervisor can simply relax the cooperative formulation from Section 5-5-2 directly, without consulting any succeeding agent. Then, Constraint (5.34) remains an equality, ensuring that the relaxed cooperative problem is at least as tight as the aggregated UB requests.

Best-first strategy By utilizing the local search tree and central relaxations, an opportunity is created to implement a best-first strategy. Given the performed relaxations, one could argue that passing an integer solution that succeeds the relaxed branch with the highest upper bound gives a higher probability of generating good solutions. This does not follow from ConcFB, as UB requests only indicate the performance of actual integer assignments, such that many requests would need to be sent first before an indication can be given which assignment is the most promising. In contrast, the local search tree generates objective values for distinct branches immediately.

Sharing compiled variables Any compiled local solution to a unique assignment of the shared variables is equal over all possible SPs that a supervisor participates in. Therefore, in addition to sharing any LB over all SPs, also the compiled variables are. However, the only difference is that the LB is useful for all supervisors in $\mathcal{C}$, and that the compiled variable domain D_q of supervisor $q \in \mathcal{C}$ is only used by the supervisor q itself. This implies the compiled domain does not have to be communicated.

Example local tree

To illustrate the concept of a local tree, consider the example with the complete ordering $A \Rightarrow B \Rightarrow C$, corresponding to the left image in Figure 5.9 and Table 5.1. The expansion of the local search tree of agent A is displayed in Figure 5.12.

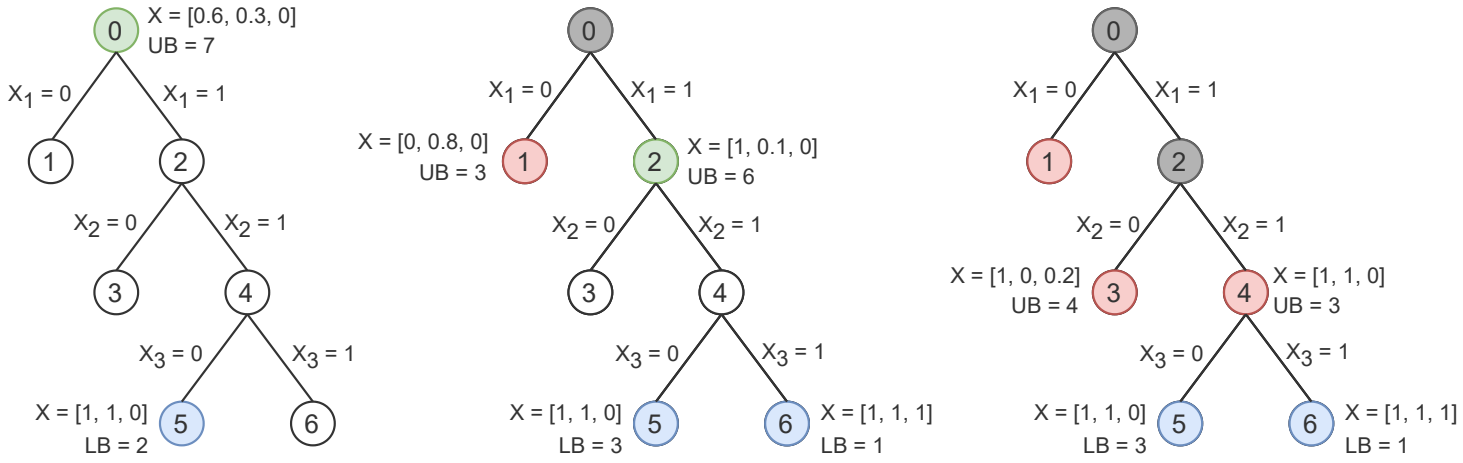


Figure 5.12: An example expansion of the local search tree. The green nodes are relaxed, the red pruned by optimality, the blue propagated and tested, and the grey are discarded.

For this example, it is assumed a globally known best solution (retrieved by a figurative parallel SP) is given as $LB = 5$. The steps of Figure 5.12 are then described as:

1. First, the cooperative problem is relaxed by agent A, with the resulting allocation vector $x = [0.6, 0.3, 0]$ and a corresponding UB of 7. This fractional allocation is rounded upwards to $x = [1, 1, 0]$ to create a feasible assignment, such that the tree can immediately be expanded to the leaf node corresponding with that assignment. Each leaf node of the local search tree corresponds with a compiled value in the variable domain, which can be propagated to the next agent. It then awaits a backtrack message from agent B, requesting a new CPA.

2. Upon receiving the backtrack message, the next round of relaxation is triggered in which the highest branches that are not relaxed yet are selected. In this case, nodes 1 and 2 are being relaxed. Node 1 is immediately pruned as the $UB \leq LB$. A new CPA is created by rounding up the fractional solution or by selecting any feasible assignment with the least needed changes, if the rounded assignment is already propagated. In this case, the compiled assignment of leaf node 6 is propagated.

3. Again, after the arrival of a new backtrack message, it performs a relaxation on the highest branches (node 3 and 4). Node 3 is immediately pruned due to optimality, and node 4 is pruned without relaxation, as all subsequent leaves have been explored already. The local search process is then terminated, as all branches have been pruned.

In comparison, for this search process four relaxations have been performed, while in the original ConcFB all eight leaves would have been relaxed. However, it should be noted that in the worst-case outcome $\sum_{n=0}^{\log_2(|D_q|)} 2^n$ relaxations have to be performed for CCB, while without the local tree (as with ConcFB) a fewer $|D_q|$ relaxations are needed. In the limit, this means that at most twice as many relaxations are performed by adding the local search tree, instead of relaxing the complete domain:

$$\lim_{|D_q| \Rightarrow \infty} \frac{\sum_{n=0}^{\log_2(|D_q|)} 2^n}{|D_q|} = \frac{2|D_q| - 1}{|D_q|} = 2 - \frac{1}{|D_q|} = 2 \tag{5.42}$$

However, with ConcFB the worst-case outcome of $|D_q|$ relaxations is always needed to ensure optimality, as the complete domain is explored. Furthermore, CCB should find a solution quicker due to a best-first strategy and decreased message complexity.

Complete algorithm

When the cooperative formulation needs to be solved using CCB, some steps are taken prior. First, each supervisor $q \in \mathcal{C}$ performs clustering on its assigned subset and broadcasts the result. Then, all supervisors compose their local and shared task set $\mathcal{M}_q^l$ and $\mathcal{M}^s$. A complete ordering is made by ascending connectivity (see Figure 5.8). Then, the first agent spawns multiple SPs. An SP remains active, constantly relaxing the local tree. It only stops when it is pruned by a parent, indicating the CPA is already found to be suboptimal earlier in the ordering. Continuing the search would then be fruitless. To request a new CPA, a cut message is sent to the parent, comparable to the backtrack message from SFB (see Algorithm

6). An agent reinitializes the tree if a new CPA is received. Lastly, if a new best solution is found it is shared over all SPs, as is any compiled solution of a full assignment (including the local tasks). Then, the pseudo-code from Algorithm 7 describes the SP for any agent.

Algorithm 7: Search Process (SP) in Complex Concurrent Bounding (CCB)

```
while not pruned do
    if new CPA received then
        Initialize local search tree
        Relax first node as UB
        Create integer shared task assignment
    else if domain not empty then
        Relax max UB branch
        Set new UB
        if cut received from child then
            Remove current assignment from domain
            Create integer shared task assignment

    if CPA.cost + UB > best_cost and domain is not empty then
        if shared task assignment not yet compiled then
            Solve local problem and save full assignment and cost
        else
            Retrieve full assignment and cost from compiler
        CPA.add(assignment)

        if CPA updated then
            if has child then
                propagate(CPA)
            else if CPA.cost > best_cost then
                best_cost ← CPA.cost
                best_CPA ← CPA
    else
        cut(CPA)
        prune(CPA)

return best_CPA
```

The first agent in the ordering creates new SPs by splitting an existing SP on the highest open branch. In the example of Figure 5.12, the tree can be split at node 1 and 2, creating two SPs. For a third SP the local tree starting at node 2 can be split further at node 3 and 4, and so forth. In such a small example this has no merit, but it illustrates the splitting in disjoint parts of domain. Furthermore, for every SP the remaining agent order is changed. Netzer et al. [5] propose more advanced dynamic reordering to balance the workload, but for this implementation a random unused reordering is selected. As the root agent is already fixed, this would mean that an SP can be created with the ordering $C \Rightarrow B \Rightarrow A$ and $C \Rightarrow A \Rightarrow B$ for the example in Figure 5.8.

5-6 Combining the components

At the beginning of this chapter, the solution structure is outlined in Section 5-1 using Figure 5.1. As the approach to each component in this structure has been clarified, the solution flow can now be described. To do so, each component is linked to a specific step in the solution process in Figure 5.13. The input of the algorithm always consists of mission information, which contains an Area of Interest (AOI), sector priorities P, and terrain traversability M. The heterogeneous agents and hierarchy are assumed to be known by each supervisor.

When a supervisor is triggered to create or revise its subordinate subsets, it performs the steps as outlined in the structure. First, clustering is done locally. Using the discounted degrees, the shared and local task sets are then retrieved, and distributed cooperation is performed using CCB to allocate the tasks to the subordinates. Upon (preemptive) termination, the resulting subsets are sent to the subordinates.

The crucial part of this solution process is that the revision of subsets is triggered both by initializing and changing the information. However, when the information is initialized only the root supervisor is triggered, such that the creation of subsets occurs sequentially through each layer of the hierarchy. When information is changed, all levels in the hierarchy are triggered *simultaneously*. This means that initially, they will revise the subsets within the overlap provided by their respective supervisor, but that they revise the solution again when new subsets are received. This allows the lowest levels to react quickly in a holarchic manner, though over time the sequential top-down revision of the subsets will also be completed.

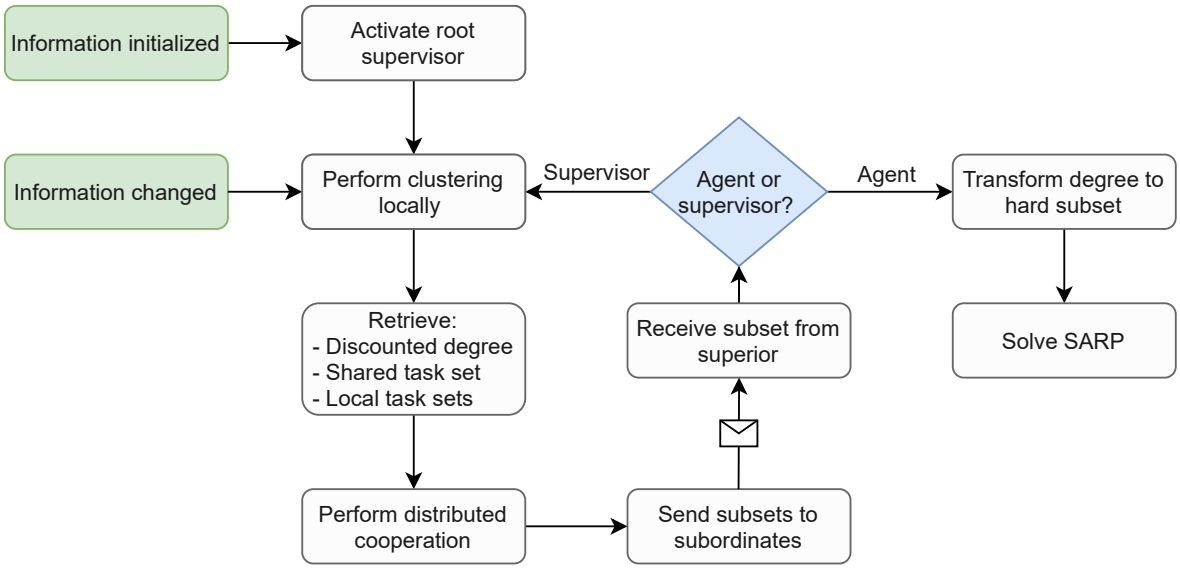


Figure 5.13: Solution flow of the distributed hierarchical intent-based coordination.

At the lowest level, the remaining overlap between the agents' subsets is not cooperatively deconflicted. Instead, the final degrees are transformed to hard subsets by assigning each sector to the agent with the globally largest allocated degree to that sector. In reality, this would require additional communication, which is not included here. Using these hard subsets, disjoint SARPs can be solved and executed by the agents.

Combined, this method should allow for sufficient flexibility, as solutions can be revised immediately. Furthermore, the method should demonstrate robustness against communication failures as this revision is possible without consulting the superior. Even more so, if cooperation of superiors is suboptimal due to preemptive termination by communication problems, the same degree of freedom can be used to mitigate these effects locally. Lastly, the approach is expected to be a scalable approach due to the natural parallelization of the computations in the hierarchy and the restricted problem size of task allocation using clustering.

Chapter 6

Performance Analysis

Fundamentally, it is argued that persistent reconnaissance is performed well using the Multi-Agent Reconnaissance Problem (MARP). To find a solution in a military environment, intent-based coordination is proposed to partition the Area of Interest (AOI) into disjoint subsets, such that multiple Single-Agent Reconnaissance Problem (SARP) can be solved independently, using for instance a heuristical approximation (Section 4-3).

There are three available benchmarks to assess if the solutions of the combined SARPs approximate the optimal MARP solution: 1) the actual optimal MARP solution, which is only tractable for small instances, 2) a lower bound obtained by pricing (Section 4-1), which is less intensive, but limited too as the Sub Problem (SP) is an NP-hard problem which needs to be solved at least once, and 3) an upper bound obtained by trivial allocation (Section 4-2).

Not only does the solution method need to approximate the optimal solution sufficiently well, but it also has to do so while exhibiting the properties of flexibility, robustness, and scalability, as outlined in Section 5-1. These properties are expected to ensure that a quality solution will also be obtainable in an uncertain and dynamic environment.

The analysis is split into three parts, as displayed in Figure 6.1. First, the default parameters are analysed, of which the results are listed in Section 6-1-2. The interested reader can find the justification in Appendix A. Subsequently, all separate components are analysed on convergence speeds and solution quality in Section 6-2. Based on these insights, the combined performance of the complete method in a military environment is assessed in Section 6-3.

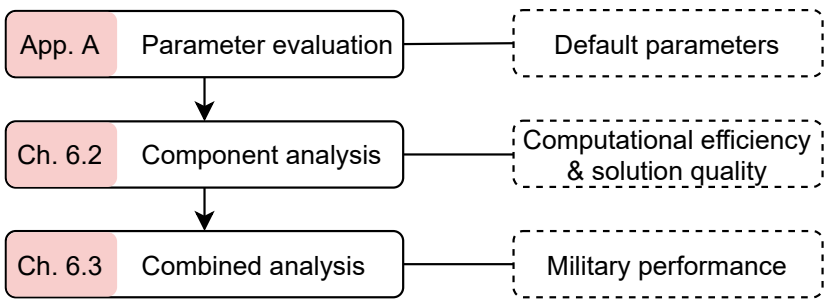


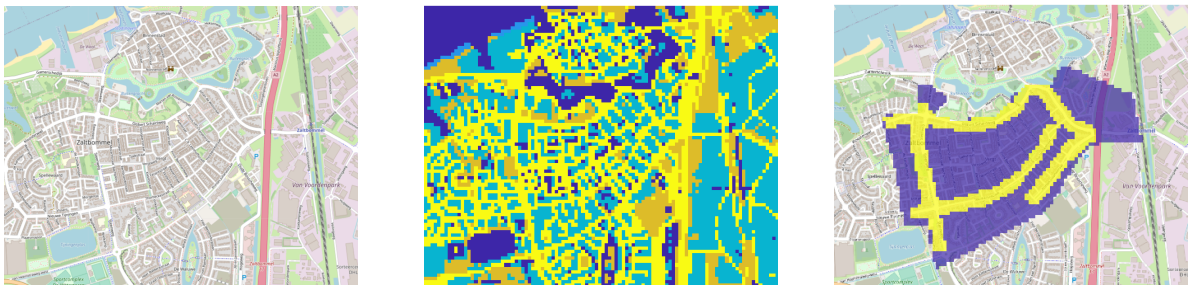
Figure 6.1: The analysis structure with the corresponding objectives.

6-1 Experiment setup

The setup for the performed simulations consists of a high-resolution scenario with terrain and priority data. A method is added to decrease the resolution to create smaller problem instances. Furthermore, the default parameter setup is provided.

6-1-1 Scenario

A neighborhood from an unspecified city is used as the scenario (Figure 6.2a). The terrain traversability M_i consists of hardly inaccessible buildings $M_i = 0.5$, roads $M_i = 1$, grass and low vegetation $M_i = 0.8$, and inaccessible water $M_i = 0$ (Figure 6.2b). The priorities defined by the operator are $P_i = 2$ for the neighborhood in general and $P_i = 3$ for the main roads (Figure 6.2c) for all $i \in \mathcal{N}$. The AOI consist of $84 \cdot 106 = 8904$ sectors of each $25 \cdot 25m$. All sectors with $P_i = 0$ can be discarded, as no risk can be present there (see Section 3-3). This means that 2275 sectors with $P_i > 0$ remain, approximating a total of $1.42km^2$.



(a) AOI **(b)** Traversability **(c)** Priority

Figure 6.2: The Area of Interest (AOI) is displayed, with a mapping of the corresponding terrain traversability M, and the priorities P provided by an operator.

Furthermore, a method is added which can reduce the problem size to enable the testing of smaller problems. Effectively, the resolution is lowered, and some sectors are removed randomly to retrieve a problem instance with an exact, predefined number of sectors. This effect is displayed in Figure 6.3, and enables precise comparison for varying instances.

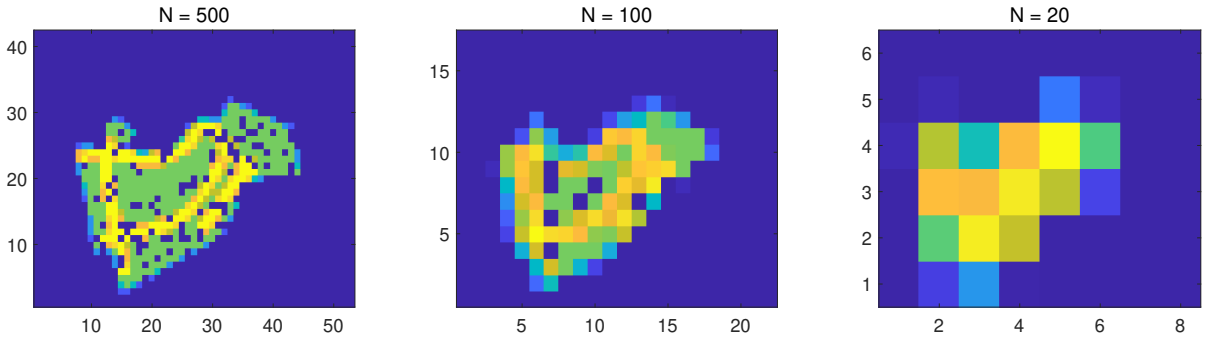


Figure 6.3: An example of reducing the resolution to match a specific amount of sectors.

6-1-2 Default parameter values

Based on the experiments provided in Appendix A, a set of default parameters is constructed for all subsequent experiments. This default setup is used unless stated otherwise. The resulting parameters are listed in Table 6.1.

Table 6.1: Default parameter values

Symbol	Value	Parameter
m	1.15	Fuzzifier
σ^h	3	Heterogeneity standard deviation
e	1.8	Overtime penalty
n^b	1	Capacity scaling
n^c	4	Maximum number of clusters per agent
ϵ	0.001	Clustering improvement threshold
β	0.03	Cut-off degree
n	20%	n^{th} percentile giving degree cap γ
δ_i^0	0.5	Initial uncertainty $\forall i \in \mathcal{N}$

The constants S, D, and ρ are discussed in Section 3-3 separately, as they do not influence to solution method itself but only the underlying model, which is assumed correct. Furthermore, some notions are used throughout this chapter. The number of sectors $|\mathcal{N}|$ is denoted as N, the number of clusters/tasks $|\mathcal{M}|$ as M, and the number of agents (sensors) $|\mathcal{P}|$ as P.

Additionally, some parameters are set using a formula. For instance, the number of clusters is set to the median of all possible numbers of clusters, depending on each possible number of sectors per cluster n, limited by a maximum of n^c clusters per subordinate, or the total number of sectors N:

$$M = \min\left(median\left\{\left\lceil\frac{N}{n}\right\rceil \;:\; \forall\, n \leq N,\; \frac{N}{P\cdot n} \geq 1\right\},\; P\cdot n^c,\; N\right) \tag{6.1}$$

Furthermore, the agents are initialized using the heterogeneity standard deviation σ^h. For each of the properties speed v_k in m/s and flight time e_k in minutes, a vector of length P is generated a 1000 times based on a normal distribution, using $\max(\mathcal{N}(14, \sigma^h), 5)$ and $\max(\mathcal{N}(35, 2\sigma^h), 10)$ respectively. For speed, this implies an average speed of 14 m/s, and at least 5 m/s. For flight time, this implies 35 min on average and 10 min at minimum. The property vector with a standard deviation closest to σ^h for speed, and $2\sigma^h$ for flight time, is selected. The starting location is set to the center of the AOI for all sensors. The absolute capacity b_k is calculated as follows:

$$b_k = \frac{60e_k \cdot d_k v_k}{d_s^2} \qquad \forall k \in \mathcal{P} \tag{6.2}$$

Here, d_k is the width of the sensor view. In reality, this is dependent on the altitude and field of view, but it is currently fixed to the sector size $d^s = 25m$ for all $k \in \mathcal{P}$. Instead of the absolute capacity, the scaled, normalized capacity $\hat{b}_k = n^b \cdot b_k / \sum_{k\in\mathcal{P}} b_k$ is implemented.

6-2 Component performance

In this section, the performance of all separate components of the solution approach is assessed. The components include the upper- and lower bounds, task allocation, and distributed cooperation. Depending on the component, performance is regarded both in computational tractability and in obtained objective value.

6-2-1 Benchmarks

Multiple benchmarking methods are used to compare the performance of the solution approach. First, a priced lower bound is implemented (Section 4-1), which is compared on computational efficiency and tightness. Secondly, a heuristical approximation is implemented to obtain a solution for the SARP (Section 4-3), which is only assessed on tightness.

Priced lower bound computational efficiency

In this first experiment, the solution time for the column generated lower bound is compared with solving the MARP to optimality using Gurobi. The experiments are performed for an increasing number of sectors $N = \{1 \dots 50\}$, and increasing sensors $P = \{1 \dots 30\}$. The experiment is repeated five times, and the results are averaged. During computations, the method is cut-off if the computation time exceeds 5 seconds. The results are displayed in Figure 6.4. It is clearly visible that the column generation depends highly on the number of sensors. The more sensors are included, the more efficient the search process. This in contrast to the exact method for which the run time increases when including more sensors.

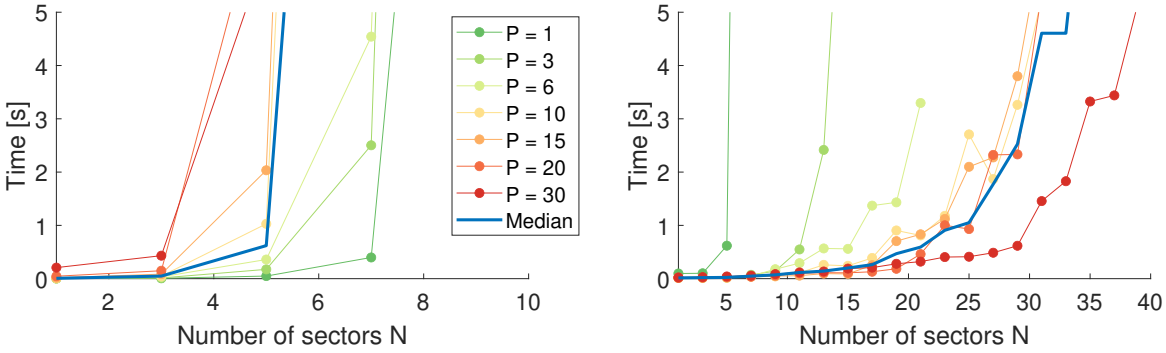


Figure 6.4: A runtime comparison between the exact solution and the priced lower bound for different numbers of sectors and sensors. Pricing is inversely dependent on the number sensors compared to the optimal approach and can handle significantly larger instances, though the runtime remains exponentially dependent.

Priced lower bound tightness

Apart from the runtimes, also the tightness is considered for the same experiment. A relatively small set of instances can be compared, as the computation time explodes inversely for both methods, making it difficult to obtain data.

The results are displayed in Figure 6.5. Apart from the outliers of a single agent, the method performs consistently regardless of the combination of sensors or sectors. A 95% confidence for each setup clearly shows this. On average, the lower bound returns an objective ≈ 0.38 times that of the optimal value.

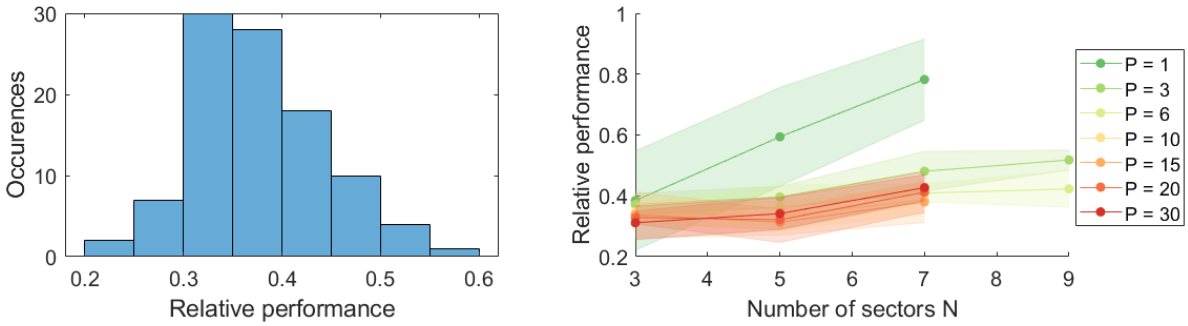


Figure 6.5: Left: a histogram of the performance of each lower bound, excluding the single-agent case. Right: the average tightness of the priced lower bound with a 95% confidence interval for each amount of sensors. Apart from the single-agent case, the results are very consistent.

SARP heuristic tightness

To obtain a solution for the SARP, multiple heuristics can be used, which are explained in Section 4-3. These heuristics are not formulated to provide a very good solution to the SARP per se but should generate sufficiently close solutions to the optimum to allow comparison in further experiments.

An experiment is performed to benchmark these heuristics against the optimal solution. To do so, the task allocation is solved for sectors ranging from $N = \{4 \dots 25\}$, and the number sensors within $P = 1, 2, 4$. After task allocation, multiple disjoint SARPs are obtained, for which a quick solution is calculated using the heuristics and the optimal solution using Gurobi. A comparison is then made between the combined objective values of the optimal solutions and the combined values of the heuristics.

