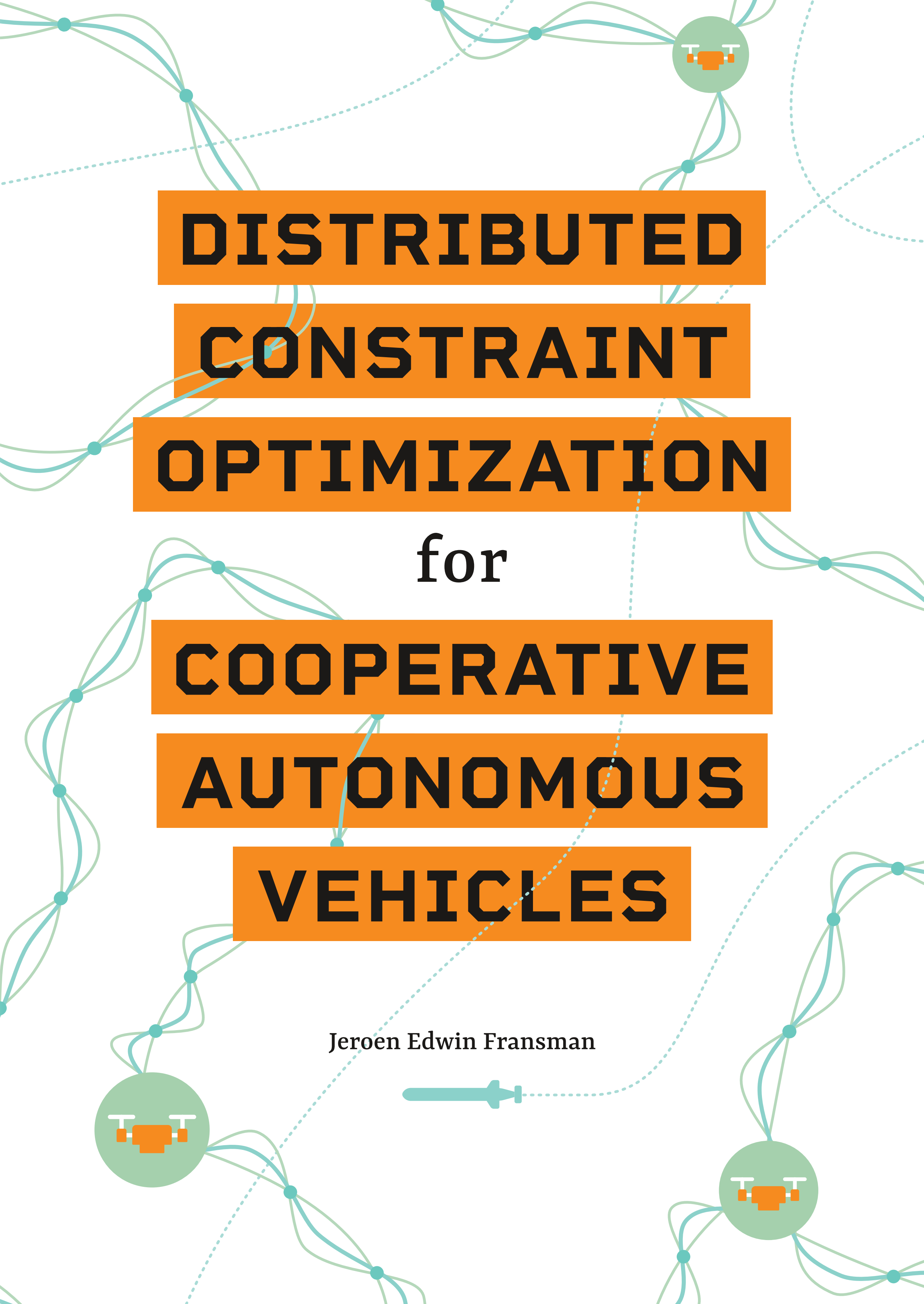


DISTRIBUTED CONSTRAINT OPTIMIZATION for COOPERATIVE AUTONOMOUS VEHICLES

Jeroen Edwin Fransman

DISTRIBUTED CONSTRAINT OPTIMIZATION

FOR COOPERATIVE AUTONOMOUS VEHICLES

Proefschrift

ter verkrijging van de graad van doctor
aan de Technische Universiteit Delft,
op gezag van de Rector Magnificus prof. dr. ir. T.H.J.J. van der Hagen;
voorzitter van het College voor Promoties,
in het openbaar te verdedigen op
maandag 14 november 2022 om 17:30 uur

door

Jeroen Edwin FRANSMAN

Master of Science in Systems & Control, Technische Universiteit Delft, Nederland,
geboren te Rotterdam, Nederland.

Dr. ir. H. S. Dol van de Nederlandse Organisatie voor toegepast-natuurwetenschappelijk onderzoek (TNO) heeft in belangrijke mate aan de totstandkoming van het proefschrift bijgedragen.

Trefwoorden: DCOP, autonomous agents, multi-agent systems

Ontwerp omslag: Amanda Nedermeijer

ISBN 978-94-6384-384-3

Een digitale versie van dit proefschrift is beschikbaar via `https://repository.tudelft.nl/`.

CONTENTS

Summary

Motivation

After the Second World War, chemical warfare agents and munitions were dumped in the Baltic Sea and the North Sea. Due to several decades of exposure to seawater, their containers are corroding and their contents are being released into the water. To assess the severity of the environmental consequences, it is important that the chemical warfare agents are located and their condition is investigated as soon as possible. Unfortunately, searching both seas requires a significantly long time to perform. In order to reduce this time, the search could be performed by Autonomous Underwater Vehicles (AUVs) that search multiple areas in parallel. A disadvantage of using AUVs is that during operation the communication with the AUVs is very limited due to the attenuation of radio signals in seawater. This makes it hard to re-distribute their assigned areas when an AUV malfunctions or has decreased scanning performance. The goal of this thesis is to develop algorithms that can be applied during underwater operations to allow AUVs to optimize their actions based on a global objective function without centralized communications.

DCOP framework

The attenuation of radio signals in seawater severely impacts underwater communication capabilities. For this reason, a distributed approach is required to optimize the actions of the AUVs. The search problem can be modeled within the Distributed Constraint Optimization Problem (DCOP) framework to be able to explicitly define both computational agents and their communications. Traditional DCOPs are not able to efficiently model real-world applications because of the definitions of the domains of the variables. A domain defines all possible values that can be assigned to the variables. In a DCOP, these domains are discrete, while real-world problems are typically characterized by continuous domains. We model both benchmark problems and real-world problems with continuous domains within the Continuous DCOP (C-DCOP) framework to apply to AUV operations. This preserves the flexibility of modeling inherent in a DCOP while removing the limitations imposed by the discrete domain definitions.

C-DCOP solvers

Two C-DCOP algorithms are presented in this thesis. Both algorithms operate directly on the C-DCOPs and do not require *a priori* discretization of the continuous domains of the (scalar) variables. The Compression-DPOP (C-DPOP) algorithm discretizes the domain of each of the variables at every iteration and *compresses* the domains. Compressing

refers to the procedure of updating the upper and lower bound of a domain centered around an intermediate solution. This process focuses on regions of the search space that have high utility values. To be able to adjust the communicational requirements of the algorithm, the size of the messages exchanged during optimization is set through a parameter. This property of the algorithm is beneficial for applications that suffer from low communication bandwidths. The Distributed Bayesian (D-Bay) algorithm leverages Bayesian optimization to solve C-DCOPs without any need for discretization. All agents model the effect of their variables on the global utility as a Gaussian process. The properties of the utility functions are modeled by an appropriate selection of the kernels of the Gaussian processes. The models are used to trade off exploration and exploitation of the search space to efficiently solve C-DCOPs. When the Lipschitz constant of the utility functions is known, the D-Bay algorithm is guaranteed to converge to the global optimum.

SIMULATION AND EXPERIMENTAL RESULTS

In this thesis, besides benchmark problems, results from both high-fidelity simulations and real-world experiments are given for real-world multi-agent search problems. A mine countermeasures operation is simulated in which AUVs update their search areas during the search based on sonar performance. The sonar performance is defined as the effective scan range of the side-scan sonar sensors. Assigned areas are re-distributed to optimize a global objective based on metrics relating to the expected time of completion and the level of confidence that all mine-like objects within the area have been detected. The results show the potential of *in situ* optimization of the search areas of the AUVs to improve their efficiency. Moreover, real-world experimental results are presented for a multi Unmanned Aerial Vehicle (UAV) search problem. The UAVs are equipped with a low-bandwidth communication mesh network to mimic the communication capabilities of underwater vehicles. The successful application of the D-Bay algorithm to real-world autonomous vehicles shows the compatibility of the algorithm with underwater operations.

CONCLUSIONS

The search for dangerous objects on the seabed should become more efficient to find and assess these objects promptly. By improving the autonomy of AUVs, the search efficiency can be increased through the cooperative optimization of their actions during the operation. The research in this thesis contributes to this strategy through the developed algorithms and their applicability to real-world problems. The simulated problems show the increase of search efficiency of AUVs during mine countermeasure operations. In addition, the real-world experiments with unmanned aerial vehicles demonstrate the applicability to multi-agent systems that suffer from low communication bandwidths.

SAMENVATTING

MOTIVATIE

In de nasleep van de Tweede Wereldoorlog zijn er (onderdelen van) chemische wapens en munitie in de Baltische Zee en de Noordzee gedumpt. De structurele integriteit van de containers is aan het afnemen door decennialange blootstelling aan zeewater. Om de gevolgen voor het milieu in kaart te brengen is het belangrijk om zo spoedig mogelijk de containers terug te vinden en hun toestand te evalueren. Om beide zeeën in zo kort mogelijke tijd af te zoeken, kunnen gebieden door meerdere Autonomous Underwater Vehicles (AUVs) gezamenlijk worden afgezocht. Een nadeel van het gebruik van AUVs is dat de communicatie met de AUVs zeer gelimiteerd is door de absorptie van radiosignalen in zeewater. Dit bemoeilijkt het aansturen van de AUVs als een AUV een storing heeft of als de sensoren niet naar behoren werken. Het doel van dit proefschrift is om algoritmes te ontwikkelen die gebruikt kunnen worden in autonome voertuigen. Met name in AUVs om hun acties te kunnen optimaliseren op basis van een globale doelfunctie zonder gecentraliseerde communicatie.

DCOP FRAMEWORK

De absorptie van radiosignalen in zeewater heeft een extreem negatieve invloed op onderwatercommunicatie. Hierdoor is een gedistribueerde aanpak nodig om de acties van de AUVs te optimaliseren. Het zoekprobleem kan worden gemodelleerd binnen het Distributed Constraint Optimization Problem (DCOP) framework om zowel de computationele agenten als hun onderlinge communicatie expliciet te definiëren. Traditionele DCOPs zijn niet in staat toepassingen in de echte wereld op een efficiënte manier te modelleren door de definitie van de domeinen. Een domein beschrijft alle mogelijke waardes die een variabele kan aannemen. In een DCOP zijn deze waardes discreet, terwijl toepassingen in de echte wereld gekarakteriseerd worden door waardes die continu zijn. In dit proefschrift zijn zowel benchmark problemen als echte wereld problemen met continue domeinen gemodelleerd in het Continuous DCOP (C-DCOP) framework. Hierdoor is de flexibiliteit van het modelleren, inherent aan een DCOP, behouden terwijl de beperkingen van de discrete domeinen zijn weggenomen.

C-DCOP SOLVERS

Twee C-DCOP algoritmes zijn geïntroduceerd in dit proefschrift. Beide algoritmes maken direct gebruik van de C-DCOPs en vereisen geen *a priori* discretisatie van de continue domeinen van de (scalar) variabelen. Het Compression-DPOP (C-DPOP) algoritme discretiseert het domein van elke variabele tijdens elke iteratie en comprimeert de domeinen. Comprimeren is de procedure van het aanpassen van de boven- en ondergrens van een domein op basis van een tussentijdse oplossing. Via dit proces focust het algoritme zich op gebieden van de zoekruimte met hoge waarde. Om de benodigde hoeveelheid communicatie van het algoritme aan te kunnen passen, kan de grootte van de berichten (die worden uitgewisseld tijdens optimalisatie) worden bepaald door middel van een parameter. Deze eigenschap van het algoritme maakt het geschikt voor toepassingen met beperkte communicatie bandbreedte. Het Distributed Bayesian (D-Bay) algoritme gebruikt Bayesiaanse optimalisatie om C-DCOPs op te lossen zonder discretisatie van de domeinen. Alle agenten modelleren de effecten van hun variabelen op het globale resultaat op basis van Gaussische processen. De eigenschappen van de lokale functies worden gemodelleerd door een selectie van de kernels van deze Gaussische processen. De modellen worden gebruikt om exploratie en exploitatie van de zoekruimte tegen elkaar af te wegen. Het D-Bay algoritme convergeert naar het globale optimum als de Lipschitz constanten van de lokale functies bekend zijn.