The results are depicted in Figure 6.6. It is clearly visible that the heuristic's overall usability depends on the average size of the SARPs, and not directly on the total number of sectors. For instance, in the case of twenty sectors and four sensors, the average number of sectors per SARP is around $20/4 = 5$, making the average performance better than the case with seven sectors and one agent.

To illustrate the relation with the SARP size, the average number of sectors per SARP is collected for each simulation. Then, the error is displayed for the sorted average sizes in the right plot of Figure 6.6. Furthermore, a 2^{nd} order polynomial fit is displayed. A clear trend is visible, as the task allocation method tries to spread out the load of each agent evenly. In the case of the default σ^h, the average SARP size can be reasonably approximated by N/P. If agents become more heterogeneous, this cannot be expected.

The same experiment is repeated for the upper bound with trivial clustering and allocation, explained in Section 4-2. The results are displayed in Figure 6.7. Immediately, it is observed that the gaps with the optimal value are more volatile. Because the load is not balanced

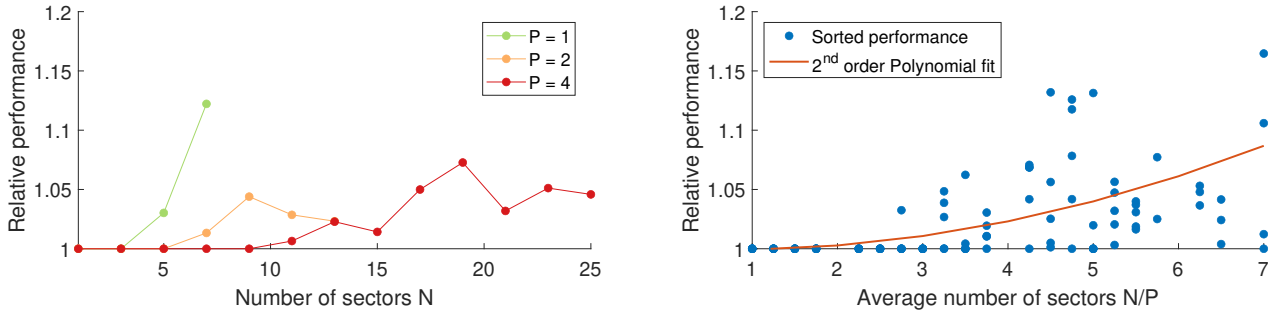


Figure 6.6: Left: the relative performance of the heuristics for different numbers of sectors and sensors. Right: A plot with 2nd order polynomial fit of the performance sorted on the average number of sectors, showing performance is dependent on the SARP size, and not the total number of sectors.

as well, the average SARP size N/P is a less accurate representation of the actual problem sizes. In the right plot this is shown. The results are much more scattered, and a trend is hardly visible. This means that when simulating larger problems, it can be expected that the random upper bound produces more results with occasionally very large SARPs, meaning that the heuristic approximation tends to overestimate the solution value obtained by the random upper bound. Nonetheless, the differences are not so dramatic as to prohibit the usage as a benchmark.

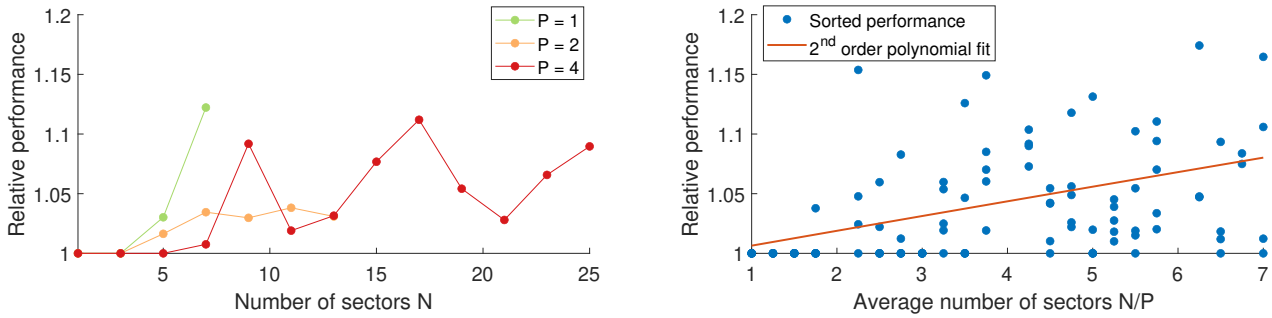


Figure 6.7: Left: the relative performance of the heuristics for different numbers of sectors and sensors. The gap is more volatile due to the lack of load balancing, leading to more variation in SARP sizes. Right: A plot with 2nd order polynomial fit of the performance sorted on the average SARP size N/P. The trend is hardly visible compared to the actual task allocation.

6-2-2 Task allocation

Using the previous results, the performance of the task allocation as formulated in Section 5-4 can be assessed. Note that the cooperative formulation is not tested yet, so no hierarchy is present. Instead, a single supervisor clusters the sectors and assigns them using local task allocation to any agent. The solution quality of the task allocation is assessed using the available benchmarks first. Then, the computational efficiency is analysed. Last, the specific performance of the post-processing interpolation is quantified.

Solution quality

The solution quality is assessed over a range of $P = \{2, 4, 6\}$ sensors. The number of sectors are selected from $N = \{2 \dots 30\}$. The results are compared to the available benchmarks. First, the optimal value is included up to seven sectors, after which the values are extrapolated linearly. A 95% confidence interval is included for both the optimal values and the averaged 2^{nd} order extrapolations of each setup. For the extrapolation, this interval gives some idea about the spread in the extrapolated values. Furthermore, three upper bounds are included. In the first, trivial clusters are created, which are allocated optimally. The second method uses actual clustering but with trivial allocation. The third combines these (see Section 4-2). Lastly, also the priced and linearly relaxed lower bound are included.

The results are displayed in Figure 6.8. The first thing noticed is that the upper bound for trivial clustering performs equal or even slightly better than the Fuzzy C-Means (FCM) clusters. This means that, at least for these problem sizes, the clustering has no added benefit, which is also suggested by the results from Appendix B. Furthermore, the 2^{nd} order extrapolation suggests that for larger problems, the performance goes down. On the one hand, this is supported by the fact that the performance approaches the upper bounds. On the other hand, the performance slightly improves with respect to the lower bound, which is shown to remain relatively constant to the optimal value. This suggests that the extrapolation underestimates the true cost. Furthermore, in both cases the values for a larger number of sectors are based on the SARP heuristics. This means that the displayed values overestimate the cost and that its true benefit is larger. As discussed in the previous section, however, this effect can be expected to be larger for the upper bound.

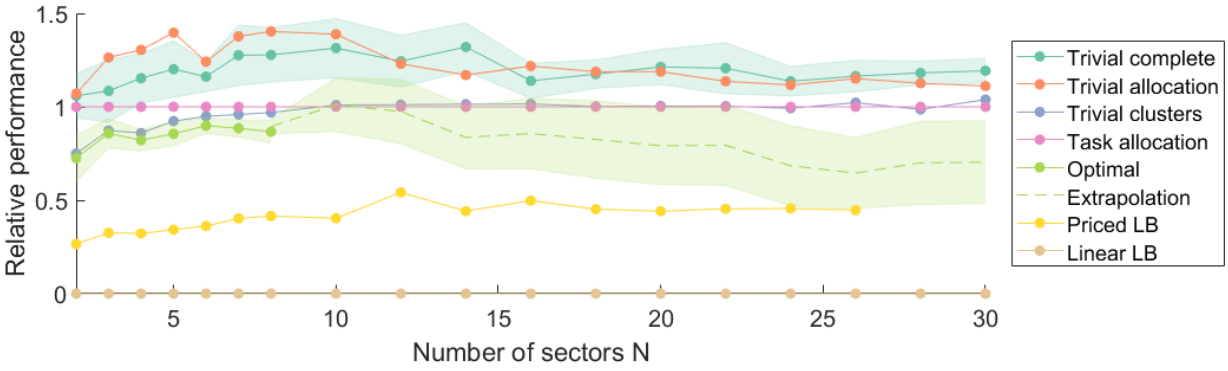


Figure 6.8: The relative performance of the single task allocation problem compared to different benchmarks. Based on these results, it is reasonable to expect a performance decrease for an increasing number of sectors.

For more insight into the extrapolations, the average values are displayed in Figure 6.9 for each sensor setup. Not only the optimal values are included, but also the task allocation values. Both the task allocation and extrapolated optimal values show a nonlinear trend, possibly even exponential. The cost for the single-agent case for an equal number of sectors seems to grow more rapidly, which is understandable given the objective function, as it includes much larger distances for a single agent. Though the four agent case is much closer to the extrapolated values, $P = 2$ and $P = 6$ show comparable performance, indicating no trend in increasing deviations between different sensor setups.

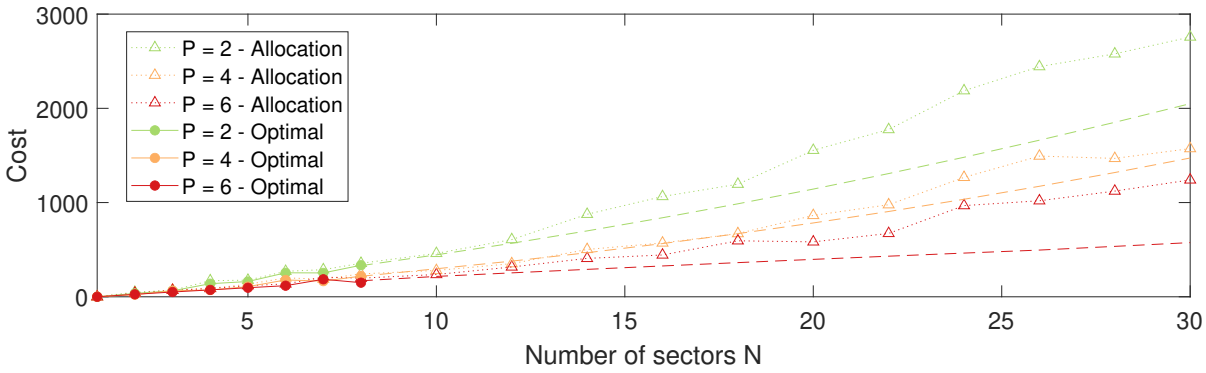


Figure 6.9: The extrapolated values for the optimal values (dashed) are compared with both the optimal values (solid) and the task allocation solutions (dotted). These results do not indicate large deviations between different sensor setups.

Computational efficiency

Apart from the solution quality, the computational speed is analysed. The numbers of sectors are varied from $N = 20 \dots 200$, and the amount of sensors from $P = 1 \dots 6$. The number of clusters is set using the formula described in Section 6-1-2.

The results are displayed in Figure 6.10, which shows the resulting number of clusters (left), the dependence of the processing time on the number of sectors (center), and the number of sensors (right). It is clearly visible that increasing the amount of sensors has a much greater influence than adding clusters. This makes sense, as adding a cluster introduces P new decisions while adding a sensor results in M new decisions. Generally, $M \gg P$ such that the complexity depends more heavily on the number of sensors.

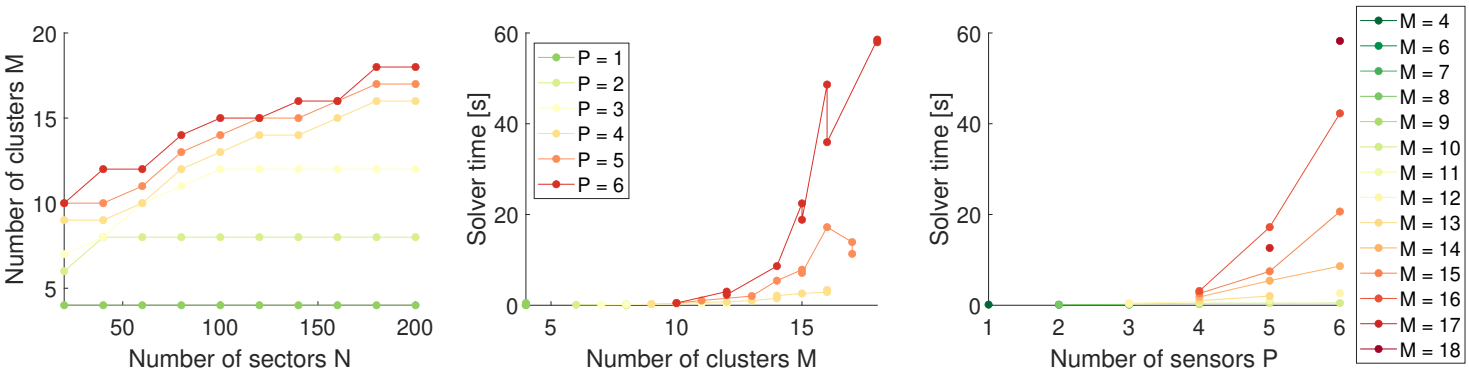


Figure 6.10: Left: the resulting number of clusters for each sensor setup. Center/right: The processing time, based on the number of clusters and sensors. The addition of more sensors has a larger impact.

Cluster post-processing

Lastly, also the added benefit of the interpolation during cluster post-processing is assessed. The goal is to mitigate the negative effects of clusters with physically scattered sectors (see

Figure 5.5). In this experiment, the added benefit of this method is quantified by solving the task allocation using the same clusters with and without post-processing. The amount of overlap created during interpolation is dependent on the clustering weights, the fuzziness, and the number of sectors and clusters (see Appendix A). Therefore, two experiments are performed, both including four sensors, with the number of sectors varying from $N = \{50 \dots 400\}$. In the first experiment, the fuzziness is varied within $m = \{1.1, 1.3, 1.6\}$, and in the second the clustering weights on the coordinates are scaled by a factor $\kappa = \{0.5, 1, 1.5\}$. The average results for four simulations are displayed in Figure 6.11, with left the varying fuzziness, and right the varying cluster weights. A 95% confidence interval is plotted for all setups. For both experiments, the confidence intervals are wide and show no significant difference with the solution including the post-processing. Therefore, based on these results, it cannot be concluded that the post-processing method has a significant effect.

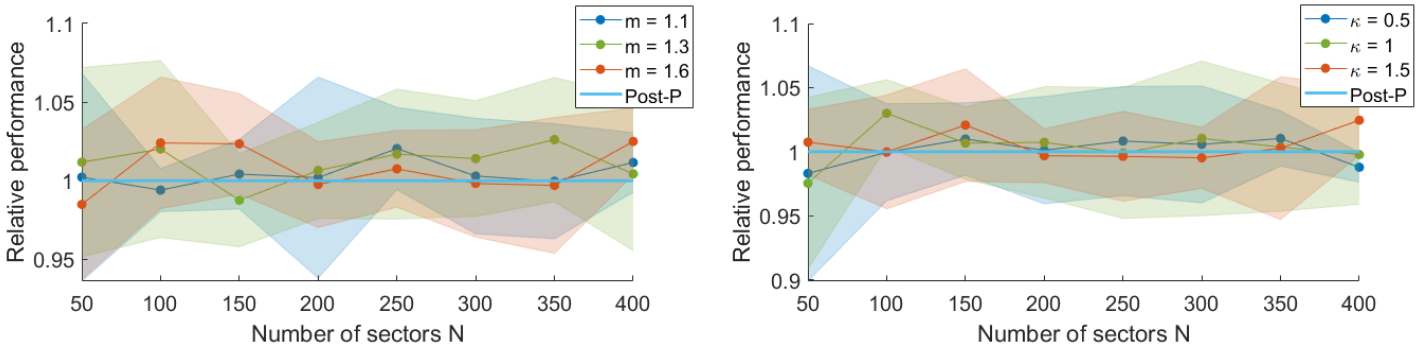


Figure 6.11: Left: the relative performance of task allocation without post-processing, for different values of the fuzzifier m, compared to the case with post-processing. Right: the same comparison, but for varying factors p increasing the clustering weights for the sector coordinates. Based on these results, no significant benefit can be observed.

6-2-3 Distributed cooperation

As formalized in Section 5-2, the concept of intent describes how overlapping clusters are created. In Section 5-5-1 it is stated that overlapping clusters with peer supervisors in the hierarchy lead to shared tasks, which are included in the cooperative formulation from Section 5-5-2. To solve this problem in a distributed fashion, the novel Complex Concurrent Bounding (CCB) algorithm is proposed, of which a full description is provided in Section 5-5-5.

The goal of the CCB algorithm is to produce solutions that are closer to the optimum in less time than alternative approaches. Furthermore, the distribution of the problem should prevent a single bottleneck. Therefore, the convergence times are compared with a centralized Gurobi implementation and a simulation of Concurrent Forward-Bounding (ConcFB), on which it is based. Furthermore, as it is an anytime algorithm, the effect of a preemptive, suboptimal solution is assessed. Lastly, the effect of the complete ordering on CCB performance is analysed.

Absolute convergence

Convergence is interpreted as the gap towards the optimal solution. Two characteristics are important when measuring this: the time to an initial acceptable solution and the tail towards the actual optimal value. For this experiment, four peer superiors are considered with each four agents, resulting in $P = 16$. The number of sectors is varied between $N = \{20, 50, 80\}$, and the fuzzifier from $m = \{1, 1.15, 1.3, 1.5\}$. Each experiment is repeated five times, and the results are averaged. Because each run returns new solutions at different times, the improvements are interpolated. To account for the different times that an initial result is obtained, results are displayed from the average time the first results were obtained.

The results are shown in Figure 6.12 in semi-log plots with a 95% confidence interval. In the left plot, the convergence for different sectors is shown. For larger problems, the time to an initial solution and the tail convergence both become larger. In the left plot, the same results are shown for different levels of fuzziness. It shows consistent evidence that a higher fuzzifier leads to slower convergence overall. Particularly outstanding is the case with $m = 1$, in which (almost) no overlap is present. The cooperation then only consists of solving the local problems, which is performed to optimality within 0.1 seconds on average.

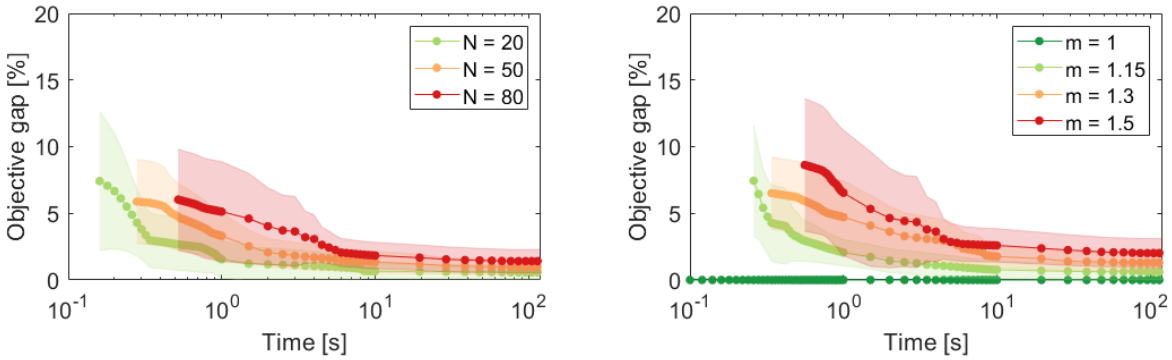


Figure 6.12: A plot showing the absolute convergence time for different amount of sensors and fuzziness on a logarithmic scale. More sectors and larger fuzziness result in slower initial and tail convergence.

Relative convergence

The convergence of the solution value is compared to that of the centralized Gurobi implementation and the original ConcFB algorithm. The comparison is relative to another algorithm by displaying the absolute difference Δ in the obtained optimality gap at a specific time. Again, the results need to be interpolated to ensure that a comparison can be made. Therefore, to prevent skewed results due to less data being available, the comparison is only presented from the initial time both methods have obtained a first result on average.

Gurobi Figure 6.13 shows the results when comparing to Gurobi. In the left plot, the results for different amounts of sectors are displayed, and in the right plot the same results are shown for different fuzzifiers. Both plots include a 95% confidence interval. The results suggest that

in a significant number of cases, CCB can obtain a better result initially. However, after the initial fraction of a second, Gurobi outperforms CCB. Furthermore, a consistent gap can be observed, indicating long tail convergence, comparable to the results from Figure 6.12. This effect is more prominent for larger fuzzifiers but not necessarily for more sectors.

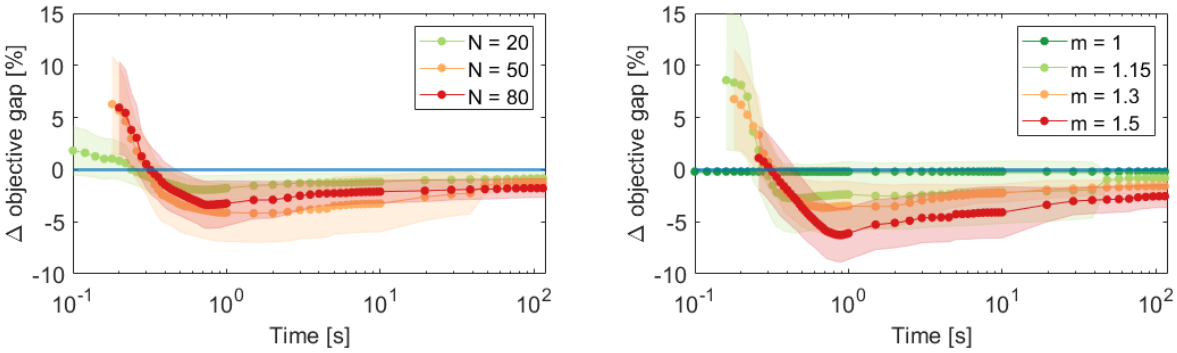


Figure 6.13: The relative convergence is measured as the absolute difference in optimality gap between CCB and Gurobi. A positive difference indicates that CCB has a smaller gap on average at that time. During the initial fraction of a second, CCB seems to perform better. After that, it performs worse due to slow convergence.

Furthermore, in Figure 6.14 a further insight is provided for the initial stages of optimization. Gurobi is found to provide a first solution quicker than CCB in 98.3% of the cases (not depicted) but that this first result is on average equal or worse than 83.3% of the first obtained results by CCB, which can be seen from the asymmetrical histogram displayed in the left image of Figure 6.14. In the right plot, the difference in optimality gap is compared at the time CCB receives its initial result. Then, in only 30% of the cases CCB performs better. Three important considerations need to be included to interpret these results. First, in a practical implementation it can be expected that communication delays worsen CCB performance. Second, not all recommendations from ConcFB are included yet, like dynamic splitting, which can increase performance. Thirdly, a weak linear relaxation of the cooperative problem might result in insufficient bounding, reducing CCB convergence. In any case, the results suggest that initial results are comparable but that a gap remains.

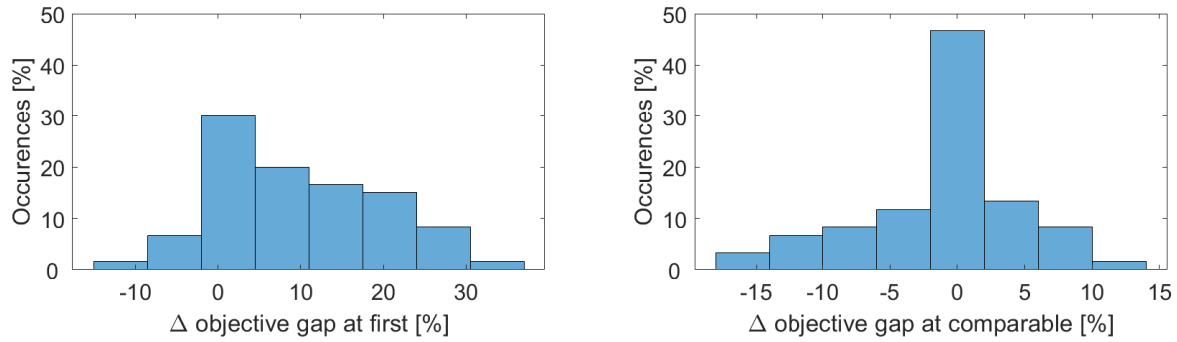


Figure 6.14: A comparison of the difference in optimality gap of the first obtained result by CCB and Gurobi (left), and the difference at the time both have a first result (right).

ConcFB To give some further insights in CCB, it is compared with ConcFB. The ConcFB algorithm is simulated by modifying the CCB algorithm. This is done in three steps. First, the concept of the local tree is removed by only allowing relaxations on leaf nodes, which represent feasible solutions that can be passed down the complete ordering. Secondly, the linear relaxation of the cooperative formulation from Section 5-5-2 is not used, but instead UB requests are sent to all succeeding superiors. These provide a local upper bound by linearly relaxing the local formulation from Section 5-4-2, while changing Constraint (5.18) to an inequality. As each supervisor does not know if any peer performs a shared task, it cannot be forced to include it. Instead, it performs only the tasks that maximize the objective of the relaxed problem. As a consequence, the aggregated upper bounds are guaranteed to be less tight than the relaxed cooperative problem. Thirdly, as no relaxation is available including all agents, proper initialization using the best-first strategy cannot be used. Instead, a Current Partial Assignment (CPA) is initialized by simply performing all accessible shared tasks.

The results for these three changes are depicted in Figure 6.15. Each setup is simulated five times using $N = \{20, 50, 80\}$, $m = \{1, \dots, 1.5\}$. The results are interpolated, but displayed only when, on average, both methods have obtained a first result. Clearly, the CCB algorithm outperforms ConcFB. For larger fuzzifiers and a higher number of sectors, the benefit increases. The results obtained in the initial stages are much better, and the tail convergence is closer to the optimum for CCB.

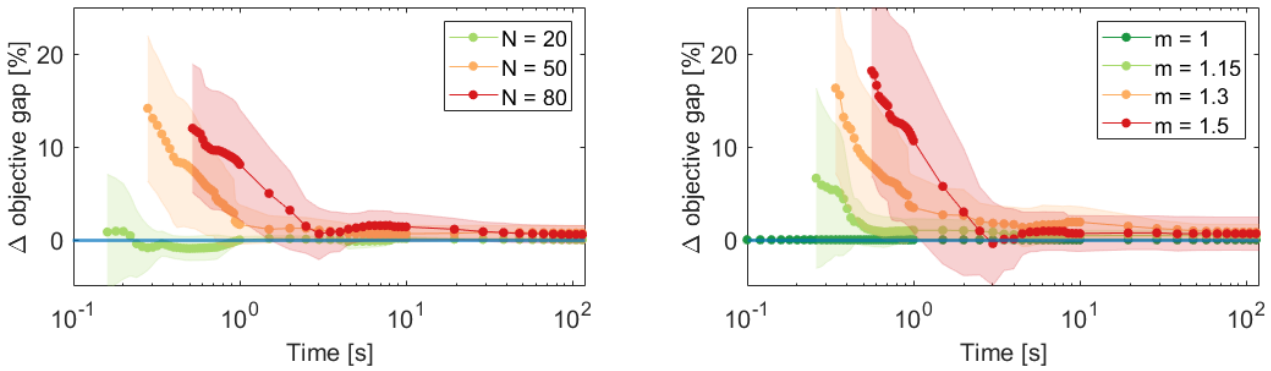


Figure 6.15: The relative convergence between CCB and ConcFB, which has no local tree, uses separate UB requests and naive initialization of the CPA. CCB clearly outperforms ConcFB, for which the effect is larger for increasing complexity.

To separately analyse the effect of the different changes to ConcFB, the experiment is repeated with different setups, including only a part of the changes. The results for this are displayed in Figure 6.16. Again, the absolute difference in the objective gap is displayed in the left plot. Only removing the local tree shows no significant difference or even a slight improvement. The setups which use the trivial initialization of the CPA perform much worse and using UB requests is slightly worse overall. It can be concluded that the initialization by local relaxation of the cooperative formulation has a major impact compared to the other changes. Possibly, the minor effects of including UB requests and the removal of the local tree can be attributed to an insufficiently tight formulation of the cooperative problem, as only a basic linearization of the quadratic constraint is implemented. In the right plot from Figure 6.16 the percentage of interpolated optimality gaps that are equal or better compared to CCB are displayed for varying fuzziness at each time instance. All three setups perform equally well

when no fuzziness is present. As (almost) no shared tasks are present. When more shared tasks are added because of larger fuzziness, the lack of proper initialization has an increasingly larger effect. The addition of UB requests has a slightly worsening effect too. However, the contrary is true for the removal of the local tree. As the fuzziness increases, the local tree becomes larger and increasingly inefficient. Again, this might be due to an insufficiently tight relaxation of the cooperative problem, such that more relaxations need to be performed when adding a local search tree.

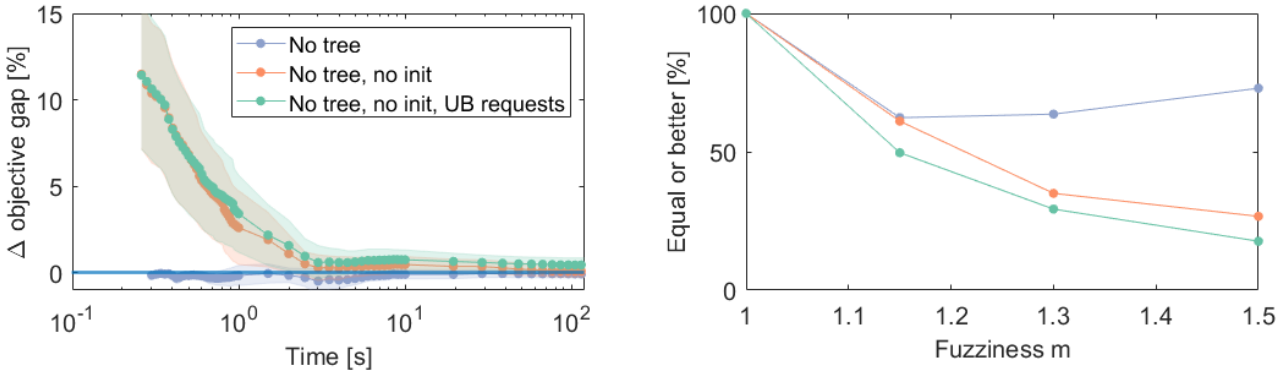


Figure 6.16: Left: the relative convergence between CCB and partial ConcFB setups. Improper initialization has a major effect, and UB requests only minor. Removing the local tree is even slightly beneficial. Right: Percentage of cases in which a partial ConcFB setup performed equal or better than the CCB result at the same time. The larger the fuzziness, the larger the negative effect of improper initialization and UB requests. The contrary is true for the local search tree.

Effect of different orderings

As explained in Section 5-5-5, the initial complete ordering is important, as the root superior remains the same when splitting the Search Processes (SPs). Therefore, the effect of different orderings is assessed. These orderings are based on the connectivity, as displayed in Figure 5.8b. Three variants are implemented: ascending, descending, and an initial random ordering.

The results are displayed in Figure 6.17. It is clear immediately that the ascending order outperforms the descending order. Furthermore, the random ordering shows consistent performance approximating the average between the ascending and descending approach. This strongly suggests ascending and descending are the best and worst in general, respectively. Ascending connectivity implies that the first agent to assign shared tasks has the least overlap with other superiors, which is beneficial as the root agent cannot be changed. Concrete, this means that the workload can be balanced more efficiently. Furthermore, the benefit is expected to become even larger when any superior is allowed to split an SP. Currently, only the first is permitted to do so, which limits the number of splits possible. The SP cannot be split when there is too little overlap, which occurs more often in the ascending case. This implies that less SPs are spawned in total, which increases the total computation time. Despite this, ascending still outperforms any other ordering on average.

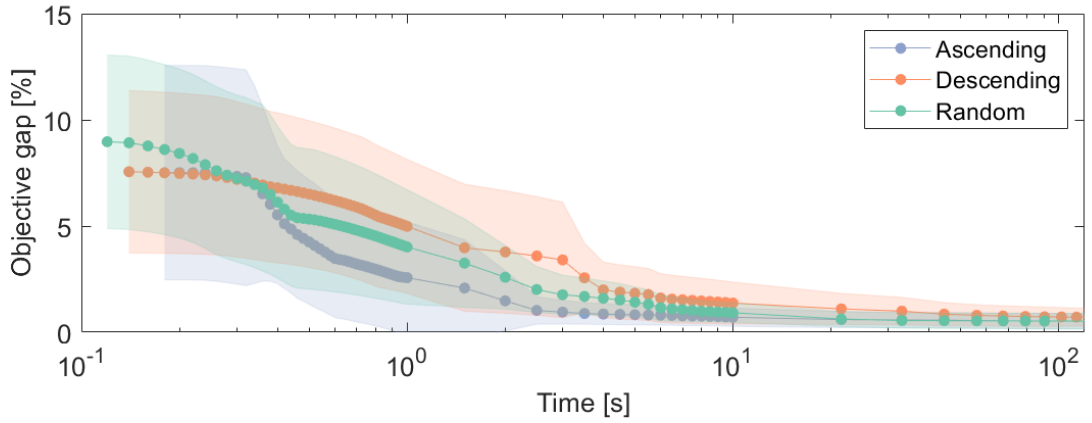


Figure 6.17: Based on the connectivity, which describes the number of peers a supervisor shares a task with, different orderings are tested. An ascending connectivity ordering consistently outperforms any other ordering, where descending appears to be the least effective.

Effect of suboptimal allocation

Task allocation is based on the assumption that a good allocation results in good SARP solutions that approach the MARP solution when combined. The validity of this assumption is benchmarked in Section 6-2-2 using optimal allocation without cooperation. Therefore, the effect of intermediate suboptimal allocation on the resulting SARP solutions needs to be assessed. The expected effect is two-fold. When there is a complete lack of optimization, no tasks are deconflicted, resulting in the worst-case outcome. Any overlap in tasks is then performed by multiple agents, which can increase the costs dramatically. Second, if any optimization has taken place, but the allocation is still suboptimal, the sectors might not be allocated to the appropriate agents, and thus the cost increases.

For this experiment, all solutions generated by CCB are collected. Furthermore, the centralized solver is also used with a different gap stopping criteria, set to $\{30\%, 15\%, 5\%, 2\%, 0.1\%\}$. The solver will stop when the actual gap falls below this limit. The experiment is performed for $N = \{20, 40, 60, 80\}$, $m = \{1, 1.15, 1.3\}$, $P = 16$ and four supervisors.

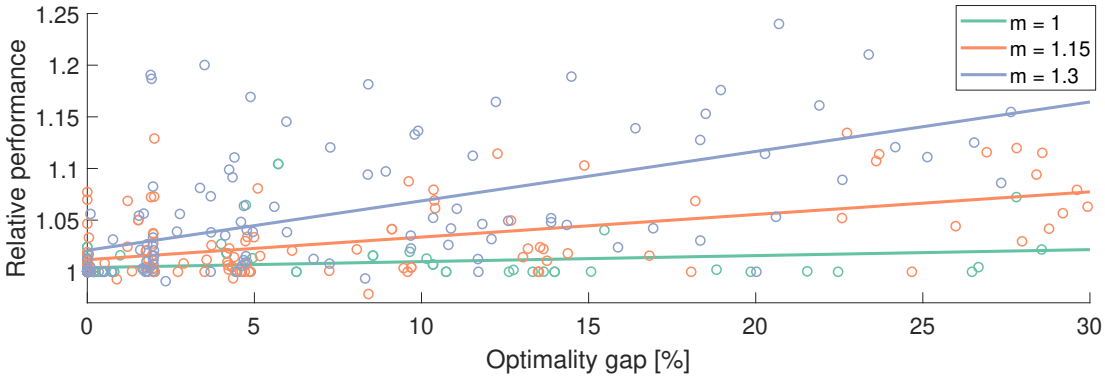


Figure 6.18: The linear fit shows that a larger allocation gap leads on average to worse MARP solutions, for which the effect increases with larger fuzziness, implying more shared tasks.

The results are plotted in Figure 6.18, and colored depending on the fuzziness. Clearly, the larger the fuzziness, the larger the effect of better task allocation. For $m = 1$, there is hardly any task to cooperate about, such that an optimality gap has a minor impact. As the top-down task allocation is a heuristical approach for the MARP, increasing the solution quality does not guarantee a better solution to the MARP, which is visible from the scattered data.

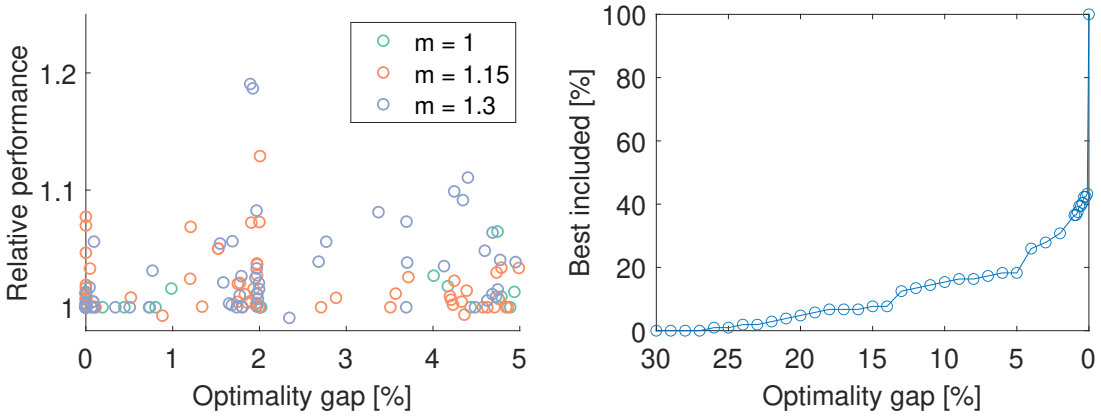


Figure 6.19: Within the 5% optimality gap no clear trend is visible. The cumulative percentage of best solutions for varying gaps shows that a significant amount of best solutions is obtained in the 1-5% range. This suggests solving allocation to optimality is not always crucial.

In Figure 6.19, a close-up is provided for the results within a 5% optimality gap, showing clearly such a guarantee does not exist, as a trend is less clear. In the right plot, the cumulative fraction of the total number of best solutions found is displayed for a decreasing optimality gap. Though most of the best solutions are found when the task allocation is optimal (around 60%), over 20% of the best solutions are found within the 1-5% range, indicating less correlation between optimizing the last 5% of task allocation and the resulting MARP performance. For gaps $\geq 5\%$ also best solutions are found, but these are scattered over a longer interval. Combined, this means that nearly half of the best solutions are found before task allocation is optimal. Furthermore, the trends from Figure 6.18 suggest that close to the optimal allocation, the results might not be best, but close to the best on average.

It can be concluded that, though convergence to optimality is slow using CCB, it quickly finds solution within a 5% objective gap from the optimal solution, which produces solutions for the MARP that are of acceptable performance. An optimal allocation could even result in a slightly worse MARP solution, as there is no guarantee for convergence when using top-down allocation for the MARP formulation.

Furthermore, suboptimal allocation can also imply a complete lack of optimization, such that overlapping parts are not deconflicted, and multiple agents might cover the same area. The performance of these worst-case solutions is displayed in Figure 6.20. For higher overlap, the cost of the worst-case solution increases rapidly. Therefore, a trade-off is present between the risk of a worst-case solution in case of no cooperation and the possible performance increase by cooperation on increasing overlap.

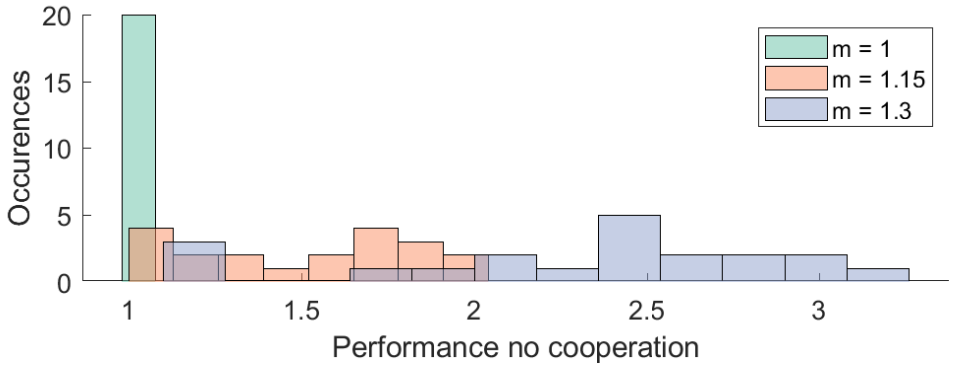


Figure 6.20: The relative performance of the worst-case allocation, which occurs when no cooperation is present. Multiple agents then perform any overlapping parts. Logically, increasing fuzziness shows worse performance.

6-2-4 Summary

In this section, the components have been analysed separately. First, benchmarks were assessed on computational performance and tightness. Secondly, the performance of task allocation is compared using these benchmarks. Lastly, distributed cooperation and specifically the performance of CCB was analysed.

Benchmarks

The lower bound obtained by column generation performs consistently for different numbers of sensors. The computation time is inversely dependent on the number of sensors, meaning that additional sensors lead to a decrease in computation time. Despite this, the computation time increases to prohibitive levels when increasing the number of sectors. In part, this can be attributed to the lack of improvements like bidirectional labeling, for which literature shows that much better results can be obtained. Still, as the SP is NP-hard, the approach remains limited.

To be able to obtain solutions to the separate SARPs even for larger problem instances, multiple heuristics are proposed. There is a clear relation between the optimality gap of the heuristic and the size of the SARP. As such, the trivial upper bounds can be expected to perform slightly worse, as the lack of balancing the SARP sizes results in worse allocation and worse approximation of the SARP cost by the heuristics.

Task allocation

Task allocation for a single supervisor is benchmarked against multiple trivial upper bounds, the (extrapolated) optimal value, and the prices and linearly relaxed lower bounds. The task allocation performs consistently better than the trivial allocation. However, only trivial clustering performs equally well for the tested problem sizes, which indicates that at least for these smaller instances the chosen clustering approach gives no added benefit. Based on the extrapolations, which seem to estimate the optimal cost reasonably consistent for different

sensor setups, the performance for task allocation decreases for larger problem instances. On the other hand, the gap with the priced lower bound decreases, suggesting the opposite. From a computational perspective, the performance relies heavily on the number of included sensors and less on the number of clusters. Lastly, also the added benefit of using the post-processing interpolation for clusters is tested. A significant reduction in performance can neither be observed for varying fuzziness, nor for scaled clustering weights of the sector coordinates, even for larger clusters.

Distributed cooperation

The novel CCB algorithm is compared with a centralized Gurobi implementation and a simulated version of ConcFB. Overall, CCB shows quick convergence to an initial solution that is within a 5% optimality gap. However, a slow tail convergence for larger problems can be observed. Compared to Gurobi, it performs comparably only during the initial fraction of a second. Thereafter, a consistent gap can be observed due to the slow tail convergence. Furthermore, Gurobi nearly always finds a first solution quicker, though in 30% of the cases CCB finds an initial solution that is better than the one Gurobi could find in the same time. Compared to ConcFB, CCB performs better, for which the effect grows for larger problem sizes and increasing fuzziness. When assessing the changes separately, the increase in performance can be contributed largely to proper initialization with the best-first strategy. Using the UB requests only has a minor negative effect, and removing the local tree even slightly improves the performance, which becomes more apparent for larger fuzziness. Possibly, this is due to the relaxation of the cooperative formulation not being tight enough.

Furthermore, the compiled domain size of each agent in the complete ordering is largely dependent on the ordering, as highlighted in Section 5-5-5. Experiments have been performed assessing a random, ascending or descending connectivity ordering (see Figure 5.8b). The results show that the ascending order performs best, the descending worst, and the random ordering in between. This can be attributed to the fact that the initial agent cannot be reordered. As such, if the first agent has the largest domain, no effective load balancing is possible when using multiple SPs.

Lastly, the effect of suboptimal allocation is assessed. It is found that there is a clear relation between suboptimal allocation and the performance of the resulting SARPs. However, this relationship is not so clear for solutions within the (roughly) 5% optimality gap. This is not surprising since the top-down task allocation approach provides no convergence guarantees whatsoever. Furthermore, in the case of no allocation, a clear performance decrease can be observed for larger fuzzifiers.

6-3 Combined performance

In this section, the performance of the complete method is assessed for a military environment specifically, which implies high dynamics and uncertainty (Section 1-2). In Section 5-1 three requirements are outlined to deal with these challenges: flexibility, robustness, and scalability. The solution method should have these characteristics while generating solutions of acceptable quality. First, the effect of the hierarchy and fuzziness on the overall performance is analysed. Then, each of the requirements is assessed.

6-3-1 Solution quality

First, the solution quality of the complete method is assessed. As a part of this, the effect of the hierarchy structure and the amount of cluster overlap are analysed separately.

Hierarchy structure

For any given problem with N sectors and P sensors, different kind of hierarchies can be constructed. As each supervisor has a group of subordinates, this group size is the main factor influencing the hierarchy structure. Examples are displayed in Figure 6.21. On the left, an asymmetrical hierarchy is shown. To the right of that, a symmetrical hierarchy is displayed with varying group sizes for each level. The third hierarchy is completely uniform, and the right is a 'centralized' hierarchy, with a single supervisor. In this thesis, the hierarchies are assumed symmetrical but not necessarily uniform.

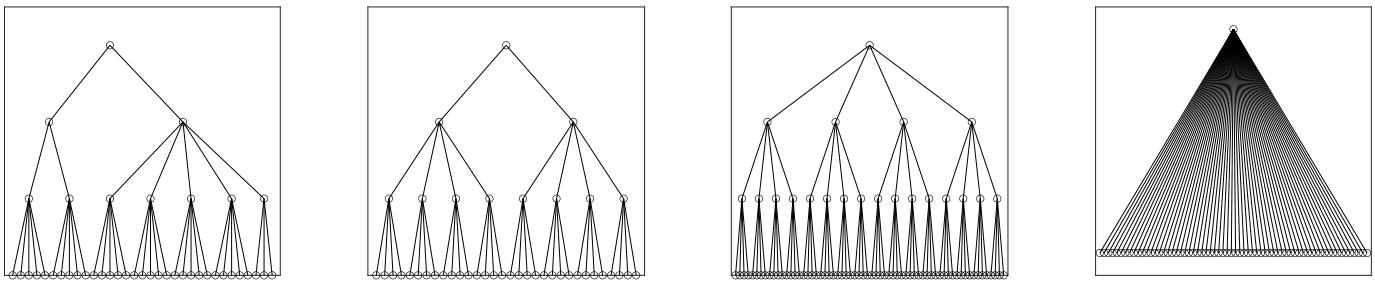


Figure 6.21: Different hierarchical structures, ranging from asymmetrical (first left), to symmetrical (second), uniform (third), and centralized (fourth).

To optimize the performance of this solution approach, the specific impact of the hierarchy structure is analysed. Multiple experiments are performed, with varying amounts of sensors within $P = \{8, 16, 32\}$. The number of sectors remains constant at $N = 150$. The group sizes are then varied at most between $\{2, 4, 8, 16, 32\}$, limited by the number of sensors used (corresponding to the centralized case from Figure 6.21). The results are displayed in Figure 6.22. Three plots are compared. On the left, the group size is compared with the relative cost. The central plot compares the group sizes against the total allocation times, and the right image shows the average optimality gap after the allocation time limit of 300s has been reached.

It is particularly striking that the computation time explodes when the group sizes approach the centralized case and much less because of the absolute group size itself. Furthermore, for the cases with $P = 16$ and $P = 32$ with group sizes equal to P, the time-out limit is reached every time, resulting in a significant optimality gap. For $P = 16$, the gap is on average 6%, which does not necessarily imply a much worse solution to the MARP (see Section 6-2-3). For the case of $P = 32$, the relative performance shows a clear increase compared to the other cases, which is most likely due to the average gap of 10%. Nonetheless, the results already indicate that increasing the group size is not profitable at all. Instead, the smallest group size of 2 is significantly better in all cases.

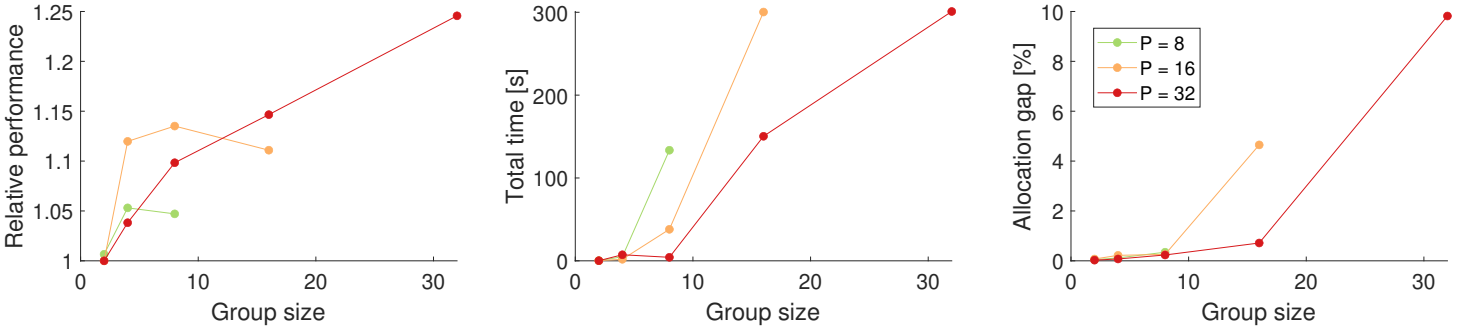


Figure 6.22: Left: the relative performance of different group sizes. Center: the total run time for different group sizes. Right: the resulting allocation gap after the time-out limit has been reached. The results are in clear favor of a group size of 2.

Furthermore, the consistency of the results is assessed for varying group sizes. This is necessary, as smaller sizes increase the height of the hierarchy, possibly compounding the effect of randomizations in the solution approach. Two elements are of influence here. First, the clustering can converge to a local optimum, depending on the clustering centroids. Second, if allocations are different, the usage of inexact heuristics can generate different results. The results are displayed in Figure 6.23. The results show that smaller problem sizes are more consistent, but that inconsistency of the results is a problem overall. This obfuscates the results from further experiments.

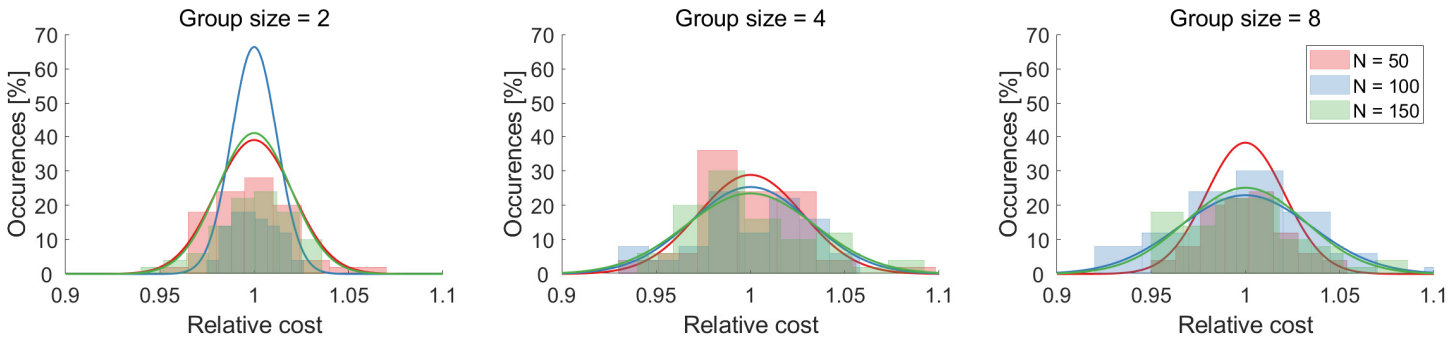


Figure 6.23: Consistency of the results for a varying problem and group size. Smaller group sizes are slightly more consistent, although all setups show large deviations.

Range of intent

By increasing cluster overlap, the solution quality is expected to improve at the cost of longer runtimes as the cooperative problem becomes larger, allowing more freedom in the allocation. To influence the cluster overlap, the fuzzifier m and the number of sectors are varied, as both these parameters have a clear relation with the resulting overlap.

An experiment is performed with $P = 16$ and a group size of two, leading to a hierarchy with five levels. The sectors are ranged between $N = \{50, 100, 150, 200\}$, and the fuzziness within $m = \{1, \ldots, 1.7\}$. The processing time and relative performance are displayed in Figure 6.24. It becomes clear that the processing time relies both heavily on the number of sectors and the fuzziness. Increasing the fuzziness clearly improves the objective value with around 10% on average. However, the positive effect flattens quickly. Therefore, as the computation time increases, it is best to keep the fuzziness as small as possible. This will also reduce the negative effect of the worst-case solution, as described in 6-2-3.