SIMULATIE EN EXPERIMENTELE RESULTATEN

In dit proefschrift zijn, naast resultaten van benchmark problemen, resultaten van high-fidelity simulatieomgevingen en echte wereld experimenten gegeven voor multi-agent zoekproblemen. Een zoekoperatie naar zeemijnen is gesimuleerd waarbij AUVs hun zoekgebied aanpassen op basis van de reikwijdte van de sonar. De reikwijdte van de sonar is gedefinieerd als de afstand waarop objecten kunnen worden gedetecteerd. Het globale doel is gerelateerd aan de verwachtte eindtijd van de operatie en het zekerheidsniveau dat alle zeemijn-achtige objecten in het zoekgebied zijn gevonden. De toegewezen zoekgebieden van de AUVs werden herverdeeld om het globale doel te optimaliseren. De resultaten laten een toename van efficiëntie zien van *in situ* optimalisatie van de zoekgebieden van de AUVs. Hiernaast zijn echte wereld experimenten uitgevoerd voor een multi Unmanned Aerial Vehicle (UAV) zoekprobleem. De UAVs waren uitgerust met een laag-bandbreedte communicatie mesh-netwerk om de communicatiemogelijkheden van onderwatervoertuigen na te bootsen. Tijdens de experimenten werd het D-Bay algoritme gebruikt om de toepasbaarheid van het algoritme op autonome voertuigen in de echte wereld aan te tonen, in het bijzonder op onderwatervoertuigen.

CONCLUSIES

Het zoeken naar gevaarlijke objecten op de zeebodem moet efficiënter worden om deze objecten te kunnen vinden en te beoordelen in afzienbare tijd. Door de autonomie van AUVs te verbeteren kunnen de zoekgebieden via coöperatieve optimalisatie efficiënter worden afgezocht. Het onderzoek in dit proefschrift draagt bij aan deze strategie door de ontwikkeling van algoritmes en hun toepasbaarheid op echte wereld problemen. De gesimuleerde problemen laten de AUVs een toegenomen efficiëntie zien tijdens een zoekoperatie naar zeemijnen. Daarnaast laten de echte wereld experimenten met UAVs de toepasbaarheid zien op problemen met multi-agent systemen met beperkte communicatie bandbreedte.

1 INTRODUCTION

1.1. BACKGROUND

There is a lurking threat beneath the waves of the North Sea that puts both the animals living in the sea and the people living around it in danger. This threat dates back to the end of the Second World War when approximately 13 000 metric tonnes of Chemical Warfare Agents (CWAs) were discarded at sea [10]. Even though these CWAs were not used during the Second World War, they were stockpiled in large quantities. After the Second World War ended the Allied forces decided to dispose of these types of weapons. To neutralize these CWAs it was decided to dump the CWAs into the Baltic sea, the Skagerrak Strait, and the North Sea by either throwing the CWAs overboard or sinking old ships filled with CWAs [2]. At that time it was considered a suitable manner of disposal as it was thought that the oceans would either neutralize or absorb the CWAs. However, it is now known that these effects were overestimated and a large part of the CWAs remain harmful up to the present day. Partly as a result of these facts, the final report of the NATO task group AVT-115 [18] concluded that the disposal of weapons at sea is no longer acceptable.

Recognizing the severity of the problem, the Chemical Munitions Search and Assessment (CHEMSEA) project was started to produce detailed maps of several known dumpsites, assess the toxicity of CWA degradation products to aquatic life, and develop a model to predict the magnitude and direction of leakage events. The findings have been released in an extensive report [3] after the project ended in 2014. In the report, it is described that Denmark registered 44 incidents related to CWAs in the years between 2003 and 2012. Additionally, Andrulewicz [2] and Knobloch *et al.* [10] describe reports of white phosphorus (from incendiary munitions) found on beaches, encounters of fishermen and maritime workers with chemical warfare materials, and serious injuries related to CWAs in Sweden, Germany, and Poland. Andrulewicz [2] adds that when the CWAs are moved from the seabed either accidentally (after being washed up on a beach) or intentionally (during bottom trawling) these agents do pose a significant risk. While this problem will remain a hazard for a long time, despite the high amounts of CWAs, it will only pose local threats if the current levels of corrosion and leakage remain constant.

However, if the leakage increases (due to the extensive corrosion of the containers) the problem regarding CWA concentrations becomes severe. As detailed by Jurczak *et al.* [9], there is a high chance that this second scenario will become a reality as it has been 70 years since the dumping actions, while the expected durability of the containment barrels is estimated at 50-60 years. To assess the risk to the marine environment and coastal communities, the detection and classification of the dumped CWAs was an important first step. Based on these findings, by CHEMSEA, the effects of the CWAs (and their derivatives) found within the water column and the seafloor sediments were investigated. The CHEMSEA project focused its surveys on the Gotland Deep. This official dumpsite covers an area of 1760 km^2. The area was partitioned into 40 subareas of 13 by 3-5 km. The surveys classified 17 000 pieces of munition (of which 50 % containing CWAs) and 33 wrecks that potentially contain CWAs. An example of a side-scan sonar image of a wreck is shown in Figure 1.1.

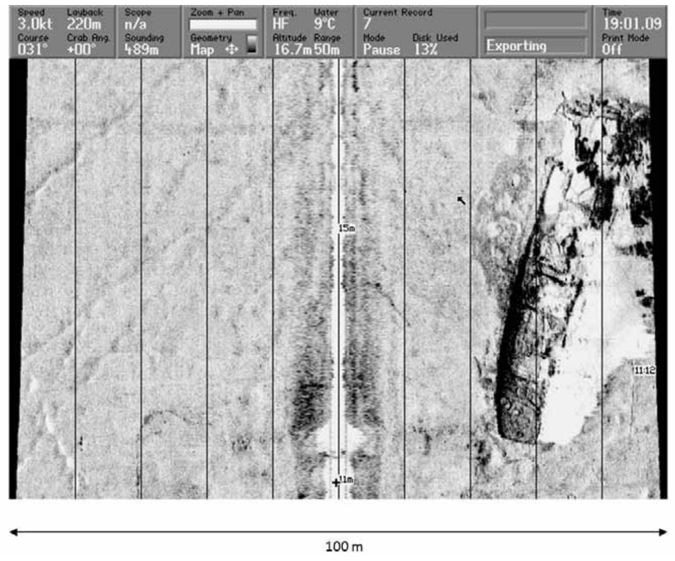


Figure 1.1.: Sonar image of a wreck in the Bornholm Basin (MERCW 2006; personal communication, V. Paka, via Knobloch *et al.* [10]. All rights are reserved).

Most dumpsites contained derivatives or oxidation products of CWAs. These degradation products retain similar toxic properties while having lower breakdown rates and are therefore considered persistent pollutants. The CHEMSEA report notes that one-third of all samples that were collected, contained at least one trace of CWAs and at one site two thirds of the samples contained arsenic.

In addition to these alarming findings, CHEMSEA has confirmed that munitions were thrown overboard during transit toward dumpsites. During surveys, Bełdowski *et al.* [4] found an undocumented dumpsite at the Gdańks Deep. This means that an unknown quantity of CWAs remains to be discovered in addition to the large amount of CWAs at the official dumpsites. Both CHEMSEA and Andrulewicz [2] advise performing sonar surveys using side-scan sonars, multibeam echo sounders and sub-bottom profilers to search for these undocumented sites. Knobloch *et al.* [10] propose periodical (sampling and scanning) surveys of the suspected dumpsites to monitor CWA concentration within the water column.

The mapping of the (un)documented dumpsites and creating a global database are goals of the International Dialogue on Underwater Munitions (IDUM)[1] organization. The IDUM states it is paramount to eradicate the point-source emitters of pollution because the pollution plumes of these point sources could merge and cause great problems to the ecosystem. This concern is shared by Jānis Kuzins (SDK Dzimtene) in a recent petition[2] sent to the European Parliament in which he urges for the removal of all chemical weapons from the Baltic Sea to prevent another *Chernobyl*. Currently, most of the surveys are done through the towing of a side-scan sonar behind a (support) vessel, as shown in the illustration in Figure 1.2.

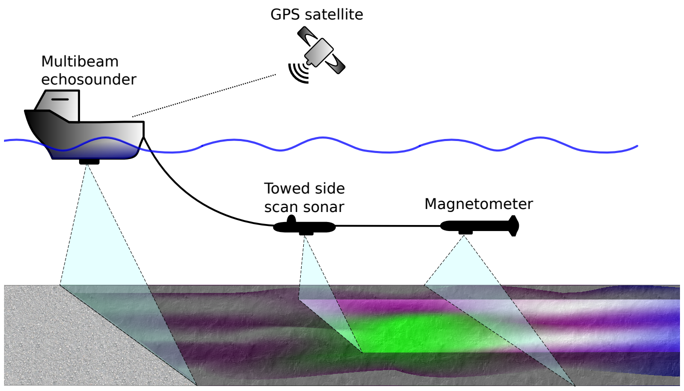


Figure 1.2.: Illustration of the survey equipment configuration used within the CHEMSEA project (image adapted from Bełdowski *et al.* [3]).

The benefit of this scanning method is that the data can be collected in real-time. A major disadvantage is that this requires large amounts of time to scan the documented dumpsites. For this reason, scanning all areas with a high probability of containing undocumented dumpsites is unfeasible.

A promising alternative is the usage of Autonomous Underwater Vehicles (AUVs) to autonomously scan the seabed and to record the sensor information for later processing on a support vessel. At present, various commercial companies exist that offer AUVs equipped with side-scan sonars and other environmental sensors. One example is the Lightweight Autonomous Underwater Vehicle (LAUV) of OceanScan[3] which is shown in Figure 1.3.

[1] https://underwatermunitions.org/
[2] https://www.europarl.europa.eu/doceo/document/PETI-CM-658942_EN.docx
[3] https://www.oceanscan-mst.com/

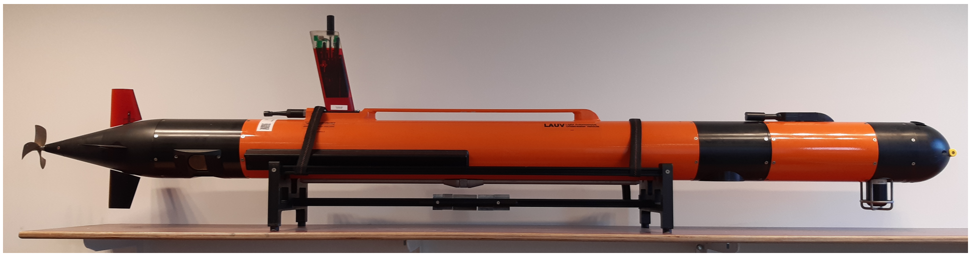


Figure 1.3.: Lightweight Autonomous Underwater Vehicle of OceanScan-MST.

Ideally, such AUVs would operate without human intervention for extended periods of time (8 hours or more) and multiple AUVs would be launched simultaneously to search in parallel. Not only would this decrease the amount of time required to survey an area, but it also increases the safety of the crew on board the support vessel.

Objects found by the AUVs can be revisited by the support vessel and classified by a human operating through visual inspection. These visual inspections are often performed by specialized underwater vehicles that are remotely controlled. Despite the obvious benefits, the usage of AUVs has its challenges. The environment in which the AUVs operate is largely unknown and the communication between the support vessel and the AUVs is very limited. This requires a certain level of autonomy from the AUVs concerning navigation, obstacle avoidance, information sharing, and cooperation with other AUVs. Advances in any of these aspects of autonomy can significantly increase the capabilities and efficiency of the AUVs. Especially in terms of the efficiency of the AUVs, there is much to be gained. Typically, the trajectories of the AUVs are defined before the AUVs are launched from the support vessel. An illustration of an AUV scanning after being launched from a support vessel is shown in Figure 1.4.

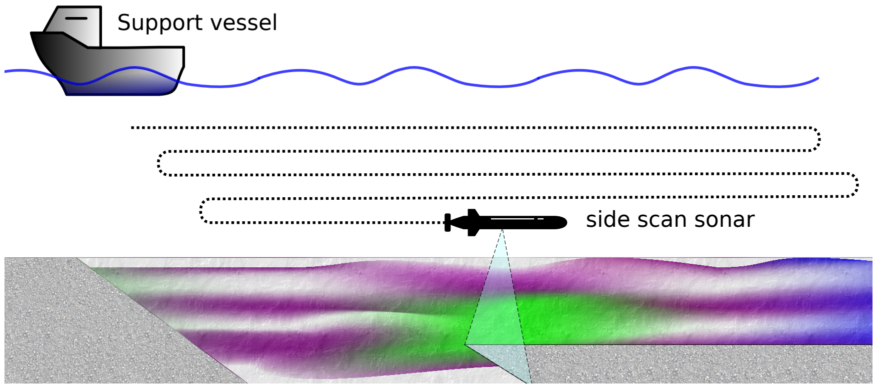


Figure 1.4.: Illustration of an AUV executing a scan while traversing a trajectory.

After the AUVs finish these trajectories, they are reclaimed by the support vessel and the sensor information is downloaded and assessed. When the sensor performance of an AUV is less than expected, this will result in gaps within the high-resolution seabed maps. These high-resolution maps are used to determine whether the area contains any objects such as barrels containing CWAs or even sea mines. Depending on the size of these gaps, the commander of the support vessel will need to trade off the time required to (re)visit these gaps (by relaunching an AUV) and the risks of undiscovered objects on the seabed.

An increase in the autonomy of the AUVs can overcome this problem by setting a goal of the operation for all AUVs, such as 'scan the area as fast as possible and localize objects with sufficient accuracy'. During operation, all AUVs could assess their performance and adjust their search trajectories accordingly. While this will decrease the number and size of the gaps, this will not result in a time-efficient solution as the AUV will still have to scan the entire assigned area by itself. Cooperation between the AUVs would allow for redistribution of their assigned subareas based on their actual sensor performance. AUVs with good sensor performance could scan a larger area compared to AUVs that have a decreased performance. In conclusion, close collaboration between the AUVs would ensure time-efficient trajectories for adequately scanning the entire area.

1.2. PROBLEM FORMULATION

The problem of coordinating multiple autonomous vehicles to survey an area of interest is commonly modeled as a coverage path planning problem [6]. Within this problem, a path for every sensor is planned to cover a certain area. The coverage path planning problem is related to various other problems. Examples of similar problems include task scheduling [21], mobile sensor coordination [25], hierarchical task network mapping [22], and cooperative search [1]. Typically, in order to solve a coverage path planning problem, the search area is divided into discrete segments, as was done in the work of Zhang *et al.* [24]. A similar approach is taken by Popa *et al.* [20] for multiple AUVs. Both approaches require the discretization of the search area. A common problem of discretization is the exponential memory requirement of the algorithms when the total survey area increases [12]. This problem can be alleviated by the use of specific methods, as shown by Low *et al.* [13] and Meliou *et al.* [16] through approximation and a dynamic programming approach [5].

An important factor to take into account in solving the path planning problem for AUVs is the communication restriction. This restriction is caused by the attenuation of radio signals in seawater and severely limits both communication range and bandwidth [8]. Due to the dangers involved in the search for CWAs or sea mines a support vessel is typically located at a significant distance from the search area. The limited communication range is therefore often the leading factor in the infeasibility of *in-situ* centralized optimization by a support vessel. Communication between AUVs is considered to be sufficient for cooperation because the distance between the AUVs is not as extensive during the operation. However, a centralized approach where a single AUV receives all information and optimizes the actions of all AUVs is considered impractical due to the low data rates of the underwater modems. For these reasons, in this Ph.D. thesis, centralized optimization approaches are considered infeasible and distributed approaches are investigated. Applying a distributed approach in practice is often problematic due to complications that arise from limitations in communication and/or computation. Especially in a cooperative search performed by AUVs, these complications become apparent as the computational hardware needs to be small and lightweight to avoid negative effects on the endurance of the AUVs.

The Distributed Constraint Optimization Problem (DCOP) framework is well suited to model the above-mentioned distributed problems (as detailed in Gershman *et al.* [7], Meisels [15], Modi *et al.* [17], Petcu *et al.* [19], and Yeoh *et al.* [23]). This is because the DCOP framework explicitly considers both computational agents and inter-agent communication. Within the DCOP framework, a problem is defined by variables and utility functions that are aggregated into a single objective function. Within the DCOP framework, variables are constrained by their domains making it suitable for problems that are (input) constrained. In other words, a domain of a variable is the set of all values that are allowed to be assigned to that variable. These domains are considered to be finite and discrete within a DCOP, while real-world problems are typically characterized by finite continuous domains. For that reason, real-world problems should be modeled within the Continuous DCOP (C-DCOP) framework, which is an extension of DCOP for finite continuous domains.

This thesis focuses on developing C-DCOP solvers that can be deployed on real-world problems and actual hardware that is subject to computational and communicational constraints.

1.3. RESEARCH OBJECTIVES

In this section, the research goals and the research approach are elaborated. The research described in this thesis is funded by the Netherlands Defence Academy (NLDA). The research is done in close collaboration between the Delft University of Technology, the Netherlands Organization for Applied Scientific Research (TNO), and the Netherlands Defence Academy (NLDA).

1.3.1. RESEARCH GOALS AND APPROACH

This thesis focuses on the cooperation between autonomous vehicles to efficiently and effectively complete an objective. The application focus of the research is on autonomous vehicles for real-world problems. This involves taking limitations (such as computational and communicational limitations) into account that are caused by real-world situations. Especially, the communication requirements of the algorithms should be taken into careful consideration to reduce their negative impact on the execution of the algorithms.

The focus of the research is to further develop C-DCOP solvers that can be applied to various real-world problems. The goal of the research is to quantify the expected vehicle performance via utility functions. Research questions to be answered in this thesis are:

- Can an autonomous platform quantify the global utility of local actions through collaboration?
- How can multi-vehicle performance functions be derived from a high-level performance function? Can these functions be integrated in terms of utility with respect to desired performance?
- What are suitable optimization strategies for in-mission replanning to reach the mission goals?
- What are the relevant methods for evaluating the research results?

1.3.2. CONTRIBUTIONS OF THIS THESIS

This thesis contributes to the state-of-the-art in modeling and cooperative optimization of multi-agent systems. The main contribution is the further development of C-DCOP solvers. These solvers can be applied to numerous C-DCOPs and are well-suited to be applied to real-world problems thanks to their focus on computational and communicational efficiency. The contributions can be summarized as:

- The Compression-DPOP (C-DPOP) algorithm is presented to solve C-DCOPs while taking the computation time and memory requirement into account. Especially in problems in dynamic environments and close collaboration between agents, these bounds need to be taken into account to ensure effective cooperation. Instead of discretizing the continuous domains into discrete domains and applying the traditional DCOP solvers, C-DPOP discretizes the domains iteratively. A major benefit is that the size of the discrete domains can be selected through a parameter, thereby explicitly limiting the computational and memory requirements of the algorithm. Combined with the any-time property (which yields an improved solution after each iteration) this algorithm is highly suitable for application to real-world problems.
- The C-DPOP algorithm is applied to a Mine Counter-Measures (MCM) operation. Within this operation, multiple AUVs cooperatively search an area for sea mines at a large distance from the support vessel. Due to the large distance, the communication between the AUVs and the support vessel is limited. Commonly, the division of the total search area over the AUVs is fixed at launch. If the sonar performance of one of the AUVs is less than expected, this will not be noticed until all AUVs have returned to the support vessel. This can lead to severe degradation of the total scanned area and strongly increase the chance of undetected mines. By modeling the MCM operation as a C-DCOP and dynamically assigning subareas to the AUVs using the C-DPOP algorithm, the AUVs cooperatively update their trajectories accordingly and increase the utility of the entire operation. Simulation results from a high-fidelity UUV simulator [14] are also given.

- The Distributed Bayesian (D-Bay) algorithm is presented. This algorithm leverages Bayesian optimization to solve C-DCOPs without the need for discretization completely. It operates directly on the continuous domains through a process referred to as sampling of the domains. Every sample is used to estimate the effects of the local utility functions on the global utility. The effects are modeled as Gaussian processes. Bayesian optimization is used to select the next sample in a sample-efficient manner based on all previously sampled values and a kernel. The kernel captures all *a priori* knowledge about the utility functions. When the Lipschitz constant of the local functions is known, the D-Bay algorithm is guaranteed to converge to the global optimum.
- Application of the D-Bay algorithm to a real-world problem. Analogous to the MCM operation, a search with Unmanned Aerial Vehicles (UAVs) is presented. UAVs are selected because they have similar properties to AUVs concerning their constraints and limitations. Both deviate from their course by wind/currents and scan an area with downward-facing sensors. Communication constraints are imposed through the use of low-bandwidth communication modules. Computation constraints are similar between UAVs and AUVs as both have embedded computers that limit the available computational power. Experiments with *real* UAVs are used to show that the D-Bay algorithm is suitable for real-world problems and can efficiently find high-quality solutions. Furthermore, the experiments are used to validate a simulation environment [11] in which additional (simulated) experiments are executed. In these experiments, various configurations of UAVs are tested.

1.4. THESIS OUTLINE

In this section, the outline of the thesis is given. This thesis is presented as a collection of papers, either published, accepted for publication, or under review. For this reason, the reader will encounter several repetitions, which allows all chapters to be read independently. The reader is advised to consider the definitions within the chapters as separate from the other chapters. The chapters describe various elements of the development of efficient solvers for C-DCOP problems. In Chapter 2, the Compression-DPOP (C-DPOP) algorithm is presented and applied to a sensor coordination problem. The algorithm is compared to DPOP and is found to outperform DPOP with uniform discretization regarding both computational resource requirement and performance in terms of utility. In Chapter 3, the C-DPOP algorithm is used within a high-fidelity UUV simulator for multiple AUVs for an MCM operation with online optimization of the survey trajectories based on sonar performance. Chapter 4 presents the Distributed Bayesian (D-Bay) algorithm. In this chapter, optimality guarantees are presented and comparisons with state-of-the-art DCOP and C-DCOP solvers are given. In Chapter 5, the D-Bay solver is applied on a real-world distributed search problem with UAVs. The experiments are additionally used to validate a simulation environment. The D-Bay algorithm can find high-quality solutions efficiently. Finally, Chapter 6 summarizes the results given in the thesis and presents topics for future work.

REFERENCES

[1] J. J. Acevedo, B. C. Arrue, I. Maza, and A. Ollero. "Cooperative large area surveillance with a team of aerial mobile robots for long endurance missions". In: *Journal of Intelligent and Robotic Systems: Theory and Applications* 70 (2013), pp. 329–345. DOI: 10.1007/s10846-012-9716-3.

[2] E. Andrulewicz. "Chemical weapons dumped in the Baltic Sea". In: *Assessment of the Fate and Effects of Toxic Agents on Water Resources*. Springer Netherlands, 2007, pp. 299–319. DOI: 10.1007/978-1-4020-5528-7_15.

[3] J. Bełdowski, J. Fabisiak, S. Popiel, Östin, Olsson, Vanninen, L. Anu, Lang, N. Fricke, Brenner, R. Berglind, Baršiene, Klusek, Pączek, M. Söderström, K. Lehtonen, M. Szubska, Malejevas, H. Koskela, and J. Fidler. *CHEMSEA findings: results from the CHEMSEA project - chemical munitions search and assessment*. Sopot: Institute of Oceanology of the Polish Academy of Sciences, 2014. ISBN: 978-83-936609-1-9. URL: https://www.researchgate.net/publication/324149647.

[4] J. Bełdowski, Z. Klusek, M. Szubska, R. Turja, A. I. Bulczak, D. Rak, M. Brenner, T. Lang, L. Kotwicki, K. Grzelak, J. Jakacki, N. Fricke, A. Östin, U. Olsson, J. Fabisiak, G. Garnaga, J. R. Nyholm, P. Majewski, K. Broeg, M. Söderström, P. Vanninen, S. Popiel, J. Nawała, K. Lehtonen, R. Berglind, and B. Schmidt. "Chemical Munitions Search & Assessment—An evaluation of the dumped munitions problem in the Baltic Sea". In: *Deep Sea Research Part II: Topical Studies in Oceanography* 128 (2016), pp. 85–95. DOI: 10.1016/j.dsr2.2015.01.017.

[5] R. Bellman. *Dynamic Programming*. Princeton University Press, 1957. ISBN: 9780691079516.

[6] E. Galceran and M. Carreras. "A survey on coverage path planning for robotics". In: *Robotics and Autonomous Systems* 61.12 (2013), pp. 1258–1276. DOI: 10.1016/j.robot.2013.09.004.

[7] A. Gershman, A. Meisels, and R. Zivan. "Asynchronous forward bounding for distributed COPs". In: *Journal of Artificial Intelligence Research (JAIR)* 34 (2009), pp. 61–88. DOI: 10.1613/jair.2591.

[8] S. Giodini, B. Binnerts, and K. Blom. "Can I communicate with my AUV?" In: *Hydro International* Unmanned Systems (2016), pp. 24–27. URL: https://resolver.tudelft.nl/uuid:c4b312ff-1528-4808-9147-e689646bca26.

[9] W. Jurczak and J. Fabisiak. "Corrosion of ammunition dumped in the Baltic Sea". In: *Journal of KONBiN* 41.1 (2017), pp. 227–246. DOI: 10.1515/jok-2017-0012.

[10] T. Knobloch, J. Bełdowski, C. Böttcher, M. Söderström, N.-P. Rühl, and S. Jens. *Chemical munitions dumped in the baltic sea*. Tech. rep. Report of the *ad hoc* expert group to update and review the existing information on dumped chemical munitions in the Baltic Sea (HELCOM MUNI), 2013. URL: https://helcom.fi.

[11] N. Koenig and A. Howard. "Design and use paradigms for Gazebo, an open-source multi-robot simulator". In: *International Conference on Intelligent Robots and Systems (IROS)*. 2004, pp. 2149–2154. DOI: 10.1109/IROS.2004.1389727.

[12] K. H. Low, J. M. Dolan, and P. Khosla. “Adaptive multi-robot wide-area exploration and mapping”. In: *International Conference on Autonomous Agents and Multiagent Systems (AAMAS)*. 2008, pp. 23–30. DOI: 10.1145/1402383.1402392.