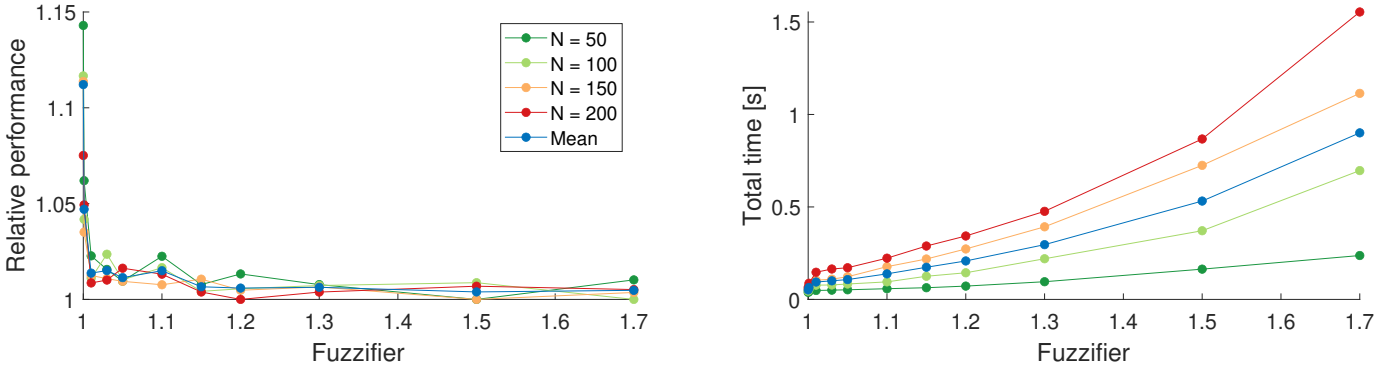


Figure 6.24: A plot showing the relative performance for different fuzziness and number of sectors, for a hierarchy with groups of 2 and 16 agents. Based on these results, it is best to keep the fuzziness as low as possible, around $m = 1.15$

Solution quality

In line with the experiment from Section 6-2-2, the performance of the complete distributed hierarchical setup is benchmarked. The priced lower bound, trivial upper bound, centralized task allocation, and optimal solution are all displayed in Figure 6.25. The results are displayed on a semi-log scale. Furthermore, the optimal solution is linearly extrapolated. The initial results show that the hierarchical setup performs slightly worse for smaller problems than the centralized case. However, this is not expected to be true for an increasing number of sectors, as shown in Figure 6.22. On average, there is a very consistent gap between the upper bound and the hierarchical solution, indicating that performance remains relatively constant. Though the gap with the extrapolated optimal solution increases, this linear extrapolation does not provide a strong argument. More specifically, when sectors are added, the cycle distance is increased on average, which increases the cost for all sectors in the sequence. This suggests a quadratic dependence, which means the linear interpolation underestimates the cost. In line with this argument, the gap with the priced lower bound also decreases, indicating consistent performance.

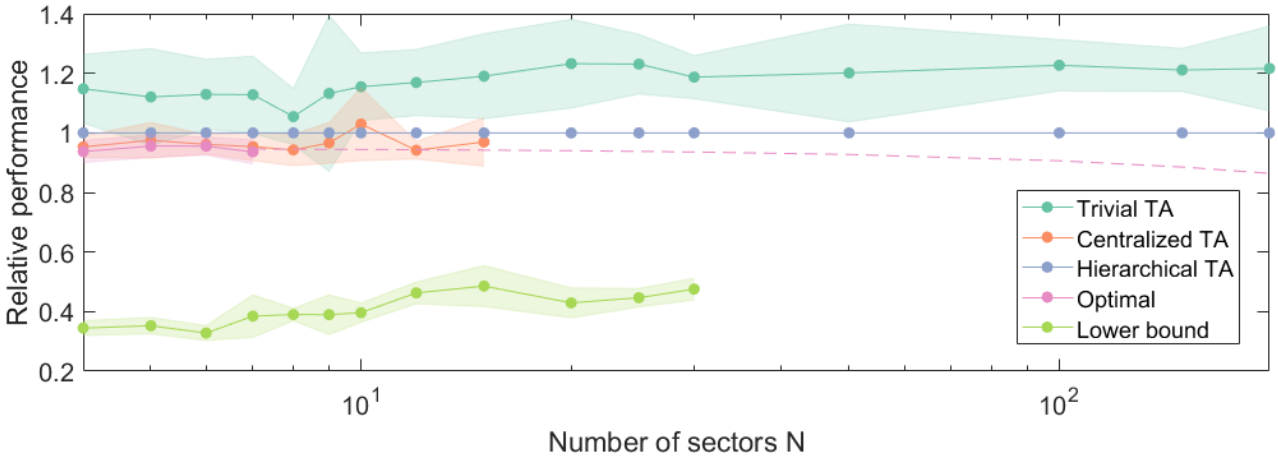


Figure 6.25: The performance of the hierarchical approach compared to the other benchmarks. The hierarchical approach shows consistent performance for an increasing number of sectors.

6-3-2 Flexibility

Flexibility implies that after changes in the environment have occurred, adjusting the solution locally already provides sufficient increase in solution quality. This means that the method is more adaptive for a broader range of unforeseen events, as the solution does not have to be revised as a whole immediately.

To assess this, a base solution is obtained for a given environment. Then, this original problem is perturbed, after which the solution is revised at different levels of the hierarchy. It is expected that the larger the perturbations, the higher in the hierarchy a revision needs to be performed. Perturbation are performed in three types: flipping an academic example, scattered, and concentrated perturbations.

Academic example

The first experiment performed is an academic example, for which there is a clear indication the solution needs to be revised after a fixed perturbation. It is composed of four unique planes, being each a combination of low or high risk and risk increase, based on specific values for priority P_i, traversability M_i, and uncertainty δ_i. The example is shown in Figure 6.26. As a perturbation, the map is flipped horizontally. This means agents will cover a suboptimal area, assuming the solution method made the best assignment originally.

The results of this perturbation are shown in Figure 6.27, for an experiment with $N = 100$ sectors, 300 simulations, 16 agents and a group size of 2. The hierarchy consists of 4 supervising levels, with the highest indicated as level 4. Four metrics are collected to show the performance. The *fraction of improvements* indicates the fraction of the total number of simulations where revising the solution from that hierarchy level improved the solution value. The *average improvement* indicates how much the solution value increases on average by revising the solution from that level. The *worst* and *best quarter occurrences* measure how often a solution from any of the hierarchy levels occurs in the worst and best 25% of

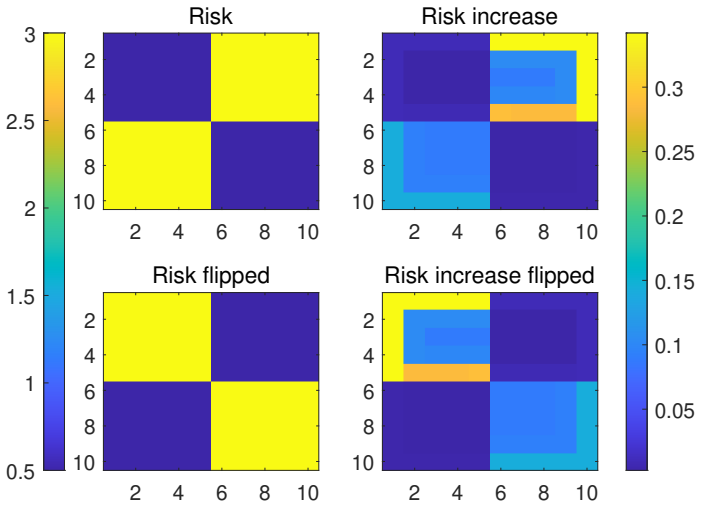


Figure 6.26: An academic example map, consisting of four distinct planes, each being a combination of high and low risk or risk increase.

all solutions respectively, as depicted in Figure 6.28. The average improvement also includes bars for the 95% confidence interval. It is clearly visible that revision at the lower two levels is not sufficient. However, revising the solution at level 4 instead of 3 yields no improvement or even a slight decrease.

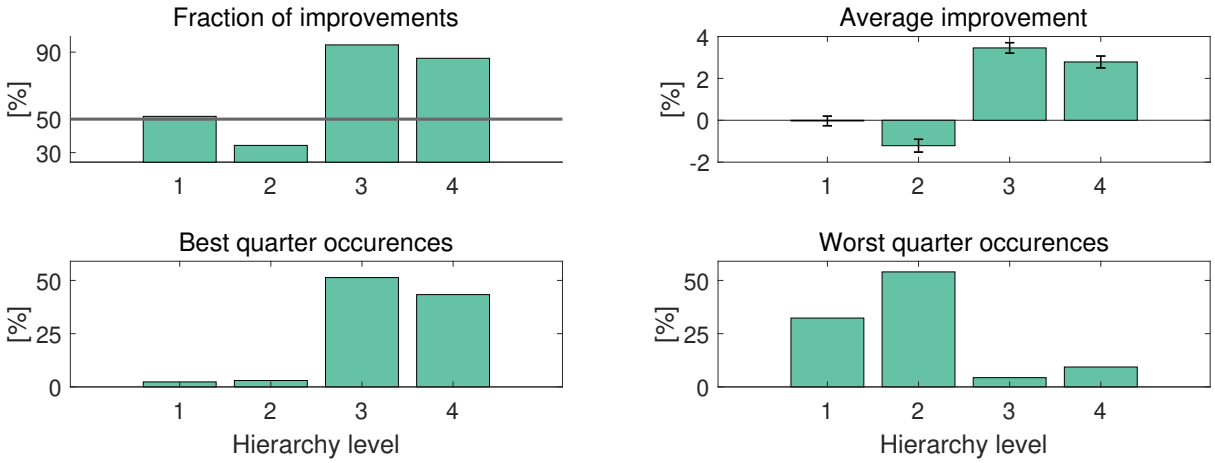


Figure 6.27: Metrics showing the effect of revising the solution at different hierarchy levels if the academic example is flipped. Clearly, improvement is only achieved when starting the revision at least on the third (second-highest) level.

The distribution of the average improvement is displayed in Figure 6.28. The performance of levels 3 and 4 is significantly better on average. However, on all levels the results vary, as more parameters influence the actual performance. For instance, the distribution of the heterogeneous capabilities varies in each simulation.

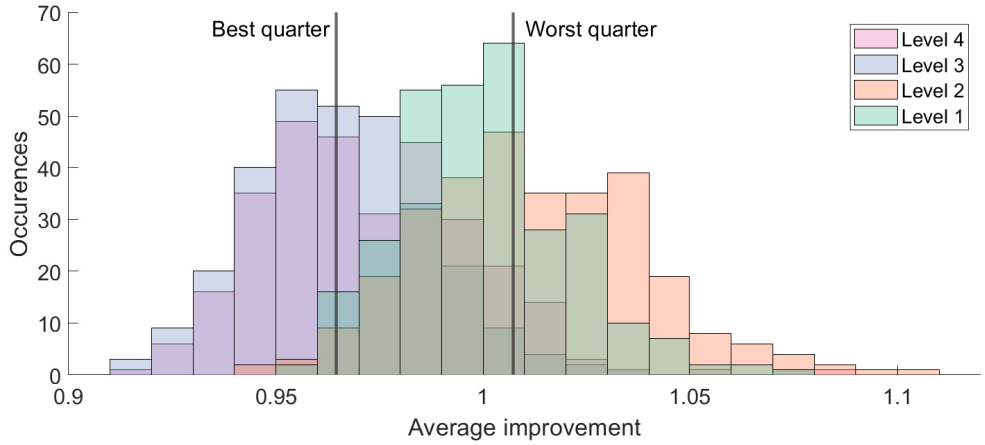


Figure 6.28: A distribution of the average improvement obtained for each level. The upper two levels (3 & 4), show consistently better results, but deviations remain large.

Scattered perturbations

Furthermore, perturbations are also performed on the original map. These are expressed as a percentage κ of the original map and completed as follows:

1. While not $\kappa\%$ changed of original map, select a random sector.
2. Collect all priorities P and uncertainty δ from all first and second-order neighbors.
3. Set new P and δ as any from these neighboring values, or a uniform value $\leq \max(P)$ or $\max(\delta)$ respectively.

These perturbations are displayed in Figure 6.29, which shows that the larger κ, the more sectors are changed. Furthermore, the sector values appear less structured. As the changes are random, they do not compare directly to, for example, a shifted focus to a parallel road.

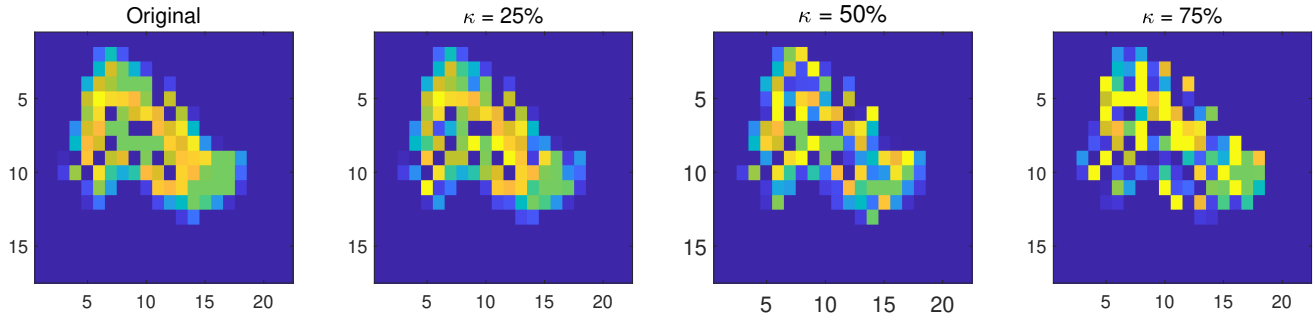


Figure 6.29: A display of the difference if perturbations are performed on the original problem for increasingly larger percentages κ.

An experiment is performed with 300 simulations for $\kappa = \{25\%, 50\%, 75\%\}$, and again 16 agents and a group size of 2. The same four metrics are collected but displayed in parallel for

all different sizes of κ in Figure 6.30. The results are less outspoken than with the academic example but consistent nonetheless. Clearly, for larger perturbations, the effect of redoing the solution for any level increases. The fraction of improvements and the average improvement are concentrated much more nominally. For the worst quarter, it seems the levels perform approximately equal. For the best quarter, the higher levels more often deliver the best-performing solutions. However, good solution are also obtained for lower levels, indicating that it cannot be assumed that redoing the solution at higher levels is always necessary.

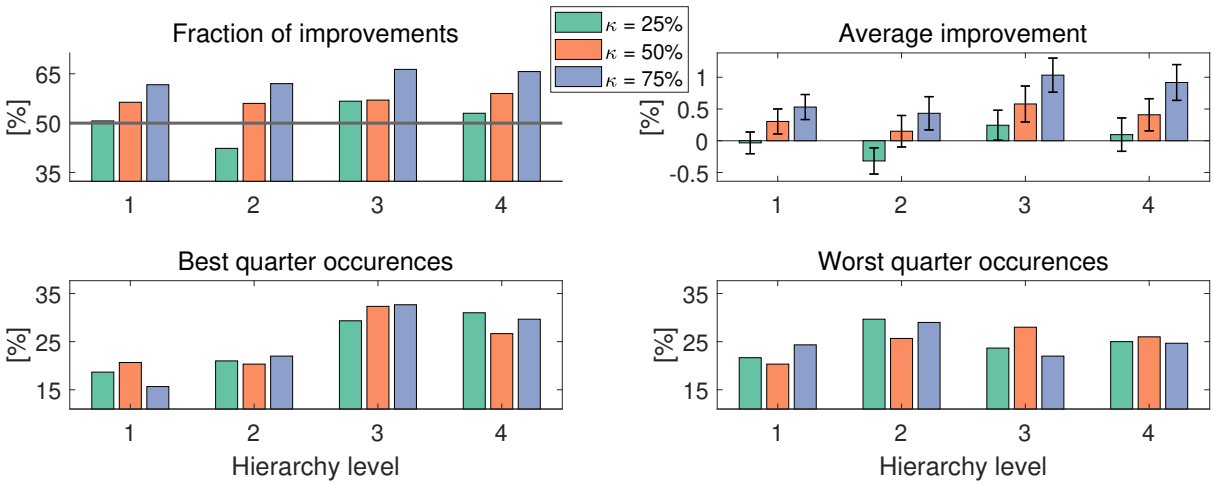


Figure 6.30: The results for varying perturbations. At every level, a larger perturbation leads to bigger improvements when revising the solution. Still, the best results are obtained when revising at the highest two levels.

Concentrated perturbations

In line with the previous experiment, also concentrated perturbations are considered. The key difference is that instead of randomly picking an unchanged node to perturb, the closest unchanged node by Euclidean distance is selected. By doing so, changes are concentrated in some part of the original problem. The difference is exemplified in Figure 6.31. It is clearly visible that for increasingly larger values of κ, the total revised area expands around the initial perturbation. In this example, the lower right corner remains untouched in all cases.

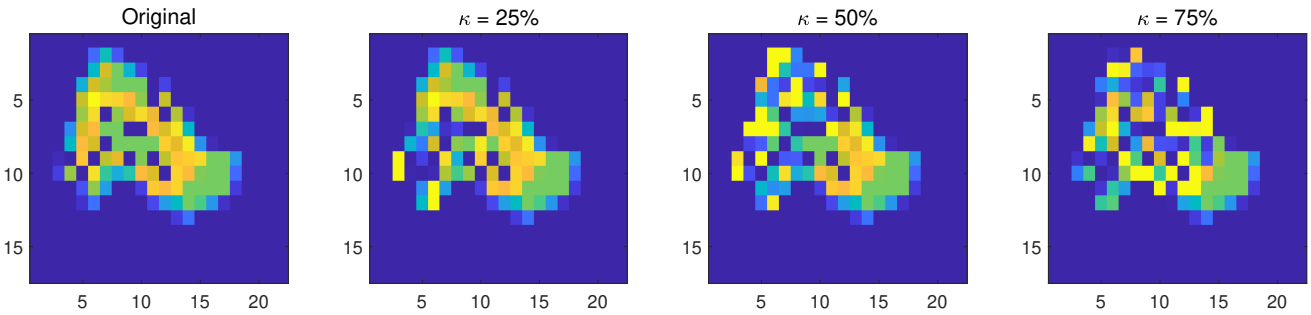


Figure 6.31: A display of the difference if perturbations are performed on the original problem for increasingly larger percentages κ of concentrated changes.

The results are displayed in Figure 6.32, and show similar behavior as with the scattered perturbations, though the difference of the average performance seems to be slightly larger between the lower and higher two levels with concentrated perturbations. This suggests that with concentrated changes, recalculating the solution at higher levels is more beneficial. This result seems counter-intuitive, as the locality of the changes suggests that a local revision is sufficient. A possibility is that as the changes are concentrated, this justifies relocating some assets at a higher level, as the changes are more prominent. In the scattered case, the changes are on average less radical, such that at a local level improvement is already possible.

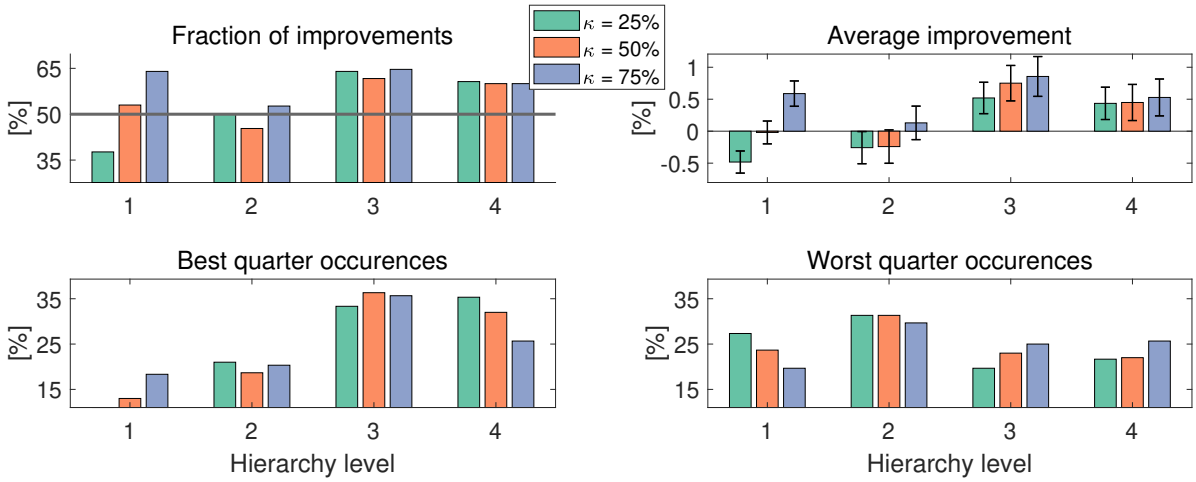


Figure 6.32: The results for varying concentrated perturbations. Compared to the scattered case, the lower two levels show slightly worse performance.

6-3-3 Robustness

The second requirement is robustness against unreliable communication. This implies that at any moment during cooperative optimization, the process can be preemptively terminated. In the worst case, cooperative optimization might not be possible at all.

In Section 6-2-3, an experiment is already dedicated to quantifying the relationship between a larger optimality gap of the cooperative formulation and the performance of the combined SARP solutions. However, in that case the effect of the hierarchy itself is not included. It is not unreasonable to believe that problems during optimization at different echelons in the hierarchy have a varying impact. Therefore, two experiments are performed, which are displayed in Figure 6.33. The first experiment introduces an optimality gap for all groups in a specific echelon (a level). The second experiment introduces an optimality gap for all supervisors and agents succeeding a level (a column).

Suboptimal level

The goal of this experiment is to quantify the effect of suboptimal allocation at different echelons. In reality, it can not be expected that a complete echelon will fail, but rather a specific group. However, for larger hierarchies the effect of a specific group on the complete solution is minor, such that it is chosen to allocate suboptimally in the complete level.

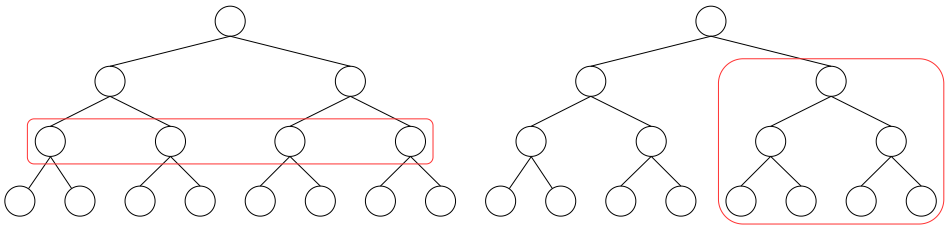


Figure 6.33: Left: suboptimal allocation for a complete echelon (a level). Right: suboptimal allocation for all supervisors and agents succeeding a superior (a column).

The experiment is performed for $P = 16$, a group size of 2, and $N = 100$. The allocation is allowed to terminate at $\{30\%, 15\%, 5\%\}$, and the worst-case solution is represented by a gap of ∞. A histogram of the results per hierarchy level is displayed in Figure 6.34. For the fourth level the results are normally concentrated around 1. This is to be expected, as no cooperative problem needs to be solved at the highest level. Furthermore, performance of suboptimal allocation at higher levels is better than at lower levels, though for the tails exactly the opposite effect is true. This indicates that lower levels are able to 'repair' the suboptimal allocation of a superior to some extent, which is not possible if at the lowest level the method terminates preemptively.

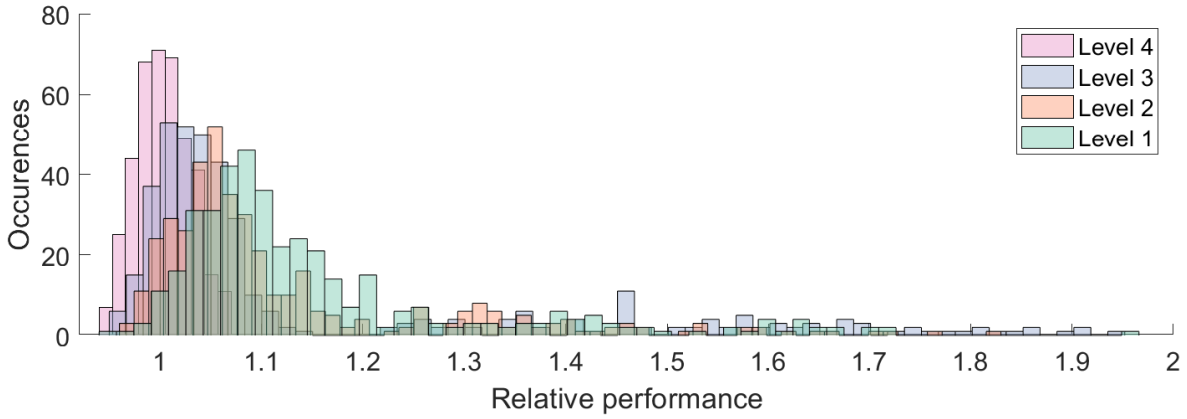


Figure 6.34: A distribution of the relative cost increase if the allocation is suboptimal at specific levels. On average, lower levels show a larger decrease but a smaller worst-case tail, implying that lower levels partially mitigate the negative effect of suboptimal allocation at higher levels.

The four metrics are displayed in Figure 6.35, which are split for each optimality gap and level. The gaps in the legend are the average gaps obtained for the provided stopping criterion. The fraction of deteriorating solutions shows that suboptimal allocation at the lowest level nearly always results in a worse solution, even for lower gaps. This fraction decreases slightly for higher levels. For the highest level there is no effect, as in roughly 50% cases it performs worse due to the nominal deviations in the solution method itself. Clearly, the average decrease in performance is much larger in case no cooperation is present at all. This effect increases for higher levels as the resulting overlap becomes greater. The worst and best quarter occurrences support this. On average, suboptimal allocation at lower levels has a higher effect. In case of no cooperation, the result is exactly the opposite.

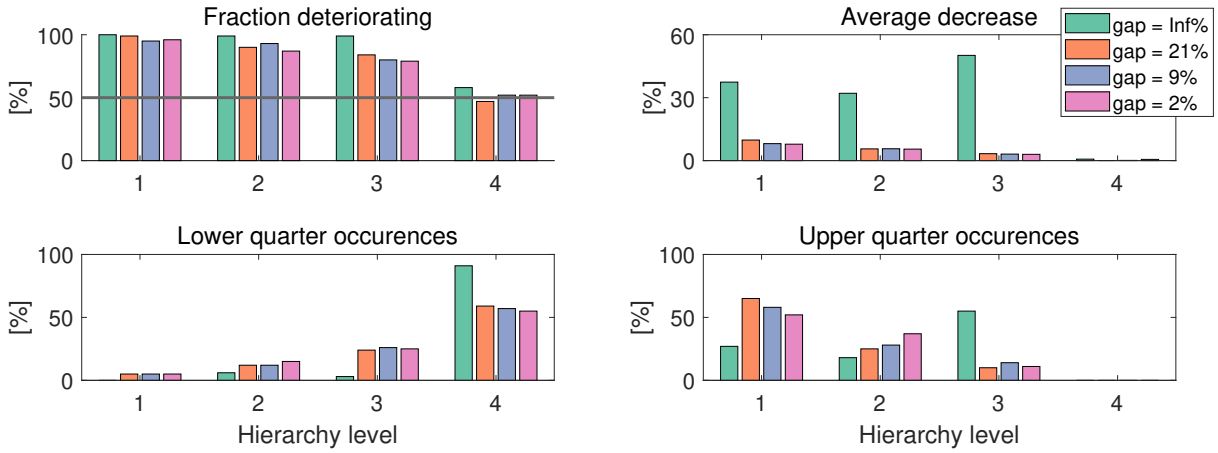


Figure 6.35: The metrics are split per level and optimality gap. The highest level is not influenced, as no cooperation is present. The worst-case no cooperation has the largest impact at the second-highest level. For suboptimal allocation the worst results are obtained at the lowest level.