[13] K. H. Low, J. M. Dolan, and P. K. Khosla. “Information-theoretic approach to efficient adaptive path planning for mobile robotic environmental sensing.” In: *International Conference on Automated Planing and Scheduling (ICAPS)*. 2009, pp. 233–240. URL: https://ojs.aaai.org/index.php/ICAPS/article/view/13344.

[14] M. M. M. Manhães, S. A. Scherer, M. Voss, L. R. Douat, and T. Rauschenbach. “UUV simulator: a Gazebo-based package for underwater intervention and multi-robot simulation”. In: *OCEANS*. 2016, pp. 1–8. DOI: 10.1109/OCEANS.2016.7761080.

[15] A. Meisels. *Distributed search by constrained agents: algorithms, performance, communication*. Springer Science & Business Media, 2007. DOI: 10.1007/978-3-642-24013-3_2.

[16] A. Meliou, A. Krause, C. Guestrin, and J. M. Hellerstein. “Nonmyopic informative path planning in spatio-temporal models”. In: *National Conference on Artificial Intelligence (AAAI)*. 2007, pp. 602–607. URL: https://www.aaai.org/Papers/AAAI/2007/AAAI07-095.pdf.

[17] P. J. Modi, W. M. Shen, M. Tambe, and M. Yokoo. “Adopt: Asynchronous distributed constraint optimization with quality guarantees”. In: *Artificial Intelligence* 161.1-2 (2005), pp. 149–180. DOI: 10.1016/j.artint.2004.09.003.

[18] R. NATO. *Environmental impact of munition and propellant disposal*. Tech. rep. RTO-TR-AVT-115, 2010. URL: https://www.sto.nato.int/publications/STO%20Technical%20Reports/RTO-TR-AVT-115/$$TR-AVT-115-ALL.pdf.

[19] A. Petcu and B. Faltings. “DPOP: a scalable method for multiagent constraint optimization”. In: *International Joint Conference on Artificial Intelligence (IJCAI)*. 2005, pp. 266–271. URL: https://www.ijcai.org/Proceedings/05/Papers/0445.pdf.

[20] D. Popa, A. Sanderson, R. Komerska, S. Mupparapu, D. Blidberg, and S. Chappel. “Adaptive sampling algorithms for multiple autonomous underwater vehicles”. In: *Autonomous Underwater Vehicles*. 2004, pp. 108–118. DOI: 10.1109/AUV.2004.1431201.

[21] D. M. Sato, A. P. Borges, P. Márton, and E. E. Scalabrin. “I-DCOP: train classification based on an iterative process using distributed constraint optimization”. In: *Procedia Computer Science* 51 (2015), pp. 2297–2306. DOI: 10.1016/j.procs.2015.05.391.

[22] E. A. Sultanik, P. J. Modi, and W. W. C. Regli. “On modeling multiagent task scheduling as a distributed constraint optimization problem”. In: *International Joint Conference on Artificial Intelligence (IJCAI)*. 2007, pp. 247–253. URL: https://www.ijcai.org/Proceedings/07/Papers/247.pdf.

[23] W. Yeoh and M. Yokoo. “Distributed problem solving”. In: *AI Magazine* 33 (2012), pp. 53–65. DOI: 10.1609/aimag.v33i3.2429.

[24] B. Zhang and G. S. Sukhatme. “Adaptive sampling with multiple mobile robots”. In: *International Conference on Robotics and Automation (ICRA)*. 2008. URL: https://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.146.3548.

[25] R. Zivan, T. Parash, and Y. Naveh. “Applying max-sum to asymmetric distributed constraint optimization”. In: *International Joint Conference on Artificial Intelligence (IJCAI)*. 2015, pp. 432–439. URL: https://dl.acm.org/doi/abs/10.5555/2832249.2832309.

2

DISTRIBUTED CONSTRAINT OPTIMIZATION FOR CONTINUOUS MOBILE SENSOR COORDINATION

DCOP (Distributed Constraint Optimization Problem) is a framework for representing distributed multi-agent problems. However, it only allows discrete values for the decision variables, which limits its application for real-world problems. In this chapter, an extension of DCOP is investigated to handle variables with continuous domains. Additionally, an iterative any-time algorithm Compression-DPOP (C-DPOP) is presented that is based on the Distributed Pseudo-tree Optimization Procedure (DPOP). C-DPOP iteratively discretizes the search space to handle problems that are restricted by time and memory limitations. The performance of the algorithm is examined through a mobile sensor coordination problem. The presented algorithm outperforms DPOP with uniform discretization regarding both resource requirements and performance.

This chapter has been published in the proceedings of the European Control Conference (ECC), 2018 [5].

2.1. Introduction

A wide range of real-world problems can be modeled as a Multi-Agents-Systems (MAS): scheduling problems [14], mobile sensor coordination [18], hierarchical task networks mapping [16], and control of modern robotics such as RoboCup Rescue [12]. Real-world problems often involve agents that are bound by inter-agent constraints (communication, movement) and by limited resources (memory, processing power). Likewise, bounded computation time is important to ensure safety when operating in an (uncertain) dynamic environment, for example, obstacles can only be avoided when the reaction is executed on time. The main challenge of MAS is to coordinate the actions of the agents by a distributed process, since a centralized process would become intractable for a large number of agents.

The distributed constraint optimization problem framework has been introduced to represent problems that are naturally distributed [11]. A DCOP is typically represented as a constraint graph, where nodes represent the variables of the problem, and edges represent a constraint or utility relation between the variables. The agents coordinate their actions by exchanging messages about the utility of their interactions. The utility values represent the differences in the cost and benefits of the actions for the individual agents. By using the utility, individual goals and internal dynamics are abstracted and hidden from other agents, which makes modeling of real-world problems less complex, since not all interactions need to be modeled in great detail. This results in modeling versatility and simplicity.

DCOP defines control variables with finite discrete domains, which limits its use for problems with continuous variables and utility functions. Within the DCOP framework, the latter type of problems are less studied [12]. For continuous path planning and multi-robot coordination/collision avoidance, specific solutions exist. In the work of Viseras *et al.* [17], a set of rapidly-exploring random trees is used as a discrete domain for the trajectories. An alternative approach is presented by Stranders *et al.* [15], where the utility functions are approximated as piecewise linear functions. These methods assume that the utility tables are readily available or can be computed beforehand. In real-world situations, this is often not the case, and the calculation of these values could be subject to large computational requirements. A generic solution transforms the continuous domains into discrete domains by uniform discretization for the desired resolution (distance between domain values). However, this would result in rapid growth in computational complexity. The complexity growth is due to the exponential growth of the *size* of the search space with respect to the interconnectivity and the size of the domains of the variables.

Numerous solvers for DCOP have been proposed; for a detailed overview of the taxonomy of DCOP algorithms, the reader is referred to the work of Cerquides *et al.* [3] and Leite *et al.* [7]. In this chapter, the focus is on the DPOP algorithm [11], since it requires a fixed number of communication steps. This property makes it suitable for real-world problems since it bounds the interactions between the agents. DPOP uses dynamic programming elements to communicate accumulated information among agents, such that every new message enables a more informed selection of the optimal control variables. To reduce the complexity growth, DPOP has been extended in numerous manners. These approaches can be divided into complete and approximate solvers. Complete solvers are guaranteed to find the (global) optimal solution, approximate solvers do not. Optimally solving DPOP is NP-HARD, which makes providing real-time guarantees unattainable [9]. Therefore, approximate solutions are considered to be a valid option for a problem with limited resources.

In local optimal solutions the computational requirements are reduced in three predominant methods:

1. *Reducing the number of interconnections* by iteratively adding interactions and re-evaluating the problem (I-DCOP [14]) until an acceptable solution is found. This reduces the maximum size of the message but could lead to infeasible solutions when the constraints of the problem are neglected.

2. *Restricting the number of variables in the messages* by dropping a set of variables from the message when the maximum message size is violated. This trades solution quality against computational complexity (A-DPOP [10]) based on lower and upper utility bounds. For memory-restricted agents, this could lead to arbitrarily poor performance.

3. *Limit the growth of the messages* by filtering inferior solutions based on global lower and upper utility bounds. This method is used to improve DPOP (MB-DPOP [2]), in combination with the Generalized Distributive Law [12] or max-sum algorithms [13]. This method can achieve a major reduction in communication however, this cannot be guaranteed. The benefits will depend greatly on the specific problem at hand.

In this chapter a fourth option is explored:

4. *Restricting the search space by considering subsets of the (continuous) domains* and iteratively applying DPOP while focusing the (continuous) domains around intermediate (local) solutions. Consequently, (continuous) domains can be discretized effectively based on local optima and limiting the need for computational resources.

The remainder of this chapter is organized as follows. First, Section 2.2 defines the DCOP model and its extension towards continuous domains. Then, Section 2.3 elaborates on the DPOP algorithm, which is extended in Section 2.4 by the proposed algorithm. Next, Section 2.5 defines the continuous mobile sensor coordination problem that is used to compare the performance and memory and computation time requirements of DPOP and the proposed algorithm. The implementation of the mobile sensor coordination problem is detailed and the results are analyzed. Finally, Section 2.6 summarizes the results and defines the future work.

2.2. DCOP DEFINITIONS & NOTATION

A distributed constraint optimization problem [12] is defined by a tuple $\langle A, X, D, F, G\rangle$, where:

A is a set of agents, i.e. $A = \{a_1, a_2, \ldots, a_{n_A}\}$ and $n_A \in \mathbb{N}$ is the number of agents.

X is a set of decision variables, i.e. $X = \{X_i \mid \forall a_i \in A\}$. $X_i = \{x_{i,1}, \ldots, x_{i,n_{X_i}}\}$ is the set of variables of agent a_i, and $n_{X_i} \in \mathbb{N}$ is its number of variables. The number of elements in set X_i is denoted as $|X_i|$.

D is a set of all domains of all variables, i.e. $D = \{D_p \mid \forall x_p \in X\}$ and the domain of variable x_p is defined as $D_p = \{d_{p,1}, d_{p,2}, \ldots, d_{p,n_{D_p}}\}$, where $n_{D_p} \in \mathbb{N}$ defines the number of elements within in the domain D_p.

F is a set of utility functions, i.e. $F = \{f_k : \mathbf{V}(f_k) \rightarrow \mathbb{R} \cup \{-\infty\}\}$. Utility functions can be used to model hard constraints by returning a value of $-\infty$ upon violation. Each function $f_k \in F$ is defined over a subset $\mathbf{V}(f_k) \in X$, also referred to as the scope of the function. The scope of F_i is similarly defined as $\mathbf{V}(F_i) = \{\mathbf{V}(f_k) | \forall f_k \in F_i\}$. Every agent a_i knows the utility functions that involve its own decision variables X_i. Formally, agent a_i knows the subset $F_i \subseteq F$, $F_i = \{f_k \in F \mid \mathbf{V}(f_k) \cap X_i \neq \emptyset\}$.

G is the global objective function that captures the aggregated utility of a complete allocation of all variables, denoted as $\mathfrak{X}$. An *allocation* $\mathfrak{X}$ maps each variable $x \in X$ to a value in its domain D. Formally, $\mathfrak{X} : X \rightarrow D$. The *projection* of an allocation $\mathfrak{X}$ over a set of variables $X_p \subseteq X$, written $\mathfrak{X}[X_p]$, is a new allocation $\mathfrak{X}_\mathfrak{p}$ formed by the values that $\mathfrak{X}$ assigns to the variables in X_p. The goal of a DCOP is to find an optimal allocation $\mathfrak{X}^* = \arg\max_{\mathfrak{X}} G(\mathfrak{X})$, where $G(\mathfrak{X}) = \sum_{f_k \in F} f_k(\mathfrak{X}[\mathbf{V}(f_k)])$.

For example, with $F = \{f_1\}$, $f_1(x_1) = {x_1}^2$, $\mathbf{V}(f_1) = \{x_1\}$, $D_1 = \{1, 2\}$, and $\mathfrak{X}_1 = \{x_1 = 2\}$, then $G(\mathfrak{X}_1) = f_1(\mathfrak{X}_1[\mathbf{V}(f_1)]) = f_1(\mathfrak{X}_1[x_1]) = f_1(2) = 4$.

The *search space* (all possible allocations of the variables) $\mathbf{X} = \prod_{x_p \in X} D_p$ defines each combination of all elements in the domain for the variables in set X, where $\prod$ is the set Cartesian product.

A DCOP is distributed in the sense that agents only interact through variables coupled by a utility function. The function set that is shared by agents a_i and a_j is denoted as $F_{i,j} = \{f_k \mid f_k \in F_i \cap F_j\}$. Likewise, the set of functions of agent a_i that are not shared by other agents is denoted as $F_{-i} = \{f_k \mid \mathbf{V}(f_k) \cap \mathbf{V}(f_j) = \emptyset,\ \forall f_j \in F \setminus F_i\}$.

2.2.1. THE C-DCOP MODEL

The Continuous DCOP (C-DCOP) model extends the DCOP model towards variables with continuous domains. Formally, it is a tuple $\langle A, X, D, F, G\rangle$, where:

A, F, G are equal to their definition in DCOP.

X is a set of decision variables $X = (X^d, X^c)$, where all variables with a discrete domain x^d belong to set $X^d = \{x_p \mid p = 1, 2, \ldots, n_{x_d}\}$ and all variables with a continuous domain x^c belong to set $X^c = \{x_q \mid q = 1, 2, \ldots, n_{x_c}\}$. The number of discrete variables and continuous variables is denoted as $n_{x^d} \in \mathbb{N}$ and $n_{x^c} \in \mathbb{N}$, respectively.

D is a domain set $D = (D^d, D^c)$, where $D^d = \{D^d_p \mid \forall x_p \in X^d\}$ and $D^c = \{D^c_q \mid \forall x_q \in X^c\}$. The continuous domain of variable x_q is defined by its lower and upper bound as $D^c_q = (\underline{d}^c_q, \overline{d}^c_q)$ indicating the domain over which the variable x_q can take a value.

2.3. DISTRIBUTED PSEUDOTREE OPTIMIZATION PROCEDURE (DPOP)

Distributed Pseudotree Optimization Procedure (DPOP) [11] is a solver for the DCOP framework. DPOP operates over a pseudo-tree [6], which is a rooted directed spanning tree, where connected nodes fall in the same branch. Every edge of the pseudo-tree is represented by a parent/child (direct) relation or by a back edge for a pseudo parent/pseudo child (indirect) relation. This representation allows for separating the main problem into sub-problems (between branches) and solving these independently before merging into the global assignment [8]. To create a pseudo-tree from a constraint graph, a depth-first search traversal can be executed. In this chapter, the Distributed Depth-First-Search (DFS) algorithm [1] is used, since it performs traversals along back edges in parallel, thereby reducing the time complexity. A DFS is defined by assigning the following properties to every node/agent a_i in the tree: the descendant/ancestor agents that are directly connected through a tree edge are indicated by $\mathbf{C}_i$, and $\mathbf{P}_i$, respectively. Descendant/Ancestor agents, that are indirectly connected through a back edge are indicated by $\mathbf{PC}_i$, and $\mathbf{PP}_i$, respectively. The set of all connected agents to agent a_i or to its descendants excluding the agent a_i itself is defined as $\mathrm{J}_i = \cup_{a_j \in \mathbf{C}_i} \mathrm{J}_j \cup \mathbf{P}_i \cup \mathbf{PP}_i \setminus \{a_i\}$.

The DPOP algorithm solves DCOP in three phases:

1. *Pseudo tree construction:* The agents distributively create the pseudo-tree structure, by a distributed pseudo-tree construction algorithm. Each agent a_i labels all its neighbors as either parent, pseudo parent, child, or pseudo child.

2. *Bottom-up utility propagation:* From the leave agents (agents without children) of the pseudo-tree, the agents pass a utility message U to their parents.

   A utility message U^i_j is sent by agent a_j to agent a_i based on the shared utility function set $F_{i,j} \subseteq F$, defined as $U^i_j \in \mathbb{R}^{n_1 \times n_2 \times \cdots \times n_{k,p}}$, where $n_{k,p} \in \mathbb{N}$ is the number of elements in D_p of variable $x_p \subset \mathbf{V}(F_{i,j})$. In words, it is a multidimensional matrix with one dimension for each variable within the scope of the shared utility functions. Parents receive utility matrices U_j^i from all their children indicating the combined utility for coupled variables between the agents.

All child matrices are combined as $U_{\mathbf{C}_i}^i = \bigoplus_{a_j \in \mathbf{C}_i} U_j^i$, where the messages are combined by a join operator ($\oplus$) as $U_{1,2}^i = U_1^i \oplus U_2^i$ such that the scope is the union of both scopes, i.e. $\mathbf{V}(U_{1,2}^i) = \mathbf{V}(U_1^i) \cup \mathbf{V}(U_2^i)$. The value of the elements of $U_{1,2}^i$ is the sum of the elements of U_1^i and U_2^i for all combinations of $\mathbf{V}(U_1^i)$ and $\mathbf{V}(U_2^i)$.

For example, $U_1^i = \begin{bmatrix} 1 & 3 \end{bmatrix}$, with $\mathbf{V}(U_1^i) = \{x_1\}$ and $U_2^i = \begin{bmatrix} 3 & 4 \end{bmatrix}$, with $\mathbf{V}(U_2^i) = \{x_2\}$. Combining these matrices results in $U_{1,2}^i = U_1^i \oplus U_2^i = \begin{bmatrix} 4 & 5 \\ 6 & 7 \end{bmatrix}$ with, $\mathbf{V}(U_{1,2}^i) = \{x_1, x_2\}$.

The utility matrices of the children are combined with the local utility matrix $U(F_{-i})$ and the utility matrices of the (pseudo) parents of the agent to form the total utility matrix U^i of agent a_i as $U^i = U_{\mathbf{C}_i}^i \oplus U(F_{-i}) \oplus \bigoplus_{a_j \in \{\mathbf{P}_i \cup \mathbf{PP}_i\}} U_i^j$.

The resulting matrix U^i is optimized over the local variables of the agent a_i. This operation is defined by a projection operator ($\perp$) and assigns the maximal utility of allocation of the local variable set X_i. The result is a matrix of lesser cardinality based on X_i, in words, for all values within the search space of a_i, the optimal value is chosen for the allocation $\mathfrak{X}_i$.

Formally, $U^i \perp X_i = \max_{\mathfrak{X}_i \in \mathbf{X}_i} U^i[\mathfrak{X}_i]$, and the scope $\mathbf{V}(U^i \perp X_i) = \mathbf{V}(U^i) \setminus X_i$. For example, when $U^i = \begin{bmatrix} 4 & 5 \\ 6 & 7 \end{bmatrix}$ with $\mathbf{V}(U^i) = \{x_1, x_2\}$, the resulting projection is $U^i \perp \{x_1\} = \begin{bmatrix} 6 & 7 \end{bmatrix}$ and $\mathfrak{X}_1 = \{\{x_1 = 2, x_2 = 1\}, \{x_1 = 2, x_2 = 2\}\}$.

After the projection, the matrix is sent toward its parent, $U_i^{\mathbf{P}_i} = U^i \perp X_i$. After the root agent of the pseudo-tree (agent without a parent) has finished this procedure, the value propagation phase is initiated.

3. *Top-down value propagation:* The root agent has accumulated the combined utility values U^i and can choose the optimal assignment of its local variable set X_i, $\mathfrak{X}_i^* = \arg\max_{\mathfrak{X}_i}(U^i \perp X_i)$. The allocation of these values is sent to all the children of the root agent. Based on these values the children allocate their own variables, $\mathfrak{X}_j^* = \arg\max_{\mathfrak{X}_j}(U^j \perp \mathfrak{X}_i^* \perp X_j)$.

   Every agent repeats this process until the leave agents are reached, completing the assignment $\mathfrak{X}^*$ for all variables within the DCOP.

A graphic representation of the DPOP algorithm can be seen in Figure 2.1, where a simple problem is shown during the execution of the algorithm.

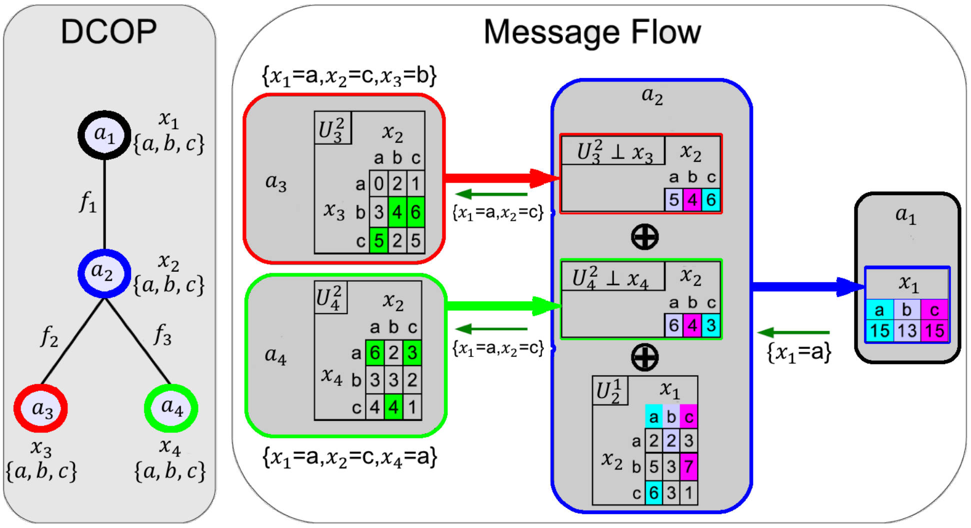


Figure 2.1.: A simple DCOP problem, adapted from [9].
Left: a pseudo-tree representation where the agents $A = \{a_1, a_2, a_3, a_4\}$ are represented as nodes. The edges represent the utility relations between the variables of the agents $F = \{f_1, f_2, f_3\}$. For example, the utility relation between a_1 and a_2, $f_1(a_1, a_2)$ yields the utility matrix $U_2^1 = \begin{bmatrix} 2 & 2 & 3 \\ 5 & 3 & 7 \\ 6 & 3 & 1 \end{bmatrix}$. The variables $X = \{X_1, X_2, X_3, X_4\}$, where $X_i = \{x_i\}$ for $i = 1,2,3,4$. All variables have identical domains $D = \{D_1, D_2, D_3, D_4\}$, where $D_1 = D_2 = D_3 = D_4 = \{a, b, c\}$.
Right: the message flow starts from the leaves of the tree (a_3, a_4) based on the projection of their utility matrices over their decision variables. The utility messages are depicted as colored arrows. These messages are combined by agent a_2 by the join operation ($\oplus$) before projecting out x_2 and sending the result to agent a_1. Afterwards, agent a_1 calculates the optimal assignment for x_1 and sends it to its child (a_2). The value messages are depicted as green arrows. Based on the value message of a_1, a_2 assigns the optimal value for x_2 before sending the combined assignment to its children (a_3 and a_4). Lastly, agents a_3 and a_4 assign their local values.

2.4. COMPRESSION-DPOP (C-DPOP)

To efficiently discretize the continuous domains of the variables within the C-DCOP framework, the Compression-DPOP (C-DPOP) algorithm is proposed. C-DPOP is an approximate any-time iterative algorithm that extends DPOP by dynamic discretization of the continuous domains by a fixed number of values per variable, $n_{s_q} \in \mathbb{N}$ for all $x_q \in X^c$. After every iteration, the width of the domain of variable x_q, w_q is decreased according to a compression factor ($0 < c_q < 1$), which results in an increased domain resolution r_q (value distance). This process can be stopped at any time if the allowed computation time has expired.

C-DPOP consists of five phases which are iteratively executed by all agents:

1. *Domain discretization:* All continuous domains of agent a_i are discretized based on its upper and lower bound $(\overline{d}_q, \underline{d}_q)$ so to generate discrete domains. Based on the number of values per domain n_{s_q}, the domain is uniformly discretized $D^c_q \rightarrow D^d_q$ such that, $D^d_q = \{d_{q,1}, \dots, d_{q,n_{s_q}}\}$ and $d_{q,r} = \underline{d}_q + (r-1)\frac{w_q}{n_{s_q}-1}$.

2. *Utility tables creation:* Based on the search space of the discrete domains the utility functions are used to calculate the utility values. Every agent calculates the local utilities $U(F_{-i})$, and the utilities of the (pseudo) parents $U^j_i \, \forall j \in \mathbf{P}_i \cup \mathbf{PP}_i$.

3. *Bottom-up utility propagation:* After the utility tables are generated, the leaves of the tree start by sending their utility matrices to their parents. These matrices are combined with the local utility matrices, which are then projected over the local variables before being sent to the parents. This phase is identical to DPOP.

4. *Top-down value propagation:* The root of the tree initiates the value propagation, after which all children allocate their local variables before sending it to their children. This phase is identical to DPOP.

5. *Domain compression:* After a (local) optimal allocation for the local variables $\mathfrak{X}^*_i$ is found, the upper and lower bounds of the domains are updated. This is done by compressing the width of the domain $w^+_q = w_q c_q$, where $w_q := \overline{d}_q - \underline{d}_q$ and centering the domain around the allocated values $\mathfrak{X}^*_i[x_q]$. The new domain is defined as $D^c_q = (\underline{d}^c_q, \overline{d}^c_q)$, where $\underline{d}^c_q = \mathfrak{X}^*_i[x_q] - \frac{w^+_q}{2}$, and $\overline{d}^c_q = \mathfrak{X}^*_i[x_q] + \frac{w^+_q}{2}$.

The compression of the domain is a key part of the algorithm since it iteratively refines the domains. The proposed restriction strategy is based on a contracting grid search. Through this strategy, the discretization of the continuous domains is focused around the (local) optimal value (exploitation of the found solution), while the compression of the domains reduces the exploration after every iteration. An overview of the strategy for a one-dimensional domain can be seen in Figure 2.2.