Suboptimal columns

In this experiment, the suboptimal allocation starts at a specific level, which is continued on all succeeding levels. Hence, the name column, as displayed in Figure 6.33. In contrast to this figure, the effect is applied for a whole level downwards and not a specific supervisor. Though the depicted example is a more likely representation of, for instance, a regional blackout, it is chosen to include all supervisors in each level to enlarge the effect for measurement purposes.

The distribution of the results per level downwards are displayed in Figure 6.36. Compared to the results for a specific level in Figure 6.34, the tail for the worst-case solutions is much longer. This implies that the reduction in performance compounds over an increasing number of levels. Furthermore, the distributions for suboptimal allocation show no difference when repeated over multiple layers, suggesting that performance on the lowest level is dominant.

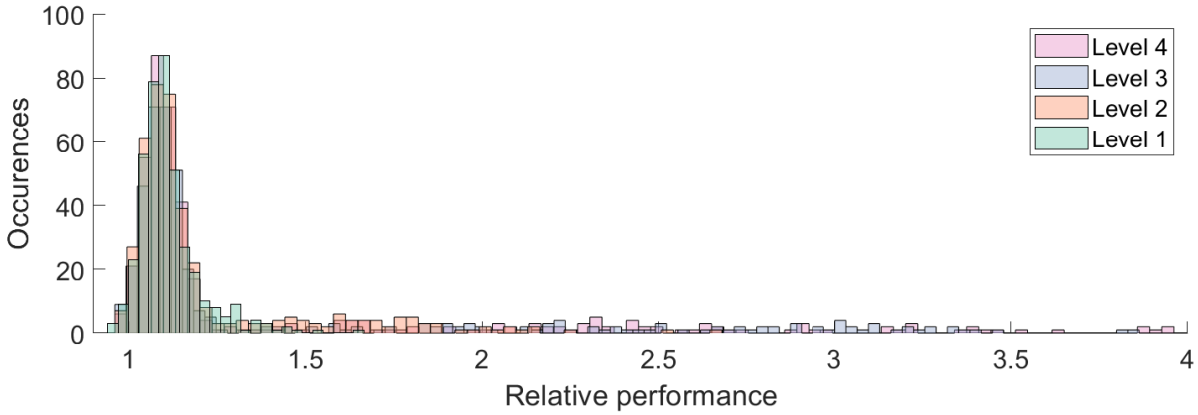


Figure 6.36: A distribution of the relative increase in cost for suboptimal allocation in a column. Performance of suboptimal allocation is equal at all levels. The worst-case solution compounds over multiple levels, generating longer tails for higher levels.

The results are further supported by the metrics in Figure 6.37. The average decrease shows a very clear compounded effect of multi-layer suboptimal allocation. Furthermore, in nearly all cases it leads to worse allocation for the MARP, as the fraction is approximately 100% for all levels and gaps. For the worst-case performance, no allocation starting from high levels is worse than only on lower levels, as more overlap is produced. However, for an optimality gap, this is not the case. Instead, performance appears relatively constant. This implies that the performance decrease is largely dependent on the success of optimization on lower levels. This is an interesting property, which indicates that the solution approach shows robust features against a high-level suboptimal allocation.

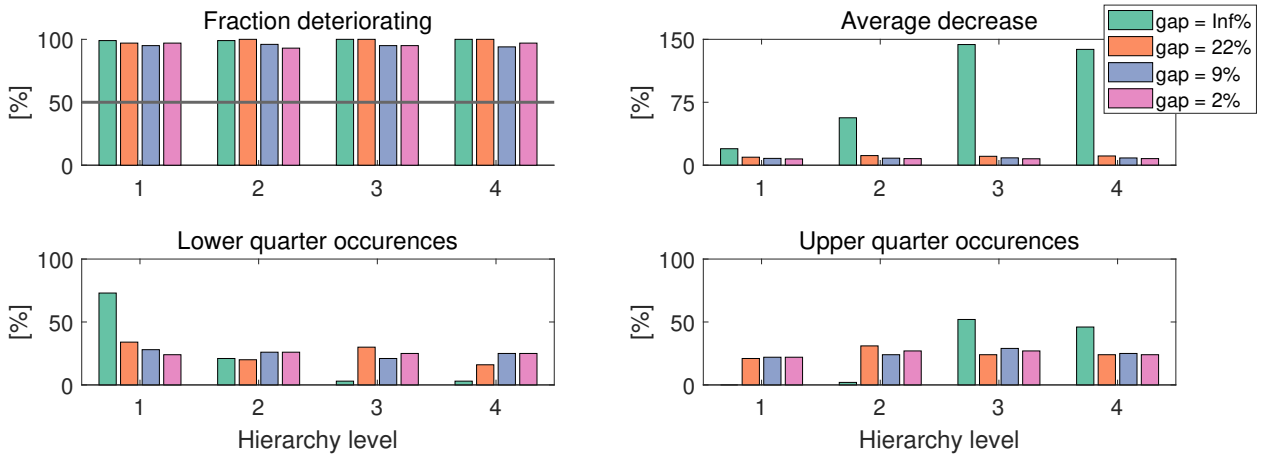


Figure 6.37: For the suboptimal columns, the average decrease in worst-case performance is significantly larger, with almost 150% reduction. The suboptimal allocation, however, seems constant over multiple levels. This shows robustness against communication failures at higher levels.

6-3-4 Scalability

Neither the addition of extra agents nor the increase in environment size should lead to an intractable problem, such that the flexibility and robustness properties can be maintained. The proposed hierarchy is expected to perform better based on two key features. First, the hierarchy naturally exploits parallelism, as the columns of the hierarchy can perform calculations simultaneously. Therefore, the total execution time is equal to the maximum time it takes to perform the sequential calculations from the root supervisor to any of the subordinate agents. Furthermore, the scalability is expected to benefit from the constant group size. This implies that for each task allocation problem being solved, the complexity is approximately fixed. This is true as the group size is predefined, and the number of clusters can be controlled directly.

The experiment is performed for $N = \{5, \dots, 1000\}$, and $P = \{4, 8, 16, 32\}$. The group size is fixed to 2. The results are compared between a single task allocation problem and the complete hierarchy. In Figure 6.38 a comparison is made for varying sectors, displayed in a log-log plot. On the left, it can be seen that for a larger amount of sensors, the calculation time grows exponentially for the number of sectors. On the right, it is clearly visible that the hierarchical approach shows (sub)linear growth instead.

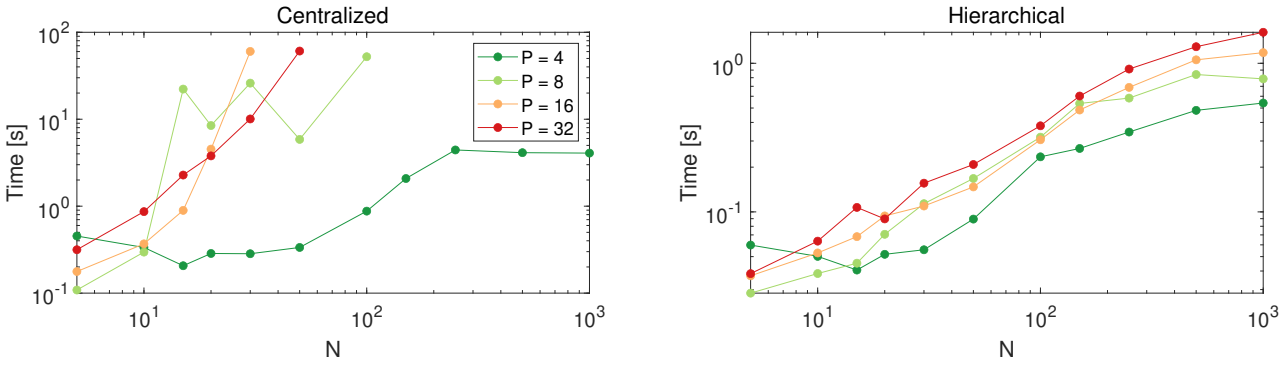


Figure 6.38: Log-log plots showing the growth in computation time for the centralized and hierarchical setup. In contrast to the centralized approach, the hierarchy shows the desired linear increase in computation time.

In Figure 6.39 the same results are displayed for increasing the number of sensors. On the left, the results for the centralized case are displayed on a semi-log plot, as the processing time varied by large amounts depending on the setup. This is most probably due to the non-linear formula for setting the number of clusters. On the right, the results for the hierarchical setup are displayed. There is a slight sublinear growth in the number of sensors, which becomes larger when more sectors are added.

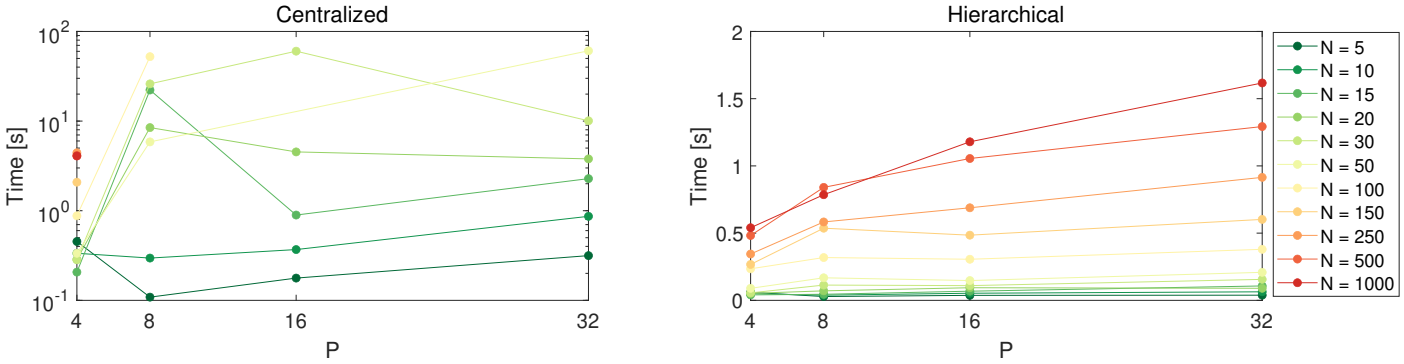


Figure 6.39: Due to large deviations in computation time, the results for the centralized case are displayed using a semi-log plot, while the hierarchy is shown on a regular scale. The centralized case increases much for more than four sensors and varies more greatly. The hierarchy is both fast and more consistent.

The hierarchy shows consistently quicker execution time, and the computation time is much more reliable in the number of sectors and sensors. Therefore, it can be concluded that the computation time of the hierarchy is more stable, quicker overall, and with a sublinear growth in the number of sectors.

6-3-5 Summary

In this section, the combined performance of the distributed, hierarchical intent-based coordination has been assessed. First, the performance of the hierarchy is analysed by evaluating the group size, the fuzzifier, and the performance compared to the benchmarks. Secondly, the flexibility is measured for an academic example, scattered and concentrated perturbations. Thirdly, the robustness against failing communication is analysed by measuring the effect of suboptimal allocation or the complete lack thereof. Lastly, the scalability is measured for increasing sectors and sensors.

Hierarchy performance

The analysis on the group size showed that a smaller group size of 2 significantly outperformed larger groups, both in solution quality and computation time. Furthermore, it is noted that due to the hierarchical structure the deviations in solutions for equal problems vary significantly, which influences the consistency of the experimental results.

Combined with varying fuzziness, it is found that a fuzzifier $m > 1$ returns better results, but that the benefit diminishes quickly. However, computation time keeps increasing, such that the m is kept as small as possible, resulting in $m = 1.15$.

The overall performance of the hierarchy is compared with the available benchmarks. For smaller problems, the distributed hierarchical setup performs slightly worse than the centralized allocation with one supervisor, which seems to contradict the result that smaller groups are better than the centralized case. The difference lies in the included number of sectors for both experiments, which suggests that for more sectors, the importance of smaller groups becomes larger. The results are very consistent compared to the trivial allocation, but seem to diverge from the extrapolated optimal solution slightly. However, this provides no strong argument as the cost is not expected to increase linearly. In contrast, the gap with the lower bound becomes smaller for increasing sectors.

Flexibility

When analysing the flexibility for the academic example, it is expected that only higher levels should return better results. This is clearly visible, confirming the validity of the experiments. However, though the benefit is visible for reoptimizing at the highest two levels, the average improvement is not large in an absolute sense, being around 3%. This could be attributed to the large deviations in results for hierarchical setups.

For scattered perturbations of the scenario the results are split per level and amount of perturbations. The larger the perturbations, the larger the benefit of revising the solution at any level. Though more often revision at the highest two levels provides the best solutions, the lower levels frequently generate it too. Also, the average improvement varies much less, except for the 75% perturbation case. This implies that revising the solution at the highest level is not always necessary.

The concentrated perturbations show comparable results, though revising the solution at higher levels seems slightly more profitable. Counter-intuitively, larger perturbations in a concentrated area suggests that relocating assets at a higher level is more beneficial.

Robustness

Robustness is assessed in two ways. First, suboptimal allocation is enforced at a specific level. Secondly, the suboptimal allocation is enforced from a level downwards to the agents. In the first experiment, suboptimal allocation at lower levels appears to have a significantly larger effect. For the tails generated by the worst-case scenario of no cooperation, the effect is reversed. This suggests that suboptimal allocation at higher levels can be partially mitigated by lower levels.

When suboptimal allocation is applied downwards to all succeeding levels, no difference can be observed between the initial suboptimal levels. This implies the performance of the lowest level is dominant, which must always allocate suboptimally in this experiment. However, the effect of no cooperation compounds over multiple levels, creating longer tails for increasingly higher levels.

Scalability

The scalability is tested for varying numbers of sectors and sensors. It has been shown that the computation time for the centralized case increases exponentially for an increasing number of sectors. In contrast, for the hierarchical case the growth in computation time seems sublinear. This can be attributed to smaller problems in each group and the natural exploitation of parallelism as groups can perform optimization independently of each other. Furthermore, though the computation time seems to increase on average for $P > 4$ for the centralized case, there is no clear trend visible. This makes the computation time unreliable and hard to predict. The hierarchical setup shows a very consistent increase for an increasing amount of sensors and sectors, which is a beneficial property.

Chapter 7

Conclusion

Despite advancements made on Robotic & Autonomous Systems (RAS) in civil industries and a large amount of literature produced on algorithms for military purposes, implementations are still lacking. The main reason for this is that state-of-the-art algorithms deal insufficiently with the highly dynamic and uncertain characteristics of a military environment, leaving room for improvement. For this thesis, the application is limited to persistent reconnaissance with heterogeneous Unmanned Aerial Vehicles (UAVs), leading to the research question:

Can a solution method be developed for persistent reconnaissance that produces consistently good results in a highly dynamic environment with unreliable communication?

It is hypothesized that intent-based Command & Control (C2), as in use with the military today, provides a possible solution approach to mitigate the problems posed by this complex environment. The challenge is how these communicative principles can be converted to a mathematical approach applicable to RAS.

As the underlying model, the Multi-Agent Reconnaissance Problem (MARP) is formulated as a compact combination of the frequency and coverage level approach for persistent reconnaissance with heterogeneous agents (Chapter 3). Intent-based coordination is proposed as a method to partition the Area of Interest (AOI) for the MARP into disjoint subsets, such that independent Single-Agent Reconnaissance Problems (SARPs) can be solved separately. To ensure that this method can deal with a dynamic and uncertain environment, it should meet three requirements: 1) it needs the flexibility to adjust solutions locally, 2) it must be robust against unreliable communication, and 3) it needs to be scalable.

These requirements lead to a distributed hierarchy with multiple groups of supervisors at each level. Each supervisor performs clustering and subsequent task allocation, and each group can perform distributed cooperation to resolve shared tasks (Chapter 5). Complex Concurrent Bounding (CCB) is proposed to perform distributed cooperation, which is an adjusted version of ConcFB for complex local problems. When the subsets for the disjoint SARPs have been obtained, they can be assessed using the benchmarks from Chapter 4. A lower bound is discussed based on the pricing step of branch & price, and a method is provided to generate a trivial allocation for an upper bound. Lastly, heuristics are provided to solve the SARPs.

Experiments are performed on an example scenario in Chapter 6. The separate components have been analysed on solution quality and computational efficiency first. The results

show that the priced lower bound performs consistently relative to the optimal solution, though computational performance remains limited due to the Sub Problem (SP) complexity. Furthermore, CCB shows a clear performance increase compared to Concurrent Forward-Bounding (ConcFB), where especially best-first strategy makes a major contribution. The addition of a local tree did not have a significant effect, though this might be attributed to insufficient tightness of the underlying formulation. Compared to the centralized solver Gurobi, CCB shows slightly worse performance initially, but mainly longer tails to convergence. This, however, is not surprising compared to a state-of-the-art solver. Moreover, the added benefit of using a distributed algorithm lies in the removal of a single bottleneck, such that problems can still be solved by a part of the supervisors if connectivity issues occur. When the components are combined in the hierarchy, groups of two supervisors are shown to generate the best results in the least time. Furthermore, the fuzziness improves performance against a higher computational cost. However, the added benefit of increasing fuzziness diminishes quickly, such that small values are sufficient and quick computations remain viable.

The high dynamics are simulated by perturbing the environment. By revising the solution at different levels and comparing the results, the difference in solution quality can be observed. The results show that a revision of the solution at the highest levels generates better solutions on average – though by a minor percentage only – but certainly not exclusively. This indicates that it is not required to constantly revise the complete solution, or at least not initially, which implies a degree of flexibility. Furthermore, the results indicate that the revision at higher levels becomes more profitable for more concentrated changes, most probably as more suitable assets become available for the drastic but local changes.

Robustness against unreliable communication is tested by applying suboptimal allocation to one or multiple levels. The results are clear: suboptimal allocation at the lower levels has a larger effect on the performance. This means that at higher levels, this effect is partially mitigated by corrective allocation at lower levels. This is a very promising feature. Furthermore, though the worst-case effect of no cooperation compounds over multiple levels, this is not necessarily a problem. First, it is unlikely that no cooperation is possible at all, but a connection exists with the subordinates. Secondly, the flexibility experiments show that the effect of not updating the subsets is smaller than updating with worst-case subsets. Therefore, it is simply better to only update the solution if cooperation is possible. Thirdly, the worst-case solution naively sums the cost for sequences that overlap, which is not an accurate representation anymore, given that the (nonlinear) cycle distances created by overlapping sequences should lower the actual cost of the worst-case outcomes.

Lastly, the computation time scales sublinear in the number of sectors, allowing for large problem instances. The computation time increases gradually for more sensors, but the rate with which the computation time increases for more sectors is not jointly dependent on the number of sensors. This indicates that the approach is scalable, and that also for realistic problems consistent results can be obtained quickly.

Concluding, the usage of intent-based coordination appears to contain the desired properties of flexibility, robustness, and scalability while providing solutions that are reasonably close to the optimal value. Though further research is necessary to validate the results in dynamic time-extended simulations, this research provides a strong argument for including a form of intent in a distributed hierarchical setup for highly dynamic and uncertain environments.

Chapter 8

Discussion

The approach outlined in this thesis consists of many components. The main goal was binding them in a unified framework based on intent-based Command & Control (C2). As such, not all components are at the cutting-edge of the current state of the art, leaving areas for improvement. Therefore, different improvements and suggestions for future work are discussed, including the outline of an alternative solution approach with convergence guarantees.

Modelling persistent reconnaissance The Multi-Agent Reconnaissance Problem (MARP) has been proposed as a suitable model for persistent reconnaissance. Though this formulation is compact, it does not explicitly include capacity. As the cycle distance might not be a sufficient metric for realistic recharging and capacity constraints, extending it to a more complex problem could prove valuable. Especially in a dynamic environment, this capacity could be dependent on multiple (mobile) recharging points. Furthermore, the model assumes a fixed value for risk each iteration. In line with other Intelligence, Surveillance & Reconnaissance (ISR) models, also a Partially Observable Markov Decision Process (POMDP) can be formulated, which includes a form of transition probabilities to include the effect of the agent on its environment during optimization. This could generate more useful results, but makes it more complex. Lastly, as already mentioned in the conclusion, no method is included to assess the effect of overlapping sequences. Multiple sequences result in varying (nonlinear) cycle times, making it difficult to determine the actual cost: what is the difference between a sector that is visited twice within 10 minutes, and then 50 minutes not, compared to being visited exactly once every half hour?

Column generation The column generation procedure still shows limited performance. As also summarized in Section 4-1-2, some concrete improvements are possible to improve the performance of the column generation. The most prominent is the extension toward bi-directional labeling [87] and decremental state-space relaxation [88], but also the heuristics can be improved, or forward labeling can be used in a relaxed version first. Furthermore, the tightness of the lower bound can be improved by applying a Dantzig-Wolfe reformulation on the MARP, instead of a set covering reformulation [114].

Task allocation The speed with which task allocation can be solved can be improved by removing the standard linearization and using a better alternative. Semidefinite Programming (SDP) might prove a useful strategy, which requires the dependency matrix to be positive definite, which can be achieved by creating a diagonal dominant reformulation. However, this is no trivial operation for which specifically designed operations exist [94, 95]. Alternatively, other more efficient linearizations can be applied [96, 101, 102]. An additional benefit is that the task allocation can be solved more quickly and that the cooperative allocation relaxations become tighter, as they are based on the same quadratic constraint. As such, Complex Concurrent Bounding (CCB) might be able to prune much larger parts of the local search tree, improving the solution speed to the cooperative problem too. Lastly, the task allocation does not include the current position of the agents or their distance towards a recharging location. This means that in a time-extended formulation, highly suboptimal allocations can occur, as agents have to relocate large distances before being able to perform any reconnaissance in their assigned subset. As such, robustness over time is not guaranteed. It might prove valuable to perform an analysis of the added cost by reassigning subsets and how potentially some type of robust assignment could reduce these effects.

Additionally, the concept of intent can also be applied in the objective function of task allocation. Currently, the overlap is only used as constraint, as it outlines which tasks are accessible and should be deconflicted. However, the degrees in the subset of each supervisor can also be used to measure how much they overlap, which can be implemented as a relative measure in the objective function. By doing so, tasks that overlap only by a minor degree are less likely to be assigned to a subordinate of that superior. Currently, any overlap is treated equally, which undoubtedly influenced the performance.

Practical constraints During these simulations, the distributed problems are all solved in a distributed fashion on a single computer. Therefore, it overcomes any issues that arise from practical distributed implementations. In reality, bandwidth is limited, package drops may occur, and additional bargaining or communication algorithms might need to be implemented to ensure the algorithms remain functional, potentially worsening performance. These practical setups also give a chance to include additional layers of logic that can be used in more demanding environments. For instance, if there is a connection loss within groups, agents might relocate to try to reconnect or reorganize in smaller groups locally.

Alternative approach During the analysis in this thesis, it became apparent that it is difficult to perform a definitive assessment, both practical and theoretical, on the performance of intent-based coordination. This is due to the large deviations in the results, which are attributed to the many stochastic elements in the combined approach and the lack of any convergence guarantees. Therefore, an argument can be made to choose another approach that might not have the best performance but generates more consistent results. By doing so, a better picture can be constructed of the intricacies of the solution method, forming a better basis for future developments.

An obvious choice would be to extend the priced lower bound with a branching step, such that branch & price is fully implemented, which converges to the optimal solution. As this approach is not scalable, it is interesting to see if intermediate fractional solutions are of any use. In Appendix B an experiment is discussed with a provisional allocation based on the relaxed

solution obtained for the lower bound. The fractional solution from the relaxed Restricted Master Problem (RMP) is converted to a feasible integer solution, which is compared to the task allocation. The initial converted solution already performs quite comparable, though for larger problems the deviations are large. This indicates that if branch & price is used, intermediate solutions might already provide acceptable performance. In line with this argument, possibly also heuristical variants or a relaxed version of the RMP can generate a preliminary feasible integer solution.

One could argue that two main properties of the originally proposed method are lost. First, this branch & price approach is not distributed, introducing a single bottleneck. Secondly, even for the heuristical approach it cannot be expected that the computations scale linearly, as no hierarchy is used. Fundamentally, this is true, but the method can be converted to fit within the distributed hierarchy with intent-based cooperation. It is crucial to include clustering in a hierarchy. Otherwise, the method will not be scalable.

Fortunately, the obtained fractional allocation already suggests which cluster is appropriate for a subordinate. Depending on the amount of partially allocated sectors to an agent, the clusters can be combined and passed down as subsets, comparable to the current implementation. The difference is that, while branch & price is running, the clusters can be adjusted based on the generated fractional solutions, which can be reallocated and passed down as updated subsets. By doing so, the solution would converge to clusters that consist solely of the optimal set of sectors, such that guaranteed convergence becomes possible. Again, using the fact that each pricing step generates fractional allocations at first, a concept of intent is also immediately retrieved. Each cluster that still contains sectors assigned to multiple subordinates can be considered a shared task. This preserves the property that lower levels in the hierarchy can correct the suboptimal allocation of higher levels.

To summarize, depending on the fractional solutions obtained by branch & price, clusters can be labeled as a local or shared task. The clusters can continuously be adjusted based on improved intermediate solutions. After time, the clusters become perfect, and the subsets purely local. Then it has converged to the optimal solution.

The merit of this approach would mainly be that a guaranteed convergence allows for more consistent results. As such, an analysis can be performed that is more detailed and thorough. This could lead to a better understanding of intent-based coordination in general, and substantiate new improvements.

Appendix A

Parameter Evaluation

The component (Section 6-2) and complete analysis (Section 6-3) are based on the default parameter setup provided in Section 6-1-2. These parameters are chosen carefully based on the experiments performed below. As overlapping clusters represent the concept of intent, all parameters influencing the amount of overlap are quantified first. Secondly, the effect of varying the overtime penalty and relative capacities in the task allocation formulation from Section 5-4 is measured. Thirdly, it is measured if varying the clustering weights can be used as a form of capability-based clustering, such that the solution better accommodates larger differences in sensor heterogeneity, as hypothesized in Section 5-3-1.

A-1 Influencing cluster overlap

If clusters overlap, they are considered shared clusters and are allocated during distributed cooperation (See Section 5-5-1). As found in Section 6-2-3, larger overlap can lead to worse solutions, but it can also improve the solution if cooperation is possible, shown in Section 6-3-1. Therefore, it is clear that the amount of overlap has a great influence on the solution quality. Because of this influence, it is desirable to tightly set the number of overlapping clusters. There are four parameters influencing the amount of overlap: 1) the fuzzifier m, 2) the post-processing method, 3) the cluster sparseness, and 4) the cluster size. For each of these parameters, it is assessed what the influence is on the cluster overlap.