If the resources to solve the C-DCOP are time and/or memory constraints, the execution of C-DPOP can stop after a fixed number of iterations. The permitted number of iterations can be calculated explicitly based on the available memory and computation time. The memory requirement is a function of the size of the domain per variable. The

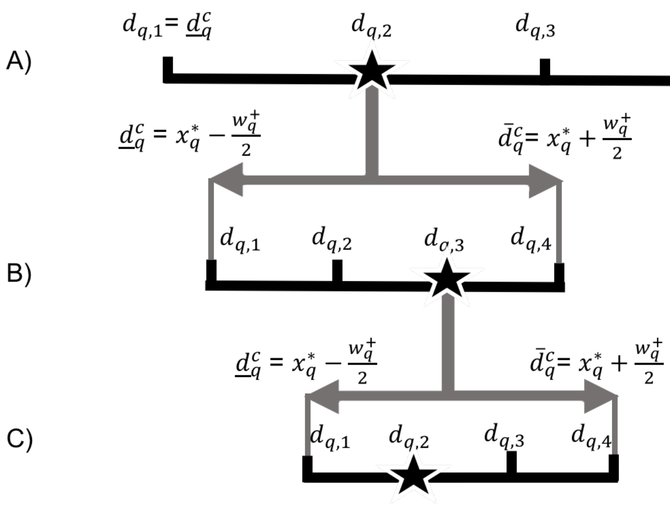

A)
B)
C)
$d_{q,1} = \underline{d}^c_q$
$d_{q,2}$
$d_{q,3}$
$d_{q,4} = \bar{d}^c_q$
$\underline{d}^c_q = x^*_q - \frac{w^+_q}{2}$
$\bar{d}^c_q = x^*_q + \frac{w^+_q}{2}$
$d_{q,1}$
$d_{q,2}$
$d_{q,3}$
$d_{q,4}$

Figure 2.2.: An overview of the domain compression strategy. The domain is represented as a horizontal black line on which the values $\{d_{q,r}\}_{r=1}^4$ are shown. Domain D^C_q is updated over three iterations (A, B, C) by reducing the width of the domain $w^+_q = w_q c_q$ and centering around the optimal value (represented as vertical arrows). The optimal value at every iteration is indicated by a star.

total required memory by agent a_i can be calculated as $m_i = |\mathbf{X}_i| m_1$, where $|\mathbf{X}_i|$ is the size of the search space for all variables in X_i, and m_1 is the required number of bytes to store the utility of a single assignment (typically 8 bytes for a float). The required time can be calculated based on the number and execution time of all function evaluations. The number of evaluations is equal to the size of the search space of all variables in the scope of the function, $n_{e_{f_k}} = |\mathbf{X}_{f_k}|$. It is a property of the number of variables in the scope and the size of their domains, since $\mathbf{X}_{f_k} = \prod_{x_q \in \mathbf{V}(f_k)} D_q$. The evaluation time of a function f_k is a property of that function and is denoted as t_{f_k}. This property is assumed to be known for all functions $f_k \in F$. Based on these two properties the required computation time for agent a_i can be defined as $t_i = \sum t_{f_k} n_{e_{f_k}}$ for all $f_k \in F_{-i} \cup F_{i,j} \forall j \in \mathbf{P}_i \cup \mathbf{PP}_i$. The achievable resolution r^i_q of agent a_i for variable x_q can be calculated based on the chosen compression factor c_q, the initial width of the domain w_q, the number of values within the domain n_{s_q}, and the number of iterations $n_{I_q} \in \mathbb{N}$, as $r^a_q = c_q^{n_{I_q}-1} \frac{w_q}{n_{s_q}-1}$.

2.5. SENSOR COORDINATION PROBLEM

2.5.1. PROBLEM STATEMENT

A mobile sensor coordination problem is used to evaluate the performance and computational requirements of the C-DPOP algorithm. The problem is adapted from Zivan *et al.* [19] where a team of mobile sensors with limited sensing range needs to coordinate their positions (from initial locations) to minimize the number of unobserved targets in an area. The problem is extended for agents with limited memory and computation time, and the agents can select positions over continuous domains within a two-dimensional plane.

The problem is described within the C-DCOP framework where the position of agent a_i is defined within the variable set $X_i = \{p_i^{\mathrm{ax}}, p_i^{\mathrm{ay}}\}$. The available resources are defined as $t_i^{\max}$ for time and $m_i^{\max}$ for memory of agent a_i. The utility functions of the problem are defined as $F = \{F_T, F_A\}$. $F_T = \{f_{t,l}\}_{l=1}^{n_T}$ is the utility function set for sensing the targets, where $n_T \in \mathbb{N}$ is the number of targets. A target T_l is defined as a point $(p_l^{\mathrm{tx}}, p_l^{\mathrm{ty}})$. The utility function of target l is described as,

$$f_{t,l} = \begin{cases} 1 & \text{if } d(T_l, X_j) \le s_j \\ 0 & \text{if } d(T_l, X_j) > s_j \end{cases},$$

where s_j is the sensing distance of the closest agent (a_j), and $d(T_l, X_j)$ is the Euclidean distance between the location of the target and the position of the agent. The utility function set for the movement is represented as $F_A = \{f_{a,i}\}_{i=1}^{n_A}$, where

$$f_{a,i} = -d(X_i, I_i)$$

and $I_i = (p_i^{\mathrm{ax0}}, p_i^{\mathrm{ay0}})$ is the initial location of agent a_i.

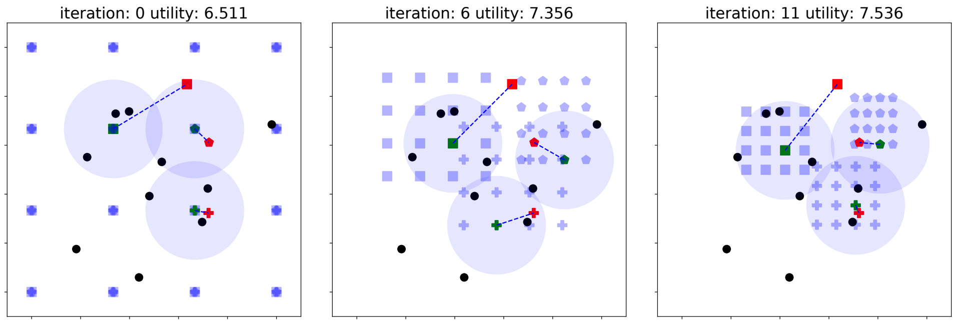
The resulting goal function is defined as

$$\begin{aligned} G(\mathfrak{X}) &= \sum_{f_k \in F} f_k(\mathfrak{X}[\mathbf{V}(f_k)]) \\ &= \sum_{f_{t,l} \in F_T} f_{t,l}(\mathfrak{X}[\mathbf{V}(f_{t,l})]) + \sum_{f_{a,i} \in F_A} f_{a.i}(\mathfrak{X}[\mathbf{V}(f_{a.i})]). \end{aligned}$$

In other words, the goal function defines the difference in utility gain from targets in the sensor range and the cost of moving to a new location for all agents.

2.5.2. Implementation

The performance of the proposed C-DPOP algorithm is compared to DPOP with a uniform discretization of the continuous domains. The DPOP method discretizes the continuous domains only once, while C-DPOP iteratively discretizes the domains as described in Section 2.4. An example of the operation of C-DPOP can be seen in Figure 2.3. The two methods are compared for randomly generated mobile sensor coordination problems. In every problem, the initial location of the agents I and the location of the targets T are chosen at random. The agents can choose any position within a bounded normalized area and therefore the domains of the variables are all equal, $D_{x,i}^c = D_{y,i}^c = (0,1)$ for all $a_i \in A$. Since all variables have equal domains and all agents have similar variables, the agent subscript and variable subscripts will be neglected for brevity. All agents are assumed to have an equal amount of available memory and computation



(a) Initial values. **(b)** Intermediate values. **(c)** Final values.

Figure 2.3.: Selecting and compressing domains during C-DPOP for a three-agent, ten-target example. The initial positions of the agents are shown as red markers, their search space (domains) as blue markers, and their optimal value as green markers. The agents are identified by markers (square, plus, pentagon). The targets are indicated as black circles. Additionally, the transparent blue circles around the intermediate solutions indicate the sensing distance and the dotted lines indicate the required movement from the initial position to the intermediate solution. Snapshots from three iterations are shown. Initially, in Figure 2.3a, the domains can be seen to overlap, seemingly only showing the search space of a single agent. After six iterations, in Figure 2.3b, the domains of the agents are compressed and centered around their found (local) optima. At this time the domains partially overlap but can be seen to concentrate on distinct areas. Finally, in Figure 2.3c, the domains do not overlap and the distance between the intermediate solutions is converging.

time. Based on these resources, the maximum achievable number of values within the domain will be limited by either the memory or the computation time. For DPOP, the maximum achievable size of the domains within the available resources is used.

The C-DPOP algorithm requires a fixed domain size and a compression factor as parameters to calculate the required number of iterations. The domain sizes and the compression factor can be selected based on the properties of the underlying problem. In the case of mobile sensor coordination, the domain size can be selected to correspond to a minimum resolution. In this case, it is defined with relation to the sensor range ($s_i = 0.2$) to achieve overlap within the initially sensed areas. Based on the area size, the domain size for C-DPOP is therefore set to 5. The compression factor should be chosen based on the properties of the underlying problem. A compression factor close to 0 will compress the domain relatively fast, excluding large segments of the domain at every iteration. Thereby, converging to a (local) optimum rapidly, which is not preferable for all problems since this can exclude segments that hold the global optimum. A compression factor close to 1 will be able to escape a local optimum more easily when a new local optimum is found, however, it will require more iterations to converge. Correspondingly, a factor close to 0 has less chance to escape a local optimum but will converge within fewer iterations. For the problem at hand, this trade-off was made based on the sensor range of the agents. Since the compression factor is relative to the (current) domain width, at the first compression, the size of the domains is reduced by the greatest absolute value. By choosing a compression factor c of 0.9 at most half of the sensor range of the agents is not sampled. A comparison of achieved utility for a three-agent, ten-target problem is shown in Figure 2.4.

During the evaluation of the performance of DPOP and C-DPOP, it was found that the utility was highly dependent on the achievable resolution (distance between domain values) of the algorithms. A high resolution will result in accurate positioning, which induces the least amount of movement cost and achieves higher sensing utility by being able to sense multiple targets simultaneously. This property can be seen in Figure 2.5 where the computation time and memory requirements to achieve a certain resolution are compared. Showing that given the same amount of resources, C-DPOP can achieve a higher resolution.

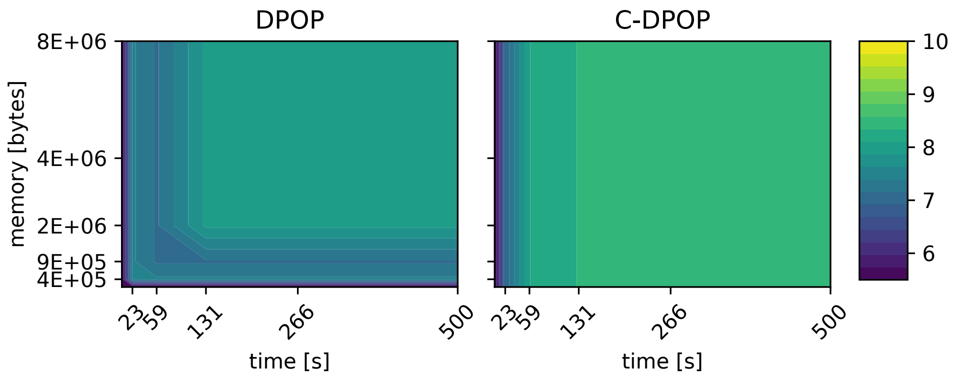


Figure 2.4.: A comparison of achieved utility for a three-agent, ten-target problem based on available resources. **Left:** the utility of the DPOP algorithm, where the increase in resources (and domain size) can be clearly seen to increase the performance. **Right:** the utility of the C-DPOP algorithm, where the effect of the constant domain size can be seen in the memory invariance. The performance can be seen to gradually increase when more iterations are possible within the available time.

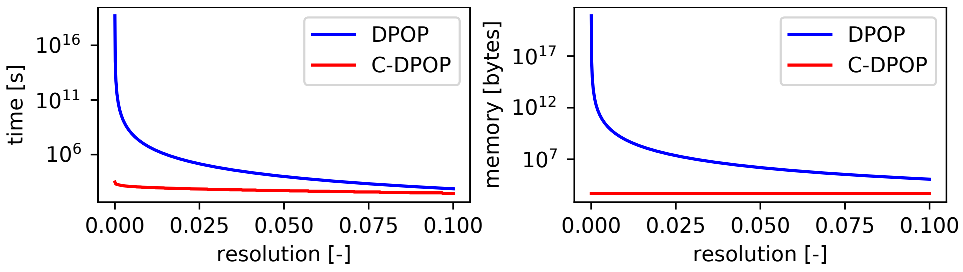


Figure 2.5.: Required time and memory comparison between DPOP approach and the C-DPOP algorithm for a three agent, ten target problem. It can be seen that the time and memory requirements for DPOP grow exponentially when the resolution is improved.