A-1-1 Fuzzifier

First, the fuzzifier is tested for the values ranging from $m = \{1 \dots 1.3\}$. This setup has $N = 200$ sectors, and $M = 10$ clusters. Each setup is simulated five times and averaged to account for the random initialization of the cluster centroids. The average amount of overlaps any cluster has with other clusters is displayed in Figure A.1. In the left plot, it is clear that all the clusters overlap with each other for a fuzzifier $m \geq 1.2$. For $m = 1$, clusters overlap only with themselves, as Fuzzy C-Means (FCM) converges to hard clustering. In the right picture, an indication of the overlap distribution is given for three fuzzifier values. Though the average number of overlaps for fuzzifier $m = 1.02$ lies around 3, the distribution includes clusters with an overlap ranging from 1 up to 6.

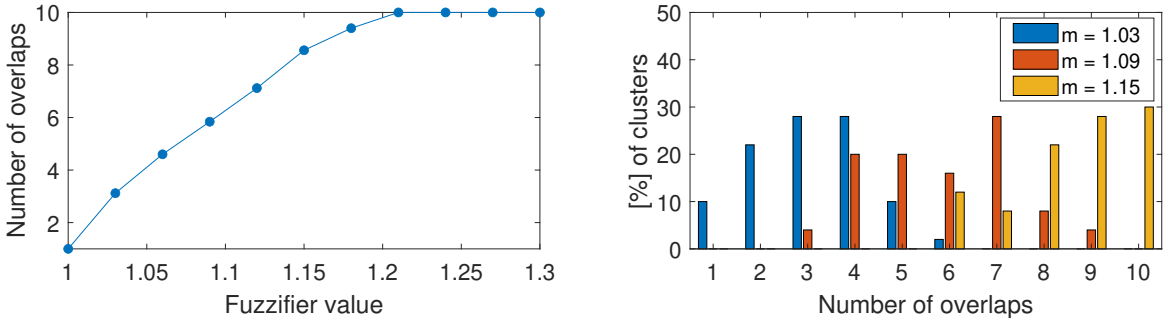


Figure A.1: A comparison of the average number of other clusters that any cluster overlaps with, for varying fuzzifiers m. In this case, all clusters overlap for $m \geq 1.2$.

A-1-2 Post-processing method

Next, also the post-processing method is considered. Two specific factors are of influence. First, this method interpolates cluster degrees which can result in extra overlap between clusters. Secondly, the cut-off threshold β removes low degrees, lowering the resulting overlap. To effectively measure the combined effect of interpolation, it is compared with the *cleaned degree*, meaning the original degree for which any degree $u_{ij} < \beta$ is also removed. The experiment is again performed five times and averaged for the same setup with $N = 200$ and $M = 10$. The fuzzifier is tested for the range $m = \{1 \dots 2\}$, and the threshold over $\beta = \{0.01 \dots 0.2\}$. In Figure A.2 the results are displayed.

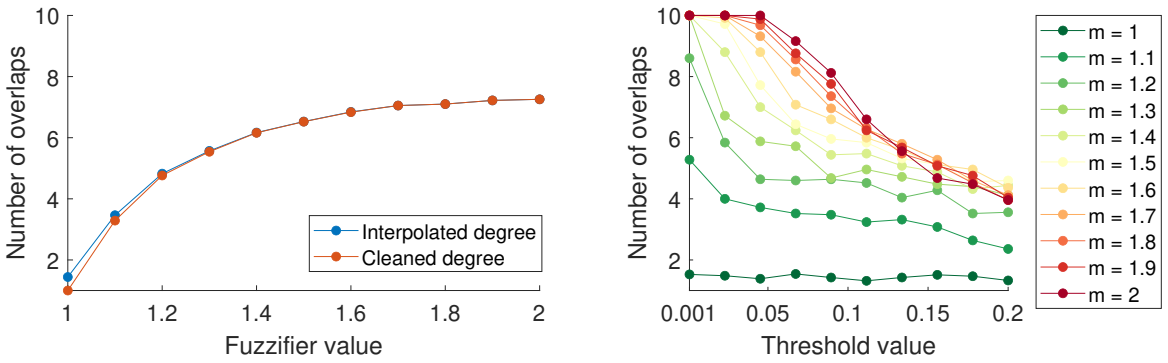


Figure A.2: Left: a comparison of the overlap for varying fuzziness for both the cleaned and interpolated degree, which are cut-off and averaged over a range of threshold β. Right: The combined effect of the fuzzifier and the threshold for the interpolated degree. The higher the threshold, the lower the overlap. This effect increases for larger fuzziness.

In the left image, the interpolated degree is compared with the cleaned degree. The mean increase in overlapping clusters is 4.8% by interpolating the degrees, for which the effect diminishes for higher fuzziness. Furthermore, it can be seen that even for $m = 1$, no hard clusters are retrieved, as the interpolation still generates overlap. The cleaned degree results in significantly fewer overlaps than in Figure A.1, where no cut-off is applied. This suggests the threshold value has a major impact on the overlap. The right plot from Figure A.2 confirms this. Here, the combined effect of the fuzziness and threshold is depicted for the interpolated degree. The higher the threshold value, the lower the number of overlaps. However, the effect

becomes larger for increasing fuzziness. For initial hard clusters ($m = 1$), the threshold does not affect the overlap after interpolation, as no degree is initially removed.

A-1-3 Cluster sparseness

Thirdly, it should be noted that the results obtained in the previous experiment are strongly related to cluster sparseness. The cluster sparseness means how much the sectors within each cluster are scattered physically, as shown in Figure 5.5. By changing the clustering weights, the relevance of the Cartesian distance between the sector coordinates is decreased, resulting in a visually sparse cluster. This sparseness, in turn, is diminished by the post-processing method. Consequentially, the effect of post-processing on cluster overlap becomes larger.

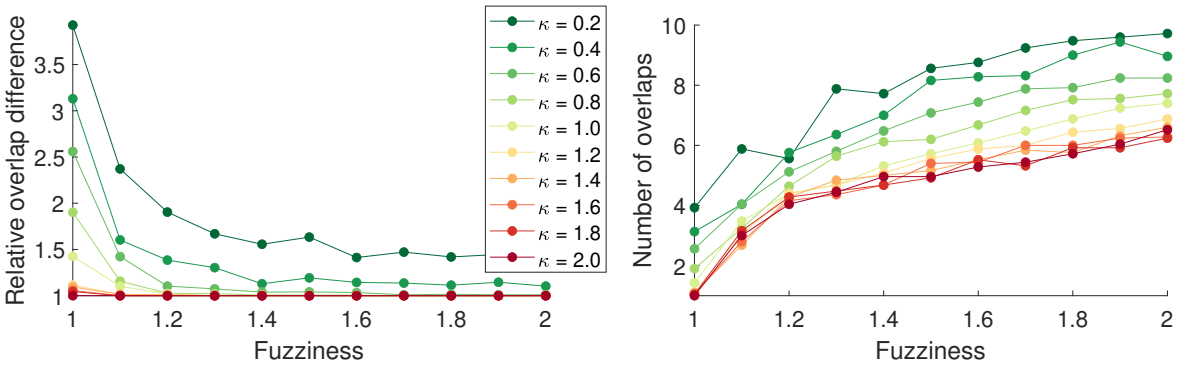


Figure A.3: Varying cluster overlap based on scaling the clustering weights for the sector coordinates by κ. Left: a plot comparing the relative increase in cluster overlap compared to the cleaned degree. Right: an absolute comparison of the resulting overlap after interpolation.

In this experiment, the coordinate features $[\bar{y}_i, \bar{x}_i]$ $\forall i \in \mathcal{N}$ of the default clustering weights are multiplied by factor κ. The default weights are set to $w = [1.5, 1.5, 0.5, 5]$ for $X_i = [\bar{y}_i, \bar{x}_i, \bar{R}_i, \bar{\dot{r}}_i]$, for all sectors $i \in \mathcal{N}$. The factor for this experiment is varied between $\kappa = \{0.2 \dots 2\}$. A small κ results in scattered, sparse clusters. A high κ results in very compact, dense clusters. The experiments are performed five times, with $\beta = 0.1$. The results are displayed in Figure A.3. In the left plot, the relative difference in cluster overlap is compared between the cleaned degree and the interpolated degree. For extremely compact clusters, the post-processing does not result in extra interpolation, even if the fuzzifier is very high. On the contrary, for very sparse clusters the interpolation has a larger effect, which is dependent on the fuzzifier. For small fuzzifiers, the cleaned degree generates up to 4 times less overlap than the interpolated degree. In the right plot, the absolute number of average overlaps per cluster is displayed. Independent of the clustering weights for the coordinates, the overlap increases for a higher fuzzifier, for which the trend is approximately equal. However, it is clear that compact clusters result in consistently less overlap, as can be expected.

A-1-4 Cluster size

Lastly, also the cluster size is considered. A fixed setup is used with $m = 1.1$, and $\beta = 0.1$. The number of clusters are varied between $N = \{50 \dots 400\}$ and $M = \{2 \dots 50\}$. In Figure A.4

the results are displayed. Instead of the absolute number of overlapping clusters, the y-axis is now expressed as a percentage of the total number of clusters present. On the x-axis, the cluster sizes are expressed as a percentage of the total number of sectors. This range is defined as $\frac{1}{M} = \{0.5 \dots 0.02\}$. It becomes clearly visible that if a few clusters cover a large portion of the sectors, the overlap becomes larger. However, if the problem consists of fewer sectors, the growth in overlap less radical, which could be due to a lack of interpolation for smaller clusters. Furthermore, it can be intuitively explained by considering that if clusters consist of many sectors, the probability for any of their sectors overlapping increases. Additionally, if fewer clusters are present, the percentage of clusters it has overlap with increases.

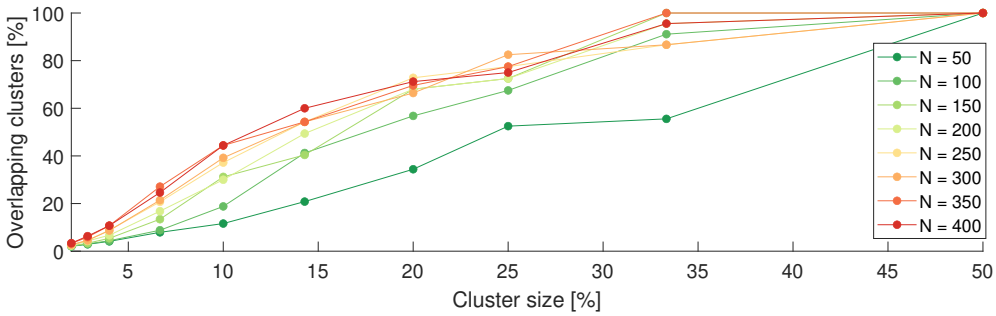


Figure A.4: The average percentage of total clusters any cluster overlaps with (y-axis), compared to the average cluster size as a percentage of the number of sectors N (x-axis).

A-2 Tuning capacity violation

The Multi-Agent Reconnaissance Problem (MARP) does not include an explicit form of capacity. Instead, a low speed penalizes the initial distance to each sector, and a short flight time penalizes the cycle distance. However, the task allocation used as a top-down heuristic does include a capacity, which is intended to prevent the allocation of long distances to unsuitable agents.

Because of this approach, inefficiencies can occur. To understand this, consider a basic example of sensors 1 and 2, needing to cover sectors A and B. Both sensors are almost equal, though sensor 1 is slightly better. Both sensors have ample speed and flight time to cover both sectors in one flight. The task allocation would then assign both sectors to sensor 1. In reality, however, the optimal solution would be to allocate one sector to sensor 1, and one to sensor 2. This illustrates that calculating the capacity as an absolute value, meaning the number of sectors it can cover in one flight, has no clear benefit for the underlying MARP. Instead, it should be used to define the *relative* difference between the available sensors. The question then is: At which point is it worthwhile to penalize a sensor compared to its relatives, and by how much?

To answer this question, the standard capacity b_k is normalized, such that $\bar{b}_k = b_k / \sum_{k \in \mathcal{P}} b_k = 1$. Then, a capacity scaling factor n^b is introduced making $\hat{b}_k = n^b \cdot \bar{b}_k$. For $n^b = 1$, all sectors can be allocated without violating the capacity constraint. For $n^b = 0.5$, only half, and so forth. Note that even with $n^b = 1$ the capacity can be violated, as the utility can lead to one

agent performing more than its relative capacity. Setting $n^b < 1$ means that exceeding the relative capacity is punished quicker.

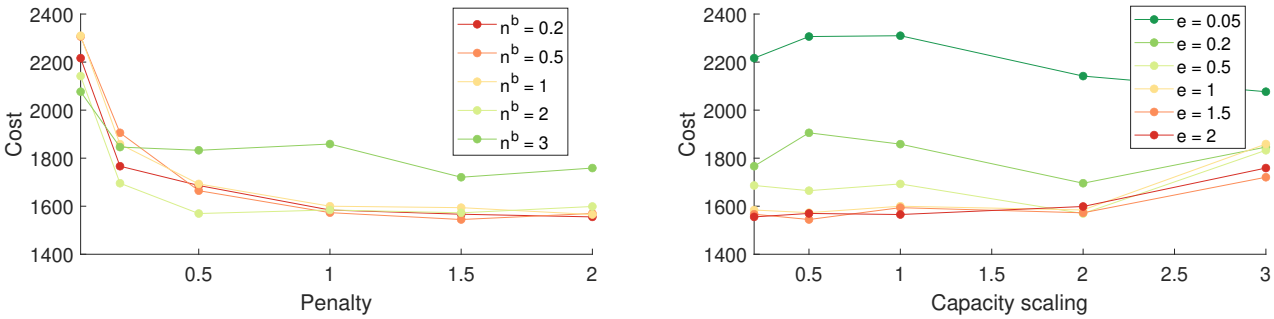


Figure A.5: Left: the effect on the MARP cost by scaling the normalized capacity with n^b. Right: The effect on the MARP cost by scaling the overtime penalty e.

An experiment is performed for a fixed setup with $N = 30$, $P = 4$, and $M = 9$. The interpolation cut-off degree is set at $\beta = 0.1$. The capacity violation penalty is ranged within $e = \{0.05 \dots 6.4\}$ and the capacity scaling factor in $n^b = \{0.1 \dots 1.5\}$ The results are displayed in Figure A.5. From both plots it becomes clear that scaling the capacity down to $n^b \leq 1$ returns the best results, though setting it lower does not improve the solution. For the penalty, there seems to be no significant difference between 1.5 and 2. Therefore, for further experiments it is set to $e = 1.8$. It is very clear that a too low penalty has a significant negative effect on the objective. Following the same argument, the objective is also significantly worse if sensors have too much capacity,and subsequently cover too many sectors.

A-3 Varying weights for capability-based clustering

The solution approach deals with heterogeneous sensors in two ways. It is directly included during allocation as a utility in the objective and as a capacity constraint. However, in Section 5-3-1 it is hypothesized that tuning the clustering weights can result in more suitable clusters for specific agents, leading to a more distinct utility.

To test this, the relative size between the weights is varied in an equal manner as in Experiment A. The weights can be increased by a factor κ for either the coordinates $[\bar{y},\ \bar{x}]$, the risk $\bar{R}$, or the risk increase $\bar{\dot{r}}$. The sensor heterogeneity is measured as the standard deviation σ^h in speed and flight time of all sensors.

The results are displayed in Figure A.6. In the left plot, the absolute cost for varying weight scaling κ and sensor heterogeneity σ^h are shown. It is clearly visible that the heterogeneous teams outperform the other teams by a large amount. Whether this can be attributed to the heterogeneity itself or a disproportionate decrease in cost by the best-performing agents is unclear. Furthermore, for all different heterogeneous setups, there is a clear decrease in performance for low weights. Most probably, this is due to scattered clusters, for which the post-processing does not yield efficient interpolations anymore. In the right plot, the performance of each weight scaling is displayed for different levels of heterogeneity. It can be observed that high weights for the coordinates are marginally better for homogeneous teams.

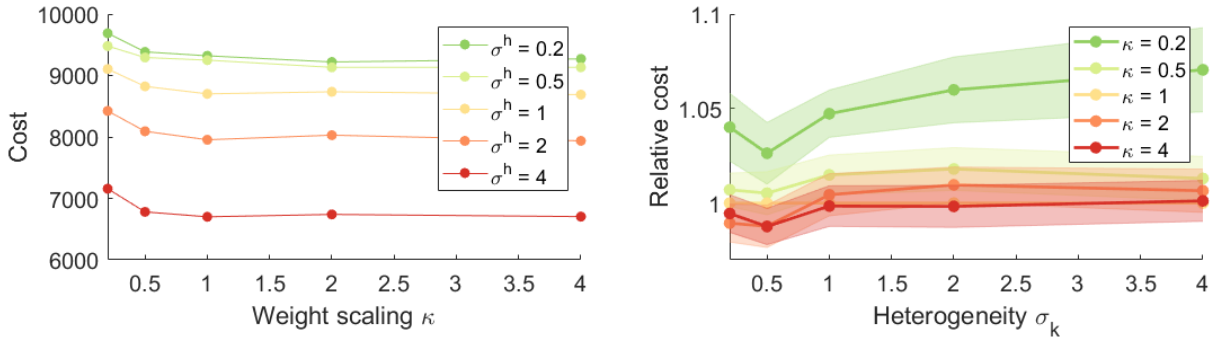


Figure A.6: Left: the total cost depending on the deviation in the sensor properties. High deviation results in much better performance. Right: the relative performance for each scale with a 95% confidence interval. A low weight for sector coordinates performs significantly worse.

This means that more compact clusters are created, regardless of risk or risk increase. For higher heterogeneity, the results are equal for the default weights or higher.

In line with this argument, an experiment is performed in which either the flight time is fixed and the speed varied, or vice versa. Then, the question is whether scaling the weights separately for the risk or risk increase yields significantly different results. As described in Section 5-3-1, it is expected that higher weights for risk increase will perform better with larger deviation in flight time. The same argument is made for a higher weight on risks for larger deviation in speed.

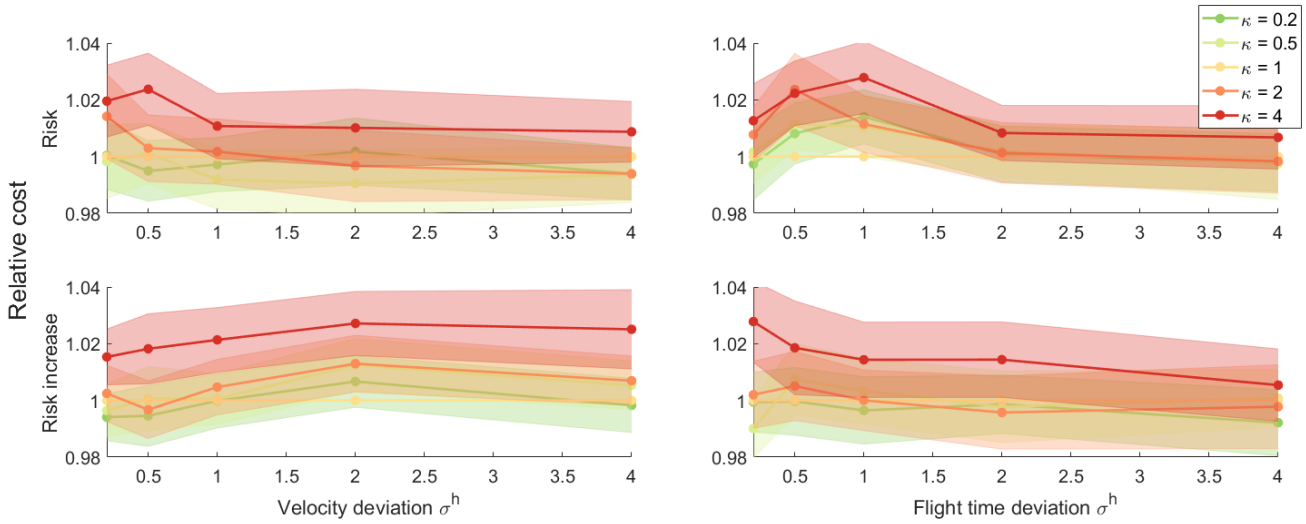


Figure A.7: A plot indicating the relation between scaling the risk or risk increase clustering weights by κ, and a heterogeneity in velocity or flight time by the standard deviation σ^h. Although some trend is visible, a clear correlation is not visible. Still, it is clear that insufficient prioritization of coordinates ($\kappa = 4$), generates worse results.

For each weight scale κ and sensor deviation σ^h, a simulation is performed 50 times. The results are displayed in Figure A.7. A 95% confidence interval is drawn to indicate the spread of the results. It is clearly visible that a high weight of $\kappa = 4$ underperforms for all cases.

However, it is difficult to draw any other conclusions. The deviations are spread out over a relatively wide, overlapping region, and the absolute performance differences are meager, as they deviate on average 0.8% from the default weight solution. Still, some trends are visible, indicating some relation between heterogeneity and clustering weights. It is, however, not as directly correlated as hypothesized.

A-4 Parameter summary

A-4-1 Intent & cluster overlap

As overlapping clusters represent intent, it is important to identify how this is influenced. Naturally, the overlap is larger for a larger fuzzifier m. The post-processing method also increases overlap, but this effect is highly dependent on the cluster sparseness. This, in turn, is dependent on the clustering weights for the sector coordinates. For the selected default weights, the added overlap by interpolation is small and diminishes for higher fuzziness. Furthermore, the overlap is reduced by a higher cut-off threshold β, for which the effect increases for higher fuzziness. Lastly, the percentage of overlapping clusters is highly correlated with the relative cluster size, for which the effect is larger if more sectors are considered. As default values, a fuzzifier $m = 1.15$ and cut-off degree $\beta = 0.03$ are chosen.

A-4-2 Tuning the capacity

Furthermore, the capacity is normalized and scaled by a factor n^b to improve the task allocation. The capacity violation penalty e is also varied to measure the combined effect. Any value $n^b \leq 1$ generates approximately the same, best case result. However, independent of the scaling, if the penalty is set too low, the performance decreases dramatically. Any value $1.5 \leq e \leq 2$ can be regarded as a safe choice.

A-4-3 Accommodating sensor heterogeneity

The relation between sensor heterogeneity, expressed as the standard deviation of the sensor properties σ^h and the clustering weights w, is analysed in further depth. It becomes clear that high heterogeneity significantly outperforms homogeneous agents. Furthermore, setting the clustering weights too low for the Cartesian coordinates results in significantly worse results. Increasing these weights beyond the default values has no apparent effect. When comparing the deviations in speed and flight time separately, some trends are visible for separate scaling of risk and risk increase weights. However, no clear relation can be uncovered, which appears to be more complex than expected. The results from these experiments are in line with the scaling of the coordinate weights, as a very high priority on either risk or risk increase lowers the performance equally as when the weight for the coordinates is lowered too much.

Appendix B

Additional Experiments

In this appendix, some additional experiments are included that are not a core part of the result but are sometimes referenced. Furthermore, some experiments are performed as a thought experiment for future work.

B-1 Convergence Fuzzy C-Means (FCM)

The limits of FCM are assessed by analyzing the convergence and runtimes, dependent on the number of clusters and sectors. For the input scenario, the data is sampled to fit the number of sectors, as explained in Section 6-1. For each setup, the experiment is repeated five times due to the stochastic initialization of the cluster centroids. The results are averaged over all simulations. The amount of sectors is varied from $N = \{100 \dots 500\}$, width for each setup a varying number of clusters $M = \{4 \dots \frac{N}{4}\}$. Furthermore, the threshold for convergence is set at 0.1% improvement and the iteration limit at 200.

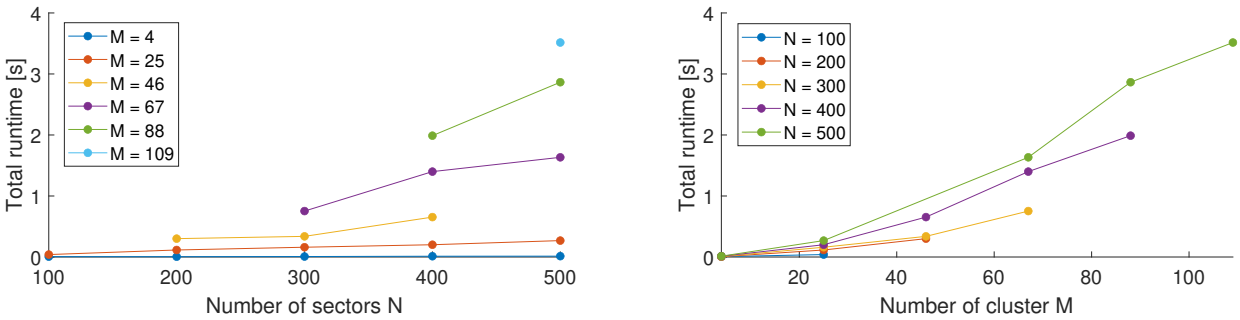


Figure B.1: The correlation between the number of sectors and clusters, and the total runtime until convergence. Adding clusters has a greater effect than adding sectors, which can be expected.

In Figure B.1, it is clearly visible that the runtime is increasingly dependent on the number of clusters, though the number of iterations needed remains roughly equal. For an equal number of sectors, a higher number of clusters shows consistently longer runtimes. The number of sectors appears to have only a mild effect on the total runtime. This effect is to

be expected. Degree calculations for FCM are repeated for every sector and every cluster. The addition of a cluster will add a calculation for every sector and vice versa. As there are naturally more sectors than clusters, the effect of adding a cluster is much larger. Though the clustering process takes more time for an increasing number of clusters, it will also increase the allocation runtime.

B-2 Clustering significance

One could argue that very small or large clusters have no benefit. This problem is particularly evident for $M = N$ or $M = 1$. In these cases, each sector is its own cluster, or all sectors are in a single cluster, respectively. In this experiment, the cluster significance is assessed.

FCM creates cluster similarity by minimizing the Euclidean distance between the feature vectors X_i, by varying the degree u_{ij} and cluster centroids C_j. The average similarity d after performing FCM is therefore given as:

$$d = \frac{1}{N \cdot M} \sum_{i \in \mathcal{N}} \sum_{i \in \mathcal{M}} u_{ij}^{m} ||X_i - c_j||^2 \tag{B.1}$$

Consequently, the different centroids are expected to become dissimilar. Dissimilarity is measured using the standard deviation between the cluster centroids, with μ being the average:

$$\sigma = \sqrt{\left(\frac{1}{M} \sum_{j}^{M} (C_j - \mu)^2 \right)} \tag{B.2}$$

These metrics are compared to the random clustering benchmark from Section 4-2 to measure the significance, for which the same similarity d and dissimilarity σ are calculated. For a setup of $N = 200$ and $M = \{2 \dots 200\}$ the relative performance is simulated 5 times. The results are displayed in Figure B.2. The similarity is significantly better for smaller cluster sizes, but the opposite is true for feature deviation. This could imply that for small clusters, the random clustering performs equally well as FCM.