2.6. CONCLUSION

In this chapter, an extension of the Distributed Constraint Optimization Problem (DCOP) called Continuous DCOP (C-DCOP) is presented to represent variables with continuous domains. The modeling simplicity and versatility of DCOP are hereby extended to include problems with continuous variables. Many real-world problems contain inter-agent constraints and limited resources, such as limited computation time and memory. Especially in dynamic environments and close collaboration, these bounds need to be taken into account. For this reason, a solver for the C-DCOP model has been presented that takes these constraints into account explicitly. The presented Compression-DPOP (C-DPOP) algorithm is an extension of Distributed Pseudotree Optimization Procedure (DPOP) that iteratively discretizes the continuous domains and dynamically updates the domains after each iteration based on the found (local) optimum.

A mobile sensor coordination problem was used to compare the performance of C-DPOP and DPOP. Here it was found that the C-DPOP algorithm outperforms DPOP for resource-constrained agents. The higher utility was achieved by using the available resources more effectively.

In future work, the discretization method of the continuous domains will be extended from uniform to utility-based, where the utility values of the previous iteration will be used to predict regions of high utility. The prediction is then refined at every iteration by incorporating the gained information from the solution. Furthermore, we will compare the performance between C-DPOP and other approximate DCOP solvers such as DSA [4]. Additionally, the convergence properties of the C-DPOP algorithm will be investigated with regard to the compression factor.

REFERENCES

[1] B. Awerbuch. "A new distributed depth-first-search algorithm". In: *Information Processing Letters* 20.3 (1985), pp. 147–150. DOI: 10.1016/0020-0190(85)90083-3.

[2] I. Brito and P. Meseguer. "Improving DPOP with function filtering". In: *International Conference on Autonomous Agents and Multiagent Systems (AAMAS)*. 2010, pp. 141–148. URL: https://dl.acm.org/doi/10.5555/1838206.1838226.

[3] J. Cerquides, A. Farinelli, P. Meseguer, and S. D. Ramchurn. "A tutorial on optimization for multi-agent systems". In: *The Computer Journal* 57.6 (2014), pp. 799–824. DOI: 10.1093/comjnl/bxt146.

[4] S. Fitzpatrick and L. Meertens. "Distributed coordination through anarchic optimization". In: *Distributed Sensor Networks*. Springer, 2003, pp. 257–295. DOI: 10.1007/978-1-4615-0363-7_11.

[5] J. Fransman, J. Sijs, H. S. Dol, E. Theunissen, and B. De Schutter. "Distributed constraint optimization for continuous mobile sensor coordination". In: *European Control Conference (ECC)*. Limassol, Cyprus, 2018. DOI: 10.23919/ECC.2018.8550486.

[6] E. C. Freuder and M. J. Quinn. "Taking advantage of stable sets of variables in constraint satisfaction problems". In: *International Joint Conference on Artificial Intelligence (IJCAI)*. 1985, pp. 1076–1078. URL: https://www.ijcai.org/Proceedings/85-2/Papers/082.pdf.

[7] A. R. Leite, F. Enembreck, and J.-P. A. Barthès. "Distributed constraint optimization problems: review and perspectives". In: *Expert Systems with Applications* 41.11 (2014), pp. 5139–5157. DOI: 10.1016/j.eswa.2014.02.039.

[8] A. Meisels. *Distributed search by constrained agents: algorithms, performance, communication*. Springer Science & Business Media, 2007. DOI: 10.1007/978-3-642-24013-3_2.

[9] A. Petcu. "A class of algorithms for distributed constraint optimization". PhD thesis. École Polytechnique Fédérale De Lausanne, 2007. DOI: 10.5075/epfl-thesis-3942.

[10] A. Petcu and B. Faltings. "Approximations in distributed optimization". In: *International Conference on Principles and Practice of Constraint Programming (CP)*. 2005, pp. 802–806. DOI: 10.1007/11564751_68.

[11] A. Petcu and B. Faltings. "DPOP: a scalable method for multiagent constraint optimization". In: *International Joint Conference on Artificial Intelligence (IJCAI)*. 2005, pp. 266–271. URL: https://www.ijcai.org/Proceedings/05/Papers/0445.pdf.

[12] M. Pujol-Gonzalez. "Scaling DCOP algorithms for cooperative multi-agent coordination". In: *Constraints* 20.4 (2015), pp. 496–497. DOI: 10.1007/s10601-015-9222-x.

[13] A. Rogers, A. Farinelli, R. Stranders, and N. R. Jennings. "Bounded approximate decentralised coordination via the max-sum algorithm". In: *Artificial Intelligence* 175.2 (2011), pp. 730–759. DOI: 10.1016/j.artint.2010.11.001.

[14] D. M. Sato, A. P. Borges, P. Márton, and E. E. Scalabrin. "I-DCOP: train classification based on an iterative process using distributed constraint optimization". In: *Procedia Computer Science* 51 (2015), pp. 2297–2306. DOI: 10.1016/j.procs.2015.05.391.

[15] R. Stranders, A. Farinelli, A. Rogers, and N. R. Jennings. "Decentralised coordination of continuously valued control parameters using the max-sum algorithm". In: *International Conference on Autonomous Agents and Multiagent Systems (AAMAS)*. 2009, pp. 601–608. URL: https://dl.acm.org/doi/10.5555/1558013.1558097.

[16] E. A. Sultanik, P. J. Modi, and W. W. C. Regli. "On modeling multiagent task scheduling as a distributed constraint optimization problem". In: *International Joint Conference on Artificial Intelligence (IJCAI)*. 2007, pp. 247–253. URL: https://www.ijcai.org/Proceedings/07/Papers/247.pdf.

[17] A. Viseras, V. Karolj, and L. Merino. "An asynchronous distributed constraint optimization approach to multi-robot path planning with complex constraints". In: *Symposium on Applied Computing*. 2017, pp. 268–275. DOI: 10.1145/3019612.3019708.

[18] R. Zivan, T. Parash, and Y. Naveh. "Applying max-sum to asymmetric distributed constraint optimization". In: *International Joint Conference on Artificial Intelligence (IJCAI)*. 2015, pp. 432–439. URL: https://dl.acm.org/doi/abs/10.5555/2832249.2832309.

[19] R. Zivan, H. Yedidsion, S. Okamoto, R. Glinton, and K. Sycara. "Distributed constraint optimization for teams of mobile sensing agents". In: *Autonomous Agents and Multi-Agent Systems* 29.3 (2015), pp. 495–536. DOI: 10.1007/s10458-014-9255-3.

3
DISTRIBUTED CONSTRAINT OPTIMIZATION FOR AUTONOMOUS MULTI AUV MINE COUNTER-MEASURES

In this chapter, Mine Counter-Measures (MCM) operations with multiple cooperative Autonomous Underwater Vehicles (AUVs) are examined within the Distributed Constraint Optimization Problem (DCOP) framework. The goal of an MCM operation is to search for mines and mine-like objects within a predetermined area so that ships can pass through the area within a safe transit corridor. Performance metrics, such as the expected time of completion and the level of confidence that all mine-like objects within the area have been detected, are used to quantify the utility of the operation. The AUVs coordinate their search segments in a distributed manner to maximize global utility. The segmentation is optimized by the Compression-DPOP (C-DPOP) algorithm, which allows explicit reasoning by the AUVs about their actions based on the performance metrics. After initial segmentation of the mine threat area, subsequent optimizations are triggered by the AUVs based on the variations in sonar performance. The performance of the C-DPOP algorithm is compared to a static segmentation approach and validated using the high-fidelity Unmanned Underwater Vehicle (UUV) simulation environment based on the Gazebo simulator.

This chapter has been published in the proceedings of the OCEANS conference, 2018 [8].

3.1. Introduction

The operational objective of a naval mine mission is a reduction of the risk that ships hit a mine while transiting through a particular body of water. Such risk reduction is achieved by conducting a Mine Counter-Measures (MCM) operation. An overview of a multi Autonomous Underwater Vehicle (AUV) MCM operation can be seen in Figure 3.1.

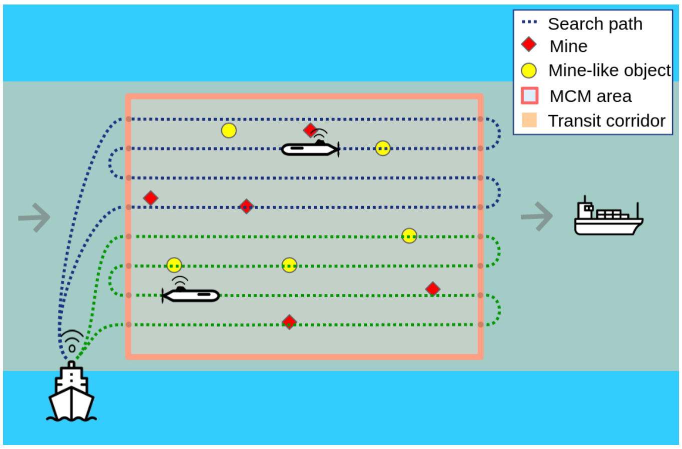


Figure 3.1.: An overview of a multi Autonomous Underwater Vehicle (AUV) Mine Counter-Measures (MCM) operation, adapted from [5].
The goal is to clear the transit corridor for shipping traffic by detecting all mines within the operation area. The AUVs are launched from a support vessel and during the search, they traverse lawnmower search paths.

As defined in [6], the MCM operation consists of several sub-tasks:

1. Detection: detect mine-like objects as mine-like echos (MILECs) by scanning the MCM area with a larger sonar range yet a lower resolution;
2. Classification: revisiting MILECs and scanning with a higher resolution yet lower sonar range to classify them as a mine-like contact (MILCO) or a false positive (NON MILCO);
3. Identification: revisiting MILCOs to acquire optical images that are assessed by a human operator to identify the MILCO as a mine or a false positive;
4. Disposal: revisiting all mines to perform disposal.

In this chapter, the focus is on the segmentation of the MCM area within the detection task for multiple cooperating AUVs. In the near future, the detection task will be performed by multiple AUVs in an autonomous manner [2]. The segmentation will be performed based on nominal sonar performance indicators such as the effective sonar range [24]. After the segments are scanned by the AUVs, the results are assessed at the support vessel [4]. In case of unsatisfactory results due to variations in the achieved sonar performance, the operation could be extended to cover the regions that are not adequately scanned [23]. This iterative process increases the total operation time considerably. To reduce the total required time, the search segments could be adapted during the operation through interaction between the AUVs.

There exists a wide range of related work involving an adaptive search for both AUVs and Unmanned Surface Vessels (USVs). Most works focus on the coverage path planning problem in which a path needs to be optimally planned for a sensor to cover a certain area [9]. In the approach of [26], the search area is discretized based on *equal information gain* of the discrete segments. USVs are assigned to explore segments yielding the least amount of cost for the individual USVs. A similar approach is taken in [20] with multiple AUVs for optimal resource assignment based on motion costs, uncertainty reduction, and optimization over secondary objectives such as communication bandwidth and energy consumption. These approaches require the discretization of the search area. A common problem of discretization is the exponential memory requirement when the total survey area increases [13]. This problem is alleviated in [14] and [16] through approximation and a dynamic programming [3] approach based on information gain. These centralized optimization approaches, where the support vessel optimizes the segmentation of all AUVs *in-situ*, are considered infeasible for the MCM operation due to communication constraints on both range and bandwidth [10]. Communication between AUVs is considered viable since during the operation their distance is not as extensive. Furthermore, the AUVs could form a communication network in which messages can be passed between AUVs that have no direct connection. However, a centralized approach, where an AUV receives all information and optimizes the actions of all AUVs, is considered impractical due to the low processing power of the AUVs.

For these reasons, in this chapter, the coverage path planning is modeled and solved in a distributed manner by the Compression-DPOP (C-DPOP) algorithm [7]. The distributed approach allows for *in-situ* adaption of the search segments through inter-AUV communication without (*a priori*) discretization of the search area. The C-DPOP algorithm is applied to the MCM problem modeled within the Distributed Constraint Optimization Problem (DCOP) framework.

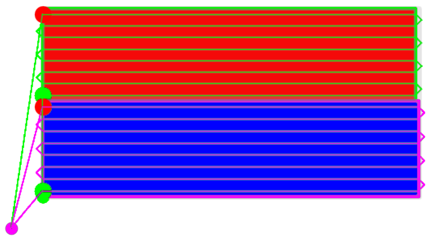
The remainder of this chapter is outlined as follows. Section 3.2 defines the MCM detection task problem as a coverage path planning problem. Afterward, the DCOP framework and the C-DPOP algorithm are detailed and the detection task is modeled as a DCOP in Section 3.3. In Section 3.4, the performance of the C-DPOP algorithm is compared to a static approach within a high-fidelity simulation environment. Finally, the results are discussed in Section 3.5.

3.2. Problem statement

The detection task is based on two local performance metrics of the AUVs:

1. Expected time of Completion (EoC): expected time for completing the search segment of the agent;
2. Probability of Detection (PoD): the level of confidence that all mine-like objects within the segment have been detected as a mine-like echo (MILEC).