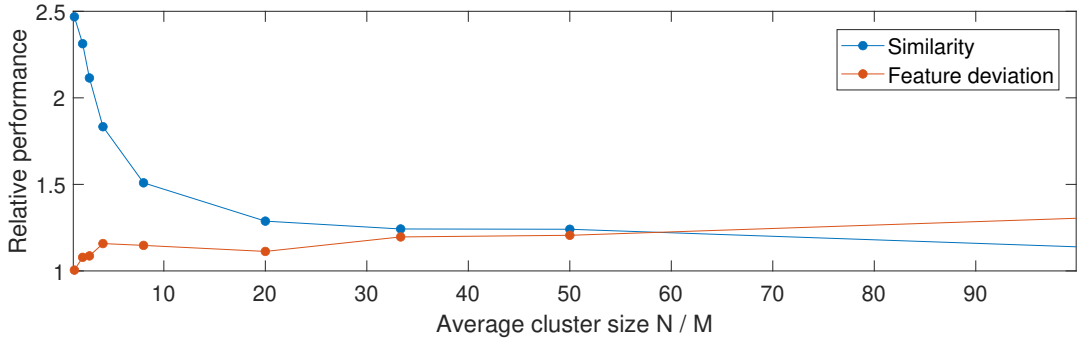


Figure B.2: Though the similarity is higher for small clusters, the low difference in feature deviations with the random approach indicates small FCM clusters are of insignificant value.

B-3 Provisional allocation based on a relaxed solution

The column generated lower bound originates from the branch & price scheme, which can be used to solve the underlying problem to optimality. As such, if a branching method would be added, the method can be used to solve the Multi-Agent Reconnaissance Problem (MARP) to optimality. The question then arises, what is the quality of the – possibly fractional – allocation, retrieved by the initial column generated lower bound?

To use the allocation, the included sequences $r \in \Omega_r$, the allocations x_{rk}, and the sequences a_{ir} are collected from the Restricted Master Problem (RMP), and transformed to a feasible allocation. First, a fractional allocation cannot be performed, so x_{rk} is transformed to a binary solution. Secondly, Constraint (4.2) of the RMP has to be satisfied, such that all sectors are covered by one sequence only. The sector is allocated to the agent with the highest fraction to remove the fractional allocation of multiple agents:

$$\hat{x}_{rk} = \begin{cases} 1 & k = \arg\max_{k \in \mathcal{P}}(x_{rk}) \\ 0 & otherwise \end{cases} \qquad \forall r \in \Omega_r,\ k \in \mathcal{P} \tag{B.3}$$

Though each partially allocated sequence is now performed by at least one agent, it is still possible that sectors are covered by multiple sequences that are now both fully allocated. Therefore, the sequences itself are transformed to prevent this problem. For each sector in each sequence, it is only kept of that sequence covers it by the largest fraction:

$$\hat{a}_{ir} = \begin{cases} 1 & r = \arg\max_{r \in \Omega_r}(a_{ir} \cdot \sum_{k \in \mathcal{P}} x_{rk}) \\ 0 & otherwise \end{cases} \qquad \forall i \in \mathcal{N},\ r \in \Omega_r \tag{B.4}$$

Lastly, the regular Single-Agent Reconnaissance Problems (SARPs) heuristics from Section 4-3 can be applied to the transformed sequences $\hat{a}_{ir}$ and allocations $\hat{x}_{rk}$.

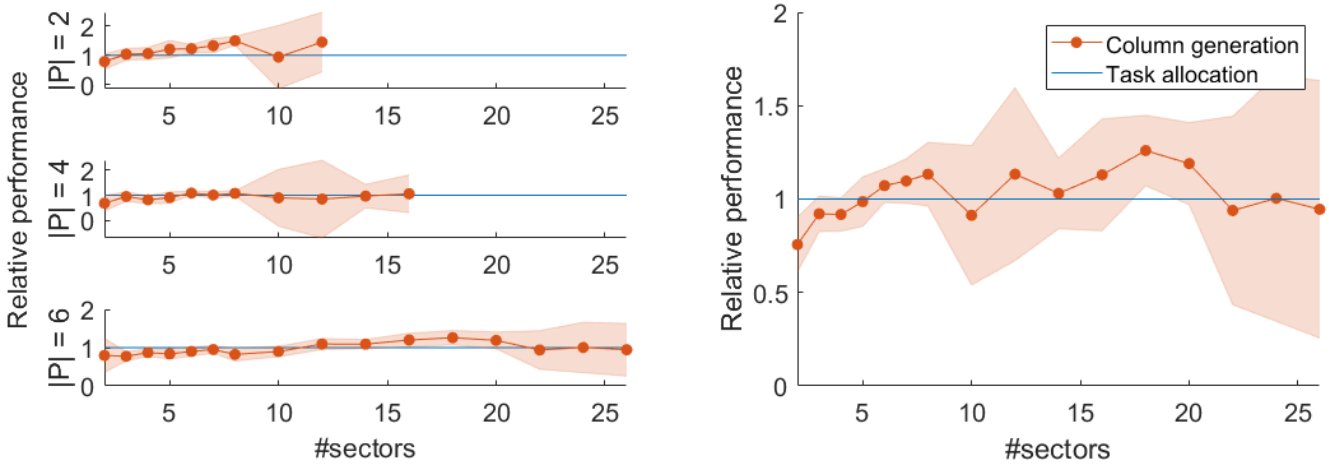


Figure B.3: The relative performance for each number of sensors (left) and the combined results (right) are shown with a 95% confidence interval. These results do not provide evidence against the use of transformed fractional allocation from the column generated lower bound.

The experiment is repeated 5 times for an increasing number of sectors $N = \{2 \dots 30\}$, and sensor $P = \{2, 4, 6\}$. The column generation is cut-off if the calculation time exceeds 10 seconds. The results are compared and displayed in Figure B.3. In the left plots, the results are shown separately for each amount of sensors. On the right, the combined result is displayed with a 95% confidence interval. The interval is much smaller for a lower number of sectors, as the column generation can be performed for all amounts of sensors. For larger problem sizes, fewer results are obtained. It becomes clear that performance is not significantly worse than task allocation. For smaller problem sizes, it seems to even outperform the task allocation method. For larger problem sizes, the intervals are too large to draw sharp conclusions, but there is no evidence against the usage of the transformed allocation.

B-4 Predicting performance differences in the hierarchy

An additional feature of the hierarchical approach is that the intermediate results within the hierarchy can also be benchmarked. An example is displayed in Figure B.4. For each supervisor, the three values represent the $LB \leq opt \leq TA$ solutions, respectively. For the agents, only $opt \leq TA$ is shown. The colors of the arcs in the hierarchy represent the relative gap between the lower bound and the task allocation. The nodes are colored depending on the relative gap between the optimal solution and the task allocation. In this specific example, it can be seen that the largest gap with the lower bound also represents the largest gap with the optimal solution. If this correlation is consistent, this could imply that it can be predicted which parts of the solution underperform. These specific parts could then be re-optimized to improve the solution.

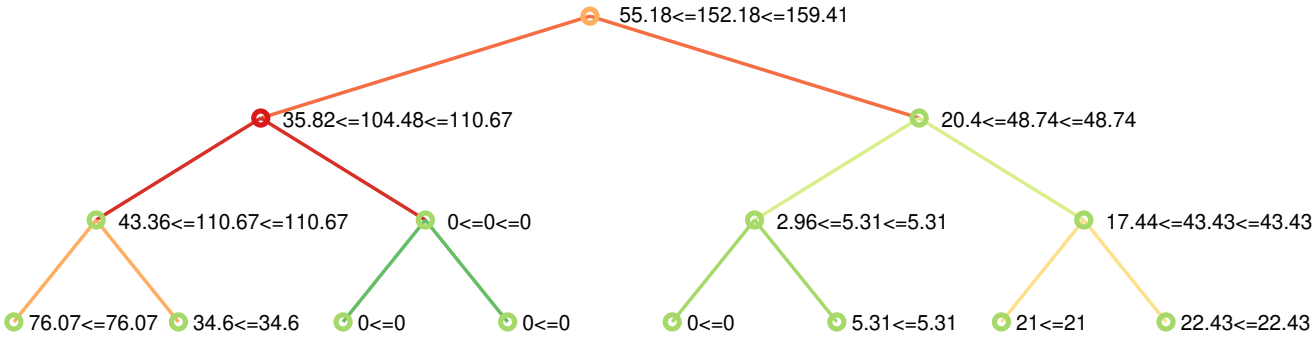


Figure B.4: Intermediate results within a hierarchy. A red arc color indicates a larger gap between the task allocation and relaxed lower bound. A red node indicates a larger gap between the task allocation and optimal solution. Possibly the gap with the lower bound predicts the gap with the optimal solution. In this selected example, it clearly does.

An experiment is performed with 30 simulations, in which these correlations are measured for $N = 6$, $P = 8$, and a group size of 2. As the computation time for the optimal solution explodes quickly it is not possible to include more sectors. The results are displayed in Figure B.5. On the x-axis, the relative performance of task allocation to the LB is shown. On the y-axis the relative performance of the task allocation is compared to the optimal value. Furthermore, a second-order interpolation is fitted to the data, capped at 1. A strong

correlation is visible, as worse relative performance to the optimal solution also implies a bigger gap with the lower bound. The converse relation, however, is not so clear. Unfortunately, this is the relation needed to predict the performance.

Using the Pearson correlation coefficient, the linear correlation has a value of 0.28, with a p-value of 4.1e-05, clearly signalling a positive correlation. This correlation cannot be applied directly, but it could provide a basis for further improvements to the solution method. Possibly, within the hierarchy supervisors can be selected as a candidate for improvement.

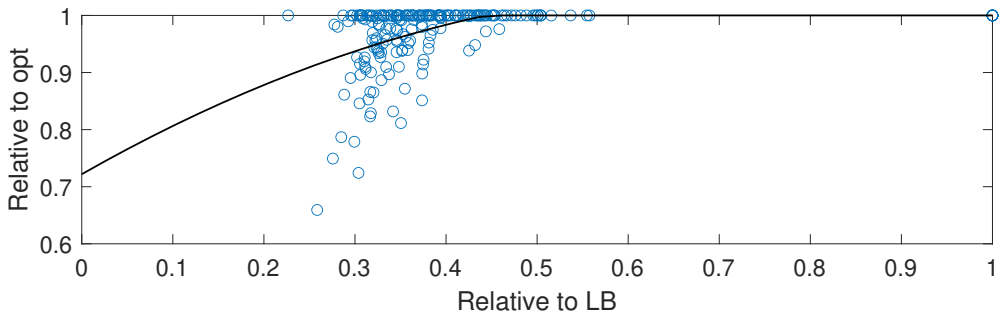


Figure B.5: The correlation between the relative performance of task allocation to the lower bound (x-axis), and the relative performance to the optimum (y-axis).

Appendix C

Task dependency proof

Section 5-4-1 highlights how the capacity and task sizes are related. To this end, task dependency is introduced to account for inefficiencies arising from the distance between tasks.

The difficulty is that capacity is measured as the ability to cover a squared distance, as opposed to the distance between tasks. Therefore, some quadratic measure must be included to express the inefficiencies as an increase in the area that has to be covered. The resulting measure is a fractional approximation of the added squared distance:

$$D_{ij}^{f} = \frac{A_i A_j ||F_i - F_j||_2^2}{\sum_{i,j \in \mathcal{M}} A_i A_j ||F_i - F_j||_2^2} \qquad \forall i,j \in \mathcal{M} \tag{C.1}$$

This formula squares the distance between the task centroids $i, j \in \mathcal{M}$, weighed by the total squared distance between all centroids. These metrics, however, are also weighed by each respective task size.

Weighing over the respective task sizes is necessary to overcome the problem of increasing cost for increasingly more, smaller tasks. This example is illustrated in Figure C.1. Given that task 1-4 and task 7-10 are collected in assignment S_1, and task 5 and 6 in assignment S_2, their areas are exactly equal in size and shape. Without weighing, however, the task dependency cost of S_1 would be much higher, as there are more tasks generating more dependencies.

More formally, given there are currently N tasks, the total number of dependencies is given as $2\sum_{n=1}^{N}(n-1) = (N-1)N$. This means $(N-1-|S_2|)(N-|S_2|)$ dependencies belong to S_1, and $(N-1-|S_1|)(N-|S_1|)$ to S_2. If a task in S_1 is split, this generates $2(N-|S_2|)$ new dependencies for S_1, but none for S_2. It needs to be shown that the sum of the dependencies in S_1 grows in size compared to S_2, when splitting the tasks in S_1 indefinitely.

After splitting tasks in S_1, there are n_1 unaltered and $\tilde{n}$ new tasks in S_1, and n_2 unaltered tasks in S_2. The total sum of the dependencies of the unaltered tasks in S_1 can be denoted as the constant b_1, and of S_2 by b_2. The $\tilde{n}$ new tasks produce new dependencies, which are at least as large as the smallest dependency b^-. Dependency b^- could even be zero, if for instance all tasks are concentric circles. The summation of the dependencies b_1, b_2, and all new dependencies of at least size b^-, give a lower bound on the actual sum of all dependencies:

$$b_1 + b_2 + b^- \tilde{n}(\tilde{n}-1) + 2b^- \tilde{n} n_1 + 2b^-(\tilde{n}+n_1)n_2 \leq \sum_{i,j \in \mathcal{M}} ||F_i - F_j||_2^2 \tag{C.2}$$

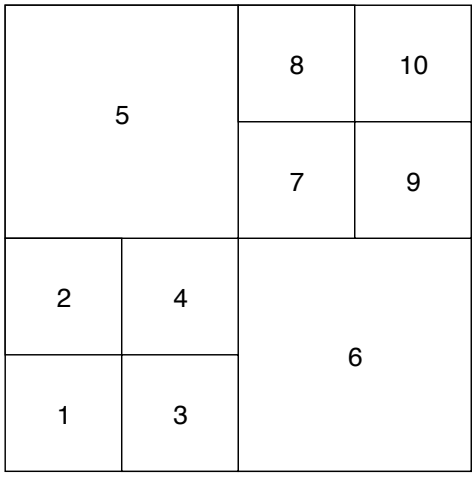


Figure C.1: An example of 10 tasks with unequal sizes. Without weighing the task dependency cost, task 1-4 and 7-10 combined would yield higher costs than 5 and 6 combined, though the areas are equal.

The cumulative fractional dependency for set S_2 can be formulated as:

$$D^f_{S2} = \frac{b_2}{\sum_{i,j\in\mathcal{M}} ||F_i - F_j||_2^2} \tag{C.3}$$

Then in the limit of $\tilde{n}$, an upper bound on this measure can be written as:

$$\lim_{\tilde{n}\to\infty} D^f_{S_2} \leq \frac{b_2}{b_1 + b_2 + b^-\tilde{n}(\tilde{n}-1) + 2b^-\tilde{n}n_1 + 2b^-(\tilde{n}+n_1)n_2} = 0 \tag{C.4}$$

This proves that without weighing, splitting tasks in S_1 indefinitely would result in $D^f_{S_2} \to 0$.

Furthermore, the largest dependency to have ever been present between S_1 and S_2 is called b^+. Then, an upper bound on the total sum of dependencies between S_1 and S_2 is given as:

$$2b^+(\tilde{n}+n_1)n_2 \geq \sum_{i\in S_1}\sum_{j\in S_2} ||F_i - F_j||_2^2 \tag{C.5}$$

In the limit of $\tilde{n}$, this forms an upper bound on the fractional cumulative dependencies between the sets:

$$\lim_{\tilde{n}\to\infty} D^f_{S_1S_2} \leq \frac{2b^+(\tilde{n}+n_1)n_2}{b_1 + b_2 + b^-\tilde{n}(\tilde{n}-1) + 2b^-\tilde{n}n_1 + 2b^-(\tilde{n}+n_1)n_2} = 0 \tag{C.6}$$

Now, it is also proven that without weighing, $D^f_{S_1S_2} \to 0$. By definition it should hold that $D^f_{S_1} + D^f_{S_2} + D^f_{S_1S_2} = 1$. Therefore, it can be concluded that $D^f_{S_1} \to 1$ for $\tilde{n} \to \infty$, while it should hold that $D^f_{S2} \approx D^f_{S1}$, as the areas are equal.

When weighing with the size of the tasks, a converse argument can be provided. Again, for the unaltered tasks in S_1 and S_2, the weighed dependency can simply be shortened as b_1 and b_2 respectively. Furthermore, the largest and smallest original task size of S_1 can be defined as A^+ and A^-, and the size of either task in S_2 as A_2. Following a logical argument, the average fraction of a sector left after splitting is $\frac{1}{\tilde{n}}$. This is true as no area is added or

removed, such that the summed fractions of all split tasks will equal one. An upper bound on the average size of a split task is then defined as $\frac{1}{\tilde{n}}A^+$. Using this, Equation (C.2) can be rewritten as an upper bound:

$$\begin{aligned} b_1 + b_2 + b^+ \left(\frac{A^+}{\tilde{n}}\right)^2 \tilde{n}(\tilde{n}-1) + 2b^+ \frac{A^+}{\tilde{n}} \tilde{n} A^+ n_1 + 2b^+ \left(\frac{A^+}{\tilde{n}}\tilde{n} + A^+ n_1\right) A_2 n_2 = \\ b_1 + b_2 - b^+ \frac{(A^+)^2}{\tilde{n}} + 2b^+ (A^+)^2 n_1 + 2b^+ A^+ A_2 (1+n_1) n_2 \geq \sum_{i,j \in \mathcal{M}} A_i A_j ||F_i - F_j||_2^2 \end{aligned} \tag{C.7}$$

Subsequently, the upper bound (C.4) can be reformulated as a lower bound:

$$\lim_{\tilde{n}\to\infty} D^f_{S2} \geq \frac{b_2}{b_1 + b_2 - b^+ \frac{(A^+)^2}{\tilde{n}} + 2b^+ (A^+)^2 n_1 + 2b^+ A^+ A_2 (1+n_1) n_2} \tag{C.8}$$

In this case, cumulative fractional dependency $D^f_{S_2} \not\to 0$ for $n \to \infty$, but remains above a certain constant.

In a similar manner, Equation (C.5) can be rewritten to the converse lower bound:

$$2b^- \left(\frac{A^-}{\tilde{n}}\tilde{n} + A^- n_1\right) A_2 n_2 \leq \sum_{i \in S_1} \sum_{j \in S_2} A_i A_j ||F_i - F_j||_2^2 \tag{C.9}$$

Such that the upper bound from Equation (C.6) becomes a lower bound:

$$\lim_{\tilde{n}\to\infty} D^f_{S_1 S_2} \geq \frac{2b^- A^- A_2 (1+n_1) n_2}{b_1 + b_2 - b^+ \frac{(A^+)^2}{\tilde{n}} + 2b^+ (A^+)^2 n_1 + 2b^+ A^+ A_2 (1+n_1) n_2} \tag{C.10}$$

Again, it is shown that $D^f_{S_1 S_2} \not\to 0$ for $\tilde{n} \to \infty$. Combined with Equation (C.8) this must imply that $D^f_{S_1} \not\to 1$ for $n \to \infty$. It can be concluded that by weighing the dependencies with the size of each task, the sum of the dependencies in each set is approximately insensitive to the number of tasks it is composed of.

Then a generalized measure of capacity can be constructed for each agent $k \in \mathcal{P}$. First, there is the fractional task dependency $D^f_k \in [0,1]$, in which all dependencies of all allocated tasks are summed. Second is the fractional task size $A^f_k \in [0,1]$, in which the size of all allocated tasks is summed. Ideally, one would then like to enforce the following constraint on the solution:

$$\frac{1}{2} D^f_k + \frac{1}{2} A^f_k \leq \hat{b}_k \qquad \forall k \in \mathcal{P} \tag{C.11}$$

This equation states that the total task size and total task dependency should not exceed the fractional area an agent can effectively service. Using these measures, a formulation can be constructed to perform the task allocation.

Bibliography

[1] Ministerie van Defensie, "Defensievisie 2035. vechten voor een veilige toekomst," 2020.

[2] S. Bungay, "The road to mission command: The genesis of a command philosophy," *British Army Review*, vol. 137, pp. 22 – 29, 2005.

[3] D. Feillet, P. Dejax, M. Gendreau, and C. Gueguen, "An exact algorithm for the elementary shortest path problem with resource constraints: Application to some vehicle routing problems," *Networks: An International Journal*, vol. 44, no. 3, pp. 216–229, 2004.

[4] J. C. Bezdek, *Pattern recognition with fuzzy objective function algorithms*. Springer Science & Business Media, 1981.

[5] A. Netzer, A. Grubshtein, and A. Meisels, "Concurrent forward bounding for distributed constraint optimization problems," *Artificial Intelligence*, vol. 193, pp. 186–216, 2012.

[6] M. A. Korthals Altes, "A framework for intent-based command and control of robotic autonomous systems for persistent reconnaissance - literature survey." Unpublished, 2020.

[7] N. Nigam, "The multiple unmanned air vehicle persistent surveillance problem: A review," *Machines*, vol. 2, no. 1, pp. 13–72, 2014.

[8] M. M. Polycarpou, Y. Yang, and K. M. Passino, "Cooperative control of distributed multi-agent systems," *IEEE Control Systems Magazine*, vol. 21, pp. 1–27, 2001.

[9] A. E. Gil, K. M. Passino, and J. B. Cruz Jr, "Stable cooperative surveillance with information flow constraints," *IEEE Transactions on Control Systems Technology*, vol. 16, no. 5, pp. 856–868, 2008.

[10] Y. Yang, M. M. Polycarpou, and A. A. Minai, "Multi-UAV cooperative search using an opportunistic learning method," *Journal of Dynamic Systems, Measurement, and Control*, vol. 129, no. 5, pp. 716–728, 2007.

[11] C. A. Baker, S. Ramchurn, W. T. Teacy, and N. R. Jennings, "Planning search and rescue missions for UAV teams," *Frontiers in Artificial Intelligence and Applications*, vol. 285, pp. 1777–1782, 2016.

[12] C. Hu, Z. Zhang, N. Yang, H. S. Shin, and A. Tsourdos, "Fuzzy multiobjective cooperative surveillance of multiple UAVs based on distributed predictive control for unknown ground moving target in urban environment," *Aerospace Science and Technology*, vol. 84, pp. 329–338, 2019.

[13] L. Lin and M. A. Goodrich, "Hierarchical heuristic search using a gaussian mixture model for UAV coverage planning," *IEEE Transactions on Cybernetics*, vol. 44, no. 12, pp. 2532–2544, 2014.

[14] J. Tisdale, A. Ryan, Z. Kim, D. Tornqvist, and J. K. Hedrick, "A multiple UAV system for vision-based search and localization," *Proceedings of the American Control Conference*, pp. 1985–1990, 2008.

[15] K. N. McGuire, C. de Wagter, K. Tuyls, H. J. Kappen, and G. C. H. E. de Croon, "Minimal navigation solution for a swarm of tiny flying robots to explore an unknown environment," *Science Robotics*, vol. 4, no. 35, 2019.

[16] Y. Wang, P. Bai, X. Liang, W. Wang, J. Zhang, and Q. Fu, "Reconnaissance Mission Conducted by UAV Swarms Based on Distributed PSO Path Planning Algorithms," *IEEE Access*, vol. 7, pp. 105086–105099, 2019.

[17] M. Scheutz, P. Schermerhorn, and P. Bauer, "The utility of heterogeneous swarms of simple UAVs with limited sensory capacity in detection and tracking tasks," in *Proceedings IEEE Swarm Intelligence Symposium*, pp. 257–264, IEEE, 2005.

[18] W. M. Shen, P. Will, A. Galstyan, and C. M. Chuong, "Hormone-inspired self-organization and distributed control of robotic swarms," *Autonomous Robots*, vol. 17, no. 1, pp. 93–105, 2004.

[19] S. Ragi and E. K. Chong, "UAV path planning in a dynamic environment via partially observable markov decision process," *IEEE Transactions on Aerospace and Electronic Systems*, vol. 49, no. 4, pp. 2397–2412, 2013.

[20] H.-J. Chae, S.-S. Park, H.-V. Kim, H.-S. Ko, and H.-L. Choi, "UAV path planning for local defense systems," in *RITA 2018*, pp. 199–211, Springer, 2019.

[21] H. Y. Shin., A. Tsourdos, V. Lappas, A. Ampatzoglou, V. Kostopoulos, S. Bertrand, J. Marzat, H. Piet-Lahanier, D. Lindgren, and M. Detratti, "Autonomous unmanned heterogeneous vehicles for persistent monitoring," *AIAA Scitech 2019 Forum*, pp. 1–24, 2019.

[22] B. D. Song, J. Kim, and J. R. Morrison, "Rolling Horizon Path Planning of an Autonomous System of UAVs for Persistent Cooperative Service MILP Formulation and Efficient Heuristics," *Journal of Intelligent and Robotic Systems Theory and Applications*, vol. 84, no. 1-4, pp. 241–258, 2016.

[23] H. X. Chen, Y. Nan, and Y. Yang, "Multi-UAV reconnaissance task assignment for heterogeneous targets based on modified symbiotic organisms search algorithm," *Sensors (Switzerland)*, vol. 19, no. 3, 2019.

[24] C. Ercan and C. Gencer, "An Integer Programming Model for the Heterogeneous MILP Fleet Routing Problemsi," *Savunma Bilimleri Dergisi*, vol. 12, no. 2, pp. 119–144, 2013.

[25] S. K. K. Hari, S. Rathinam, S. Darbha, K. Kalyanam, S. G. Manyam, and D. Casbeer, "The generalized persistent monitoring problem," in *American Control Conference (ACC)*, pp. 2783–2788, 2019.

[26] K. Sundar and S. Rathinam, "Algorithms for Heterogeneous, Multiple Depot, Multiple Unmanned Vehicle Path Planning Problems," *Journal of Intelligent and Robotic Systems: Theory and Applications*, vol. 88, no. 2-4, pp. 513–526, 2017.

[27] J. Faigl and P. Vana, "Unsupervised learning for surveillance planning with team of aerial vehicles," *Proceedings of the International Joint Conference on Neural Networks*, vol. 2017-May, pp. 4340–4347, 2017.

[28] Yan Jin, A. A. Minai, and M. M. Polycarpou, "Cooperative real-time search and task allocation in UAV teams," in *42nd IEEE International Conference on Decision and Control*, vol. 1, pp. 7–12 Vol.1, 2003.

[29] L. F. Bertuccelli, H. L. Choi, P. Cho, and J. P. How, "Real-time multi-UAV task assignment in dynamic and uncertain environments," *AIAA Guidance, Navigation, and Control Conference and Exhibit*, pp. 1–16, 2009.

[30] X. Hu, H. Ma, Q. Ye, and H. Luo, "Hierarchical method of task assignment for multiple cooperatingUAVteams," *Journal of Systems Engineering and Electronics*, vol. 26, no. 5, pp. 1000–1009, 2015.

[31] J. Cortés, S. Martínez, T. Karatas, and F. Bullo, "Coverage control for mobile sensing networks," *IEEE Transactions on Robotics and Automation*, vol. 20, no. 2, pp. 243–255, 2004.

[32] K. Guruprasad and D. Ghose, "Coverage optimization using generalized voronoi partition," *arXiv preprint arXiv:0908.3565*, 2009.

[33] K. Laventall and J. Cortés, "Coverage control by multi-robot networks with limited-range anisotropic sensory," *International Journal of Control*, vol. 82, no. 6, pp. 1113–1121, 2009.

[34] A. Trotta, M. D. Felice, F. Montori, K. R. Chowdhury, and L. Bononi, "Joint Coverage, Connectivity, and Charging Strategies for Distributed UAV Networks," *IEEE Transactions on Robotics*, vol. 34, no. 4, pp. 883–900, 2018.

[35] M. Burger, M. Huiskamp, and T. Keviczky, "Complete field coverage as a multi-vehicle routing problem," *IFAC Proceedings Volumes*, vol. 46, no. 18, pp. 97–102, 2013.

[36] I. Maza and A. Ollera, "Multiple UAV cooperative searching operation using polygon area decomposition and efficient coverage algorithms," *Distributed Autonomous Robotic Systems*, vol. 6, 2007.

[37] P. Stodola, J. Drozd, J. Nohel, J. Hodickỳ, and D. Procházka, "Trajectory optimization in a cooperative aerial reconnaissance model," *Sensors*, vol. 19, no. 12, p. 2823, 2019.

[38] M. Schwager, M. P. Vitus, S. Powers, D. Rus, and C. J. Tomlin, "Robust adaptive coverage control for robotic sensor networks," *IEEE Transactions on Control of Network Systems*, vol. 4, no. 3, pp. 462–476, 2017.

[39] D. A. Anisi, P. Ögren, and X. Hu, "Cooperative minimum time surveillance with multiple ground vehicles," *IEEE Transactions on Automatic Control*, vol. 55, no. 12, pp. 2679–2691, 2010.

[40] I. I. Hussein and D. M. Stipanović, "Effective coverage control for mobile sensor networks with guaranteed collision avoidance," *IEEE Transactions on Control Systems Technology*, vol. 15, no. 4, pp. 642–657, 2007.

[41] C. Franco, D. Paesa, G. Lopez-Nicolas, C. Sagues, and S. Llorente, "Hierarchical strategy for dynamic coverage," in *IEEE International Conference on Intelligent Robots and Systems*, pp. 5341–5346, 2012.

[42] S. A. Sadat, J. Wawerla, and R. Vaughan, "Fractal trajectories for online non-uniform aerial coverage," in *IEEE International Conference on Robotics and Automation (ICRA)*, pp. 2971–2976, 2015.

[43] P. Vincent and I. Rubin, "A framework and analysis for cooperative search using UAV swarms," in *Proceedings of the 2004 ACM symposium on Applied computing*, pp. 79–86, ACM, 2004.

[44] S. G. Manyam, S. Rasmussen, D. W. Casbeer, K. Kalyanam, and S. Manickam, "Multi-UAV routing for persistent intelligence surveillance & reconnaissance missions," *International Conference on Unmanned Aircraft Systems*, pp. 573–580, 2017.

[45] S. L. Smith and D. Rus, "Multi-robot monitoring in dynamic environments with guaranteed currency of observations," *Proceedings of the IEEE Conference on Decision and Control*, pp. 514–521, 2010.

[46] J. Scherer and B. Rinner, "Persistent multi-UAV surveillance with energy and communication constraints," in *IEEE International Conference on Automation Science and Engineering*, pp. 1225–1230, 2016.

[47] K. Kalyanam, S. Manyam, A. Von Moll, D. Casbeer, and M. Pachter, "Scalable and Exact MILP Methods for UAV Persistent Visitation Problem," *IEEE Conference on Control Technology and Applications*, pp. 337–342, 2018.

[48] J. Scherer and B. Rinner, "Persistent multi-UAV surveillance with data latency constraints," *arXiv preprint arXiv:1907.01205*, 2019.

[49] G. Sun, R. Zhou, B. Di, Z. Dong, and Y. Wang, "A novel cooperative path planning for multi-robot persistent coverage with obstacles and coverage period constraints," *Sensors (Switzerland)*, vol. 19, no. 9, 2019.

[50] N. Nigam and I. Kroo, "Persistent surveillance using multiple unmanned air vehicles," *IEEE Aerospace Conference Proceedings*, pp. 1–14, 2008.

[51] N. Nigam, S. Bieniawski, I. Kroo, and J. Vian, "Control of multiple UAVs for persistent surveillance Algorithm and flight test results," *IEEE Transactions on Control Systems Technology*, vol. 20, no. 5, pp. 1236–1251, 2012.

[52] C. C. Olsen and D. L. Kunz, "A utility approach to UAS-based persistent ISR," *AIAA Information Systems-AIAA Infotech at Aerospace*, no. 209989, pp. 1–14, 2018.

[53] J. Wang, J. Guo, M. Zheng, Z. Wang, and Z. Li, "Uncertain multiobjective orienteering problem and its application to UAV reconnaissance mission planning," *Journal of Intelligent and Fuzzy Systems*, vol. 34, no. 4, pp. 2287–2299, 2018.

[54] Q. Fan, F. Wang, X. Shen, and D. Luo, "Path planning for a reconnaissance UAV in uncertain environment," *IEEE International Conference on Control and Automation, ICCA*, vol. 2016-July, pp. 248–252, 2016.

[55] B. Hefferan, O. M. Cliff, and R. Fitch, "Adversarial patrolling with reactive point processes," in *Proceedings of the ARAA Australasian Conference on Robotics and Automation*, pp. 39–46, 2016.

[56] B. Kartal, J. Godoy, I. Karamouzas, and S. J. Guy, "Stochastic tree search with useful cycles for patrolling problems," in *Proceedings - IEEE International Conference on Robotics and Automation*, pp. 1289–1294, IEEE, 2015.

[57] T. Kusnur, S. Mukherjee, D. M. Saxena, T. Fukami, T. Koyama, O. Salzman, and M. Likhachev, "A planning framework for persistent, multi-MILP coverage with global deconfliction," *arXiv preprint arXiv:1908.09236*, 2019.

[58] M. Charitidou, "Multi-robot path planning for persistent coverage tasks with performance guarantees," Master's thesis, Delft Center for Systems & Control, 2019.

[59] J. Finke and K. M. Passino, "Stable cooperative vehicle distributions for surveillance," *Journal of Dynamic Systems, Measurement and Control, Transactions of the ASME*, vol. 129, no. 5, pp. 597–608, 2007.

[60] J. R. Oliveira, R. Calvo, and R. A. Romero, "Integration of virtual pheromones for mapping/exploration of environments by using multiple robots," *Proceedings of the IEEE RAS and EMBS International Conference on Biomedical Robotics and Biomechatronics*, pp. 835–840, 2014.

[61] J. M. Palacios-Gasós, E. Montijano, C. Sagüés, and S. Llorente, "Distributed coverage estimation and control for multirobot persistent tasks," *IEEE Transactions on Robotics*, vol. 32, no. 6, pp. 1444–1460, 2016.

[62] C. Huang, W. Li, C. Xiao, B. Liang, and S. Han, "Potential field method for persistent surveillance of multiple unmanned aerial vehicle sensors," *International Journal of Distributed Sensor Networks*, vol. 14, no. 1, 2018.

[63] M. A. Batalin and G. S. Sukhatme, "The analysis of an efficient algorithm for robot coverage and exploration based on sensor network deployment," in *Proceedings of the IEEE International Conference on Robotics and Automation*, pp. 3478–3485, 2005.

[64] A. Mellone, G. Franzini, L. Pollini, and M. Innocenti, "Persistent coverage control for teams of heterogeneous agents," in *Proceedings of the IEEE Conference on Decision and Control*, pp. 2114–2119, 2018.

[65] G. Chmaj and H. Selvaraj, "Distributed processing applications for UAV/drones: a survey," in *Progress in Systems Engineering*, pp. 449–454, Springer, 2015.

[66] J. Liu, M. Chu, and J. E. Reich, "Multitarget tracking in distributed sensor networks," *IEEE Signal Processing Magazine*, vol. 24, no. 3, pp. 36–46, 2007.

[67] H. Choset, "Coverage for robotics - A survey of recent results," *Annals of Mathematics and Artificial Intelligence*, vol. 31, no. 1-4, pp. 113–126, 2001.

[68] T. Cabreira, L. Brisolara, and P. R Ferreira, "Survey on coverage path planning with unmanned aerial vehicles," *Drones*, vol. 3, no. 1, p. 4, 2019.

[69] R. Jans and Z. Degraeve, "Meta-heuristics for dynamic lot sizing: A review and comparison of solution approaches," *European journal of operational research*, vol. 177, no. 3, pp. 1855–1875, 2007.

[70] A. Farinelli, L. Iocchi, and D. Nardi, "Multirobot systems: a classification focused on coordination," *IEEE Transactions on Systems, Man, and Cybernetics, Part B*, vol. 34, no. 5, pp. 2015–2028, 2004.

[71] L. E. Parker, "Distributed intelligence: Overview of the field and its application in multi-robot systems.," in *AAAI Fall Symposium: Regarding the Intelligence in Distributed Intelligent Systems*, pp. 1–6, 2007.

[72] H. Abbas, S. Shaheen, and M. Amin, "Organization of multi-agent systems: An overview," *International Journal of Intelligent Information Systems*, vol. 4, no. 3, pp. 46–57, 2015.

[73] B. Horling and V. Lesser, "A survey of multi-agent organizational paradigms," *Knowledge Engineering Review*, vol. 19, no. 4, pp. 281–316, 2004.

[74] A. Tate, "Generating project networks," in *Proceedings of the 5th international joint conference on Artificial intelligence-Volume 2*, pp. 888–893, 1977.

[75] R. E. Fikes and N. J. Nilsson, "Strips: A new approach to the application of theorem proving to problem solving," *Artificial intelligence*, vol. 2, no. 3-4, pp. 189–208, 1971.

[76] R. I. Brafman and C. Domshlak, "From one to many: Planning for loosely coupled multi-agent systems.," in *ICAPS*, vol. 8, pp. 28–35, 2008.

[77] A. Torreño, E. Onaindia, and Ó. Sapena, "FMAP: Distributed cooperative multi-agent planning," *Applied Intelligence*, vol. 41, no. 2, pp. 606–626, 2014.

[78] M. Tambe, "Towards flexible teamwork," *Journal of artificial intelligence research*, vol. 7, pp. 83–124, 1997.

[79] K. Geihs, "Engineering challenges ahead for robot teamwork in dynamic environments," *Applied Sciences*, vol. 10, no. 4, p. 1368, 2020.

[80] P. R. Cohen and H. J. Levesque, "Teamwork," *Nous*, vol. 25, no. 4, pp. 487–512, 1991.

[81] B. J. Grosz and S. Kraus, "Collaborative plans for complex group action," *Artificial Intelligence*, vol. 86, no. 2, pp. 269–357, 1996.

[82] H. Skubch, *Modelling and controlling of behaviour for autonomous mobile robots.* Springer Science & Business Media, 2012.

[83] M. Güzelsoy and T. K. Ralphs, "Duality for mixed-integer linear programs," *International Journal of Operations Research*, vol. 4, no. 3, pp. 118–137, 2007.

[84] M. Dror, "Note on the complexity of the shortest path models for column generation in vrptw," *Operations Research*, vol. 42, no. 5, pp. 977–978, 1994.

[85] C. Nilsson, "Heuristics for the traveling salesman problem," *Linkoping University*, vol. 38, pp. 00085–9, 2003.

[86] M. Desrochers, *An algorithm for the shortest path problem with resource constraints*. École des hautes études commerciales, Groupe d'études et de recherche en analyse des décisions, 1988.

[87] G. Righini and M. Salani, "Symmetry helps: Bounded bi-directional dynamic programming for the elementary shortest path problem with resource constraints," *Discrete Optimization*, vol. 3, no. 3, pp. 255–273, 2006.

[88] G. Righini and M. Salani, "New dynamic programming algorithms for the resource constrained elementary shortest path problem," *Networks: An International Journal*, vol. 51, no. 3, pp. 155–170, 2008.

[89] L. D. P. Pugliese and F. Guerriero, "A survey of resource constrained shortest path problems: Exact solution approaches," *Networks*, vol. 62, no. 3, pp. 183–200, 2013.

[90] A. Bettinelli, A. Ceselli, and G. Righini, "A branch-and-cut-and-price algorithm for the multi-depot heterogeneous vehicle routing problem with time windows," *Transportation Research Part C: Emerging Technologies*, vol. 19, no. 5, pp. 723–740, 2011.

[91] D. Feillet, "A tutorial on column generation and branch-and-price for vehicle routing problems," *4OR*, vol. 8, no. 4, pp. 407–424, 2010.

[92] A. Saxena, P. Bonami, and J. Lee, "Convex relaxations of non-convex mixed integer quadratically constrained programs: extended formulations," *Mathematical programming*, vol. 124, no. 1, pp. 383–411, 2010.

[93] A. Saxena, P. Bonami, and J. Lee, "Convex relaxations of non-convex mixed integer quadratically constrained programs: projected formulations," *Mathematical programming*, vol. 130, no. 2, pp. 359–413, 2011.

[94] M. W. Carter, "The indefinite zero-one quadratic problem," *Discrete Applied Mathematics*, vol. 7, no. 1, pp. 23–44, 1984.

[95] A. Billionnet, S. Elloumi, and M.-C. Plateau, "Improving the performance of standard solvers for quadratic 0-1 programs by a tight convex reformulation: The QCR method," *Discrete Applied Mathematics*, vol. 157, no. 6, pp. 1185–1197, 2009.

[96] S. Mallach, "Compact linearization for binary quadratic problems subject to assignment constraints," *4OR*, vol. 16, no. 3, pp. 295–309, 2018.

[97] R. Fortet, "Applications de l'algebre de boole en recherche opérationelle," *Revue Française de Recherche Opérationelle*, vol. 4, no. 14, pp. 17–26, 1960.

[98] P. L. Hammer and S. Rudeanu, *Boolean methods in operations research and related areas*, vol. 7. Springer Science & Business Media, 2012.

[99] F. Glover and E. Woolsey, "Further reduction of zero-one polynomial programming problems to zero-one linear programming problems," *Operations Research*, vol. 21, no. 1, pp. 156–161, 1973.

[100] F. Glover and E. Woolsey, "Converting the 0-1 polynomial programming problem to a 0-1 linear program," *Operations research*, vol. 22, no. 1, pp. 180–182, 1974.

[101] L. Liberti, "Compact linearization for binary quadratic problems," *4OR*, vol. 5, no. 3, pp. 231–245, 2007.

[102] W. P. Adams, R. J. Forrester, and F. W. Glover, "Comparisons and enhancement strategies for linearizing mixed 0-1 quadratic programs," *Discrete Optimization*, vol. 1, no. 2, pp. 99 – 120, 2004.

[103] F. Fioretto and W. Yeoh, "Distributed constraint optimization problems and applications: A survey," *Journal of Artificial Intelligence Research*, vol. 61, 02 2016.

[104] M. Yokoo, O. Etzioni, T. Ishida, and N. Jennings, *Distributed Constraint Satisfaction: Foundations of Cooperation in Multi-Agent Systems*. Springer-Verlag, 2001.

[105] D. A. Burke and K. N. Brown, "A comparison of approaches to handling complex local problems in dcop," in *Sixth International Workshop on Distributed Constraint Reasoning*, pp. 27–33, 2006.

[106] D. A. Burke and K. N. Brown, "Efficient handling of complex local problems in distributed constraint optimization," in *17th European Conference on Artificial Intelligence*, pp. 701–702, 2006.

[107] J. Davin and P. Modi, "Hierarchical variable ordering for multiagent agreement problems," in *Proceedings of the International Joint Conference on Autonomous Agents and Multiagent Systems*, pp. 1433–1435, 2006.

[108] S. Khanna, A. Sattar, D. Hansen, and B. Stantic, "An efficient algorithm for solving dynamic complex dcop problems," in *International Joint Conference on Web Intelligence and Intelligent Agent Technology*, vol. 2, pp. 339–346, 2009.

[109] C. Portway and E. H. Durfee, "Ordered multi-variable multi-constrained distributed constraint optimization framework," in *International Conference on Web Intelligence and Intelligent Agent Technology*, vol. 2, pp. 379–382, 2010.

[110] T. Grinshpoun, "Clustering variables by their agents," in *International Conference on Web Intelligence and Intelligent Agent Technology*, vol. 2, pp. 250–256, 2015.

[111] F. Fioretto, W. Yeoh, and E. Pontelli, "Multi-variable agents decomposition for dcops," in *Proceedings of the AAAI conference on artificial intelligence*, vol. 30, 2016.

[112] A. Gershman, A. Meisels, and R. Zivan, "Asynchronous forward bounding for distributed cops," *Journal of Artificial Intelligence Research*, vol. 34, pp. 61–88, 2009.

[113] W. Yeoh, A. Felner, and S. Koenig, "Bnb-adopt: An asynchronous branch-and-bound dcop algorithm," *Journal of Artificial Intelligence Research*, vol. 38, pp. 85–133, 2010.

[114] F. Vanderbeck, "On dantzig-wolfe decomposition in integer programming and ways to perform branching in a branch-and-price algorithm," *Operations Research*, vol. 48, no. 1, pp. 111–128, 2000.

Glossary

List of Acronyms

AOI	Area of Interest
C2	Command & Control
CCB	Complex Concurrent Bounding
ConcFB	Concurrent Forward-Bounding
CONOPS	Concept of Operations
CPA	Current Partial Assignment
DAI	Distributed Artificial Intelligence
DCOP	Distributed Constraint Optimization Problem
EMCON	Emission Control
ESPP	Elementary Shortest Path Problem
FCM	Fuzzy C-Means
HFVRP	Heterogeneous Fleet Vehicle Routing Problem
HTN	Hierarchical Task Network
ISR	Intelligence, Surveillance & Reconnaissance
LB	Lower Bound
MAP	Multi-Agent planning
MARP	Multi-Agent Reconnaissance Problem
MAS	Multi-Agent system
MDHVRP	Multi-Depot Heterogeneous Vehicle Routing Problem
MILP	Mixed Integer Linear Program
MIQCP	Mixed Integer Quadratically Constrained Program
MP	Master Problem
MRS	Multi-Robot System
MTSP	Multiple Traveling Salesmen Problem
POMDP	Partially Observable Markov Decision Process
QCLP	Quadratically Constrained Linear Program
QCR	Quadratic Convex Reformulation
RAS	Robotic & Autonomous Systems
RMP	Restricted Master Problem
SA	Situational Awareness

SARP	Single-Agent Reconnaissance Problem
SDP	Semidefinite Programming
SFB	Synchronous Forward Bounding
SP	Search Process
SP	Sub Problem
SPP	Shortest Path Problem
TSP	Traveling Salesman Problem
UAV	Unmanned Aerial Vehicle
UB	Upper Bound
VRP	Vehicle Routing Problem

List of Symbols

$\bar{\delta}_i^t$	Average uncertainty of adjacent sectors from sector $i \in \mathcal{N}$ at time t
β	Cut-off degree below which any degree is removed as not significant
β_k	Overused capacity for subordinate $k \in \mathcal{P}$
δ_i^t	Uncertainty in sector $i \in \mathcal{N}$ at time t
$\dot{r}_i^t$	Risk of sectors $i \in \mathcal{N}$ at time t
$\dot{R}_j$	Average risk increase of cluster $j \in \mathcal{M}$
ϵ	Clustering improvement threshold
γ	Degree belonging to the n^{th} percentile of all ordered degrees $u_{ij}\ \forall i \in \mathcal{N},\ j \in \mathcal{M}$
γ_i^t	Uncertainty dispersion to sector $i \in \mathcal{N}$ at time t
$\hat{b}_k$	Normalized capacity given as fractional area agent $k \in \mathcal{P}$ can service
$\hat{c}^{lb}$	Lower bound on the minimum reduced cost obtainable by extending label L
$\hat{c}_{jk}^{step}$	Added reduced cost by stepping to vertex $j \in \mathcal{V}$ with agent $k \in \mathcal{P}$ from label L
$\hat{c}_k$	Reduced cost for the sequence of label L
$\hat{J}_{ijk}$	Approximation of added reduced cost by including edge $\{i, j\}$ in sequence for agent $k \in \mathcal{P}$
$\hat{J}_{rk}$	Reduced cost if sequence $r \in \Omega$ is performed by agent $k \in \mathcal{P}$
$\hat{u}_j$	Lower bound on distance to vertex $j \in \mathcal{V}$
$\hat{u}_{ij}$	Interpolated degree of sector $i \in \mathcal{N}$ belonging to cluster $j \in \mathcal{M}$
$\hat{z}$	Lower bound on cycle distance
κ	Scaling factor used for various experiments
Λ_i	Set of supervisors $q \in \mathcal{C}$ that can perform task $i \in \mathcal{M}$
λ_i	Duals for the sectors $i \in \mathcal{N}$
λ_k	Duals for the agents $k \in \mathcal{P}$
$\mathcal{E}$, $\mathcal{E}_k$	Set of graph edges (of agent $k \in \mathcal{P}$)
$\mathcal{M}$	Set of clusters (=tasks)

$\mathcal{M}^s$	Set of shared clusters with a group of supervisors $\mathcal{C}$
$\mathcal{M}_q$, $\mathcal{M}_q^l$	Set of (local) clusters of supervisor $q \in \mathcal{C}$
$\mathcal{N}$	Set of sectors
$\mathcal{P}$	Set of agents (subordinates)
$\mathcal{P}_q$	Set of subordinates for supervisor $q \in \mathcal{C}$
$\mathcal{U}_k$	Linearly ordered set composing a sequence of vertices and cumulative distances $\{j, d_k\}\ \forall i \in \mathcal{V},\ k \in \mathcal{P}$, of label L
$\mathcal{V}$, $\mathcal{V}_k$	Set of graph vertices (of agent $k \in \mathcal{P}$)
$\mathcal{W}$	Set of unvisited vertices of label L
Ω	Set of all possible sequences
Ω^r	Restricted set of sequences
ρ	A weight balancing the risk and risk increase objective cost
σ^h	Heterogeneity standard deviation
$\tilde{u}_{ij}$	Discounted degree of sector $i \in \mathcal{N}$ belonging to cluster $j \in \mathcal{M}_q$ of supervisor $q \in \mathcal{C}$
$\{y_i, x_i\}$	Cartesian coordinates for sector $i \in \mathcal{N}$
A_i^f	Normalized size of task $i \in \mathcal{M}$
A^s	Standard squared sector area ($= d_s^2$) for every sector in $\mathcal{N}$
A_j	Total area of cluster $j \in \mathcal{M}$
A_{ik}	A constant indicating if vertex $i \in \mathcal{V}$ can be visited by agent $k \in \mathcal{P}$
a_{ir}	Constant indicating if sector $i \in \mathcal{N}$ is included in sequence $r \in \Omega$
a_{jk}	Total utility of agent $k \in \mathcal{P}$ performing task $j \in \mathcal{M}$
b_k	Absolute capacity given as area agent $k \in \mathcal{P}$ can service
c_k^z	Cycle cost for the sequence of label L
c_{jk}^{step}	Added cost by stepping to vertex $j \in \mathcal{V}$ with agent $k \in \mathcal{P}$ from label L
C_j	Centroid of cluster $j \in \mathcal{M}$
c_k	Total cost for the sequence of label L
c_{ij}	Distance between vertex i and $j \in \mathcal{V}$
D	Constant influencing dynamic increase of uncertainty by dispersion
D_{ij}^f	Normalized dependency between tasks $i, j \in \mathcal{M}$
d^s	Standard sector width
d_k	Cumulative distance to the current vertex of label L
D_q	Compiled variable domain of supervisor $q \in \mathcal{C}$
e	Inefficiency penalty for capacity violation
e_k	Flight time in minutes for agent $k \in \mathcal{P}$
F_j	Centre of mass of cluster $j \in \mathcal{M}$
J_{rk}	Cost if sequence $r \in \Omega$ is performed by agent $k \in \mathcal{P}$
L, L'	A label used during forward labeling
l_{jk}^1	Linear score indicating the suitability of agent $k \in \mathcal{P}$ dealing with the risk in task $j \in \mathcal{M}$

l^2_{jk}	Linear score indicating the suitability of agent $k \in \mathcal{P}$ dealing with the risk increase in task $j \in \mathcal{M}$
m	Fuzzifier used during Fuzzy C-Means (FCM)
M_i	Terrain traversability in sector $i \in \mathcal{N}$
n^b	Factor used to scale the normalized capacity
n^c	Maximum number of clusters per agent
P_i	Operator priority of sector $i \in \mathcal{N}$
r_i^t	Risk of sectors $i \in \mathcal{N}$ at time t
R_j	Total risk of cluster $j \in \mathcal{M}$
S	Constant influencing static increase of uncertainty
s, s_k	Source node in $\mathcal{V}$ (for agent $k \in \mathcal{P}$)
u_i, u_{ik}	Distance of the sequence to vertex $i \in \mathcal{V}$ (for agent $k \in \mathcal{P}$)
u_{ij}	Degree of sector $i \in \mathcal{N}$ belonging to cluster $j \in \mathcal{M}$
u_{ik}, u_{iq}	Degree with which sector $i \in \mathcal{N}$ belongs to agent $k \in \mathcal{P}$ (or subordinate $q \in \mathcal{C}$)
u_i^t	Uncertainty reduction by agent presence in sector $i \in \mathcal{N}$ at time t
v_i, v_{ik}	Remaining distance of the sequence from vertex $i \in \mathcal{V}$ (for agent $k \in \mathcal{P}$)
v_k	Speed in m/s for agent $k \in \mathcal{P}$
w	A weight vector used to emphasize specific features during clustering
X_i	Feature vector for sector $i \in \mathcal{N}$
x_{ij}, x_{ijk}	Decision variable if vertex j is visited after $i \in \mathcal{V}$ (for agent $k \in \mathcal{P}$)
x_{ik}	Decision variable if task $i \in \mathcal{M}$ is performed by subordinate $k \in \mathcal{P}$
x_{rk}	Decision variable if sequence $r \in \Omega$ is performed by agent $k \in \mathcal{P}$
z, z_k	Cycle distance of the sequence (for agent $k \in \mathcal{P}$)