The task is completed when the PoD of the search area is higher than the required PoD (P^{req}) within a maximum operation time t^{max}, which are both set by an operator. The MCM search area is modeled as a monotone rectilinear polygon, $s = (x, y, w, h)$, where x and y indicate the center of the area and w, h denotes the width and height, respectively. The scan segment of AUV i is defined similarly to the MCM area as $s_i = (x_i, y_i, w_i, h_i)$. The combined segments of all AUVs is denoted as $S = \bigcup_{i=1,\dots,n} s_i$, where n is the number of AUVs. An example of segmentation for two and three AUVs can be seen in Figure 3.2. The total operation time defined as $T = \max\limits_{i=1,\dots,n} T_i$, where T_i is the EoC of AUV i.


(a) Segmentation example for two AUVs.

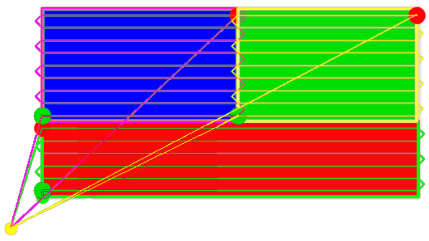

(b) Segmentation example for three AUVs.

Figure 3.2.: Example of segmentation of an MCM search area. The covered segments of the AUVs are presented as colored areas. The lawnmower patterns of the AUVs are shown as contrasting colored lines starting from and finishing in a colored circle. These circles indicate the initial and rendezvous position, respectively. The green and red circles mark the start and finish of the lawnmower pattern.

The AUVs are equipped with a pair of Side-Scan Sonars (SSS) (mounted on the port and starboard side) in order to scan according to a *lawnmower* pattern consisting of several *legs*. Legs are defined as straight lines within the pattern. The AUVs are actively scanning the seabed while traversing the legs.

The lawnmower pattern is optimal for rectangular search areas where the turn radius of the AUV (d_i^{turn}) is smaller than the distance between the legs, which is determined by the sonar range (r_i) [1]. For AUVs the sonar range is typically several factors higher than the turn radius, therefore only lawnmower patterns are considered in this chapter.

The sonar range is defined as the distance over which the sonar achieves a particular PoD. It is based on the properties of the SSS, the environment, and the height over the seabed during scanning. Sailing close to the seabed results in a short sonar range and a high PoD since the Signal-to-Noise Ratio (SNR) of the sonar will be high. An increase in height will increase the sonar range, but consequently decreases the SNR and thereby the PoD. Based on this trade-off the height over the seabed is fixed during scanning based on the required PoD for the MCM search area. Additional AUV properties which are taken into account are the velocity during travel (v_i^{transit}), and the velocity during scanning (v_i^{scan}).

The EoC for agent i (T_i) is a function of the scan segment s_i and the transit time based on the initial position of the agent,

$$T_i = t_i^{\text{initial}} + t_i^{\text{scan}} + t_i^{\text{return}}$$

where t_i^{initial} is the initial transit time towards the scan segment, t_i^{scan} is the time spent scanning, and t_i^{return} is the rendezvous time from the scan segment back to the initial position.

The transit times depend on the Euclidean distance $d(\cdot)$ and the transit velocity of the AUV as,

$$\begin{aligned} t_i^{\text{initial}} &= d(p_i^{\text{initial}}, p_i^{\text{start}}) \,/\, v_i^{\text{transit}} \\ t_i^{\text{return}} &= d(p_i^{\text{initial}}, p_i^{\text{finish}}) \,/\, v_i^{\text{transit}} \end{aligned}$$

where $p_i^{\text{initial}} = (x_i^{\text{initial}}, y_i^{\text{initial}})$, $p_i^{\text{start}} = (x_i^{\text{start}}, y_i^{\text{start}})$, and $p_i^{\text{finish}} = (x_i^{\text{finish}}, y_i^{\text{finish}})$ indicate the position of the initial location, start of the first leg, and the finish of the final leg, respectively. Note that, since the depth is considered a constant, it is neglected from the notations.

The time spent scanning (t_i^{scan}) depends on the total length of the legs and the number of turns within the lawnmower pattern. In order to minimize the number of turns, the longest side of the scan segment ($d_i^{\text{long}} = \max(w_i, h_i)$) is taken as the scan direction of the AUV. As a result, the shortest side ($d_i^{\text{short}} = \min(w_i, h_i)$) is used to determine the number of required legs l_i within the lawnmower pattern according to the sonar range of the agent. The size of the turns is determined by the turn radius of the AUV (d_i^{turn}) and the distance between the legs (d_i^{leg}). Consequently, the time spent scanning is defined as

$$\begin{aligned} t_i^{\text{scan}} &= \frac{l_i d_i^{\text{long}}}{v_i^{\text{scan}}} + \frac{\left(2d_i^{\text{turn}} + d_i^{\text{leg}}\right)(l_i - 1)}{v_i^{\text{scan}}}, \\ l_i &= \left\lceil \frac{d_i^{\text{short}}}{2r_i} \right\rceil, \\ d_i^{\text{leg}} &= \frac{d_i^{\text{short}}}{l_i}, \end{aligned}$$

where $\lceil \cdot \rceil$ denotes the ceiling function. Note that the number of legs l_i depends on the combined range of the port and starboard side SSS systems of the AUV.

The global goal function G is defined as the utility relation between areas covered with sufficient PoD and the EoC, formalized as a rectangular Gaussian distribution,

$$G = \exp\left(-\left(\frac{(R^{\text{coverage}} - 1)^2}{(\sigma^{\text{coverage}})^2} \right)^{f^{\text{coverage}}} - \left(\frac{(R^{\text{time}})^2}{(\sigma^{\text{time}})^2} \right)^{f^{\text{time}}} \right),$$

$$R^{\text{coverage}} = \frac{\lambda(S)}{\lambda(s)}, \qquad R^{\text{time}} = \frac{T}{t^{\max}},$$

where $\lambda(\cdot)$ denotes the Lebesgue measure [12] indicating the area of a segment, R^{coverage} is the ratio of the MCM area that is covered to the required PoD, R^{time} is the time ratio of the required time over the maximum allowed time $t^{\max}$. An operator can tune the relative significance between coverage and required time through the scale factors σ^{coverage}, σ^{time}, f^{coverage}, and f^{time}. Using these factors, the relative importance can be indicated between the coverage and time ratio, as well as the slopes of the utility function with regard to coverage and required time. A graphical overview of the global utility is depicted in Figure 3.3.

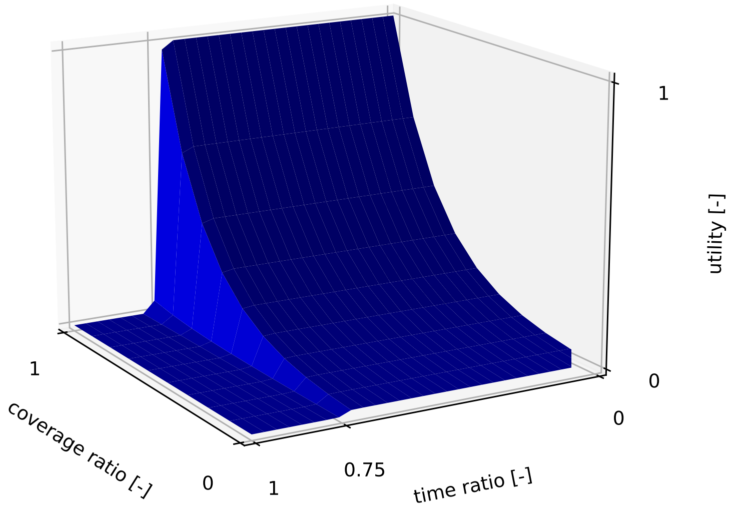


Figure 3.3.: An overview of the global utility function that is based on the ratio of the area of the covered segment R^{coverage} and the required time ratio R^{time} for $\sigma^{\text{coverage}} = 0.25$, $\sigma^{\text{time}} = 0.75$, $f^{\text{coverage}} = 0.5$, and $f^{\text{time}} = 50$. Note the steep drop at the time ratio equal to 0.75, indicating that (almost) no utility is gained when the required time exceeds the maximum available time.

The global goal function, as described above, is iteratively optimized throughout the detection task. After the initial optimization, the AUVs start to traverse their lawnmower patterns and, after every leg, their achieved sonar range is assessed. Due to seabed conditions and environmental conditions such as current, the achieved sonar range could deviate from their nominal sonar range. If an AUV detects a variation from the nominal sonar range r_i, it sends a trigger to all AUVs to reinitialize the optimization. If there is no variation, it will continue to traverse the remaining legs without informing the other AUVs. This decreases the required level of communication, which is beneficial for communication-constrained underwater search operations.

When the optimization is triggered, the remaining MCM area is updated by subtracting the segments that have been sufficiently covered. After the solution is found, the AUVs resume their search based on the updated solution. This process is repeated every time an AUV triggers the optimization.

3.3. Distributed Constraint Optimization Problem

In this chapter, the MCM detection operation is modeled as a Distributed Constraint Optimization Problem (DCOP), which is a generalization of a Distributed Constraint Satisfaction Problem (DCSP) [25].

A DCOP is defined by a tuple $\langle A, X, D, F, G \rangle$ [21] where;

A	is a set of agents,
X	is a set of decision variables,
D	is a set of domains for all variables,
F	is a set of utility functions,
G	is the global objective function.

A DCOP is distributed in the sense that agents only interact with agents that are coupled through their variables by a utility function. This allows for the modeling of the problem as a constraint or function graph, thereby deconstructing the problem into an ordered pseudo-tree, making it possible to optimize various subproblems in parallel, such as the calculation of the EoC for all the AUVs within the MCM detection task. The nodes of the graph represent the local actions of the AUVs while the edges represent a constraint or utility relation between the actions. An example of a conversion from a constraint graph into a pseudo-tree can be seen in Figure 3.4.

This deconstruction makes the DCOP framework especially suitable for modeling multi-AUV operations because the global performance of the AUVs can be described by the interactions of their local actions. For example, the total scanned area is the union of all the segments scanned by the agents.

The goal of a DCOP is to optimize the global utility function by assigning values for all variables in a distributed manner. The complete assignment of all variables is denoted as $\mathfrak{X} = \bigcup_{i=1}^{n} X_i$. The variables can be assigned a value from within a bounded domain, denoted as the action space $\mathfrak{D} = \bigcup_{i=1}^{n} D_i$. Within the detection task, this indicates the assignment of segments to the AUVs within the MCM search area. AUVs coordinate their actions by exchanging messages about the utility of the interactions between their variables. The cost (in time) and benefit (in coverage) of the segments are expressed in terms of utility towards the global goal.

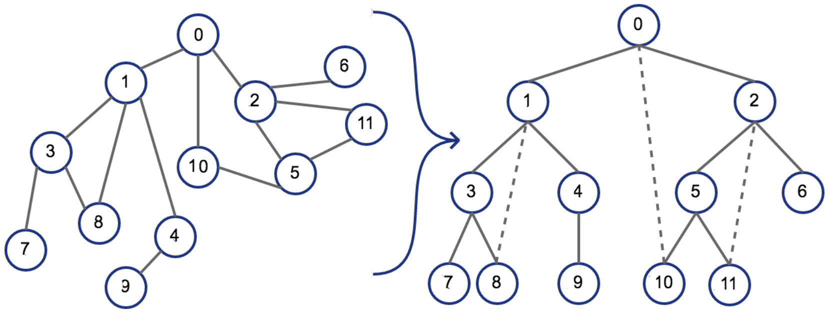


Figure 3.4.: An example of the conversion of a DCOP problem as a constraint graph into a pseudo-tree. The nodes represent variables or actions. The edges represent a constraint or utility relation between the nodes. Indirect or pseudo-connections are illustrated as dotted edges.

These variables are optimized *in situ* by the Compression-DPOP (C-DPOP) algorithm [7]. C-DPOP is based on the Distributed Pseudo-Tree Optimization Procedure (DPOP) [19], which is a DCOP solver that uses dynamic programming elements to communicate accumulated information about the global utility. DPOP requires a fixed number of communication steps during optimization, which is beneficial for MCM since underwater communication is subject to severe constraints.

A drawback is that the message size increases exponentially for large domains. This drawback is especially unfavorable for variables with continuous domains (D^{cont}) that are discretized to a high resolution, for example, the subdivision of the search area into a grid of predefined squares. C-DPOP eliminates this drawback by iteratively creating discrete domains (D^{disc}) by discretizing the continuous domains. At every iteration, the discrete domains and the (local) optimum are used to compress the continuous domains around this optimum. The compression decreases the size of the continuous domains, which results in discrete domains of increasing resolution after every iteration. The algorithm terminates when the resolution of all discretized domains is smaller than a predefined threshold. An overview of the C-DPOP algorithm is given in Figure 3.5.

The MCM detection task is represented within the DCOP framework as

$$\begin{aligned}
&A = (a_1, \ldots, a_n)\,,\\
&X = (X_1, \ldots, X_n)\,,\text{ where } X_i = \{s_i\},\\
&D = \left(D_1^{\text{cont}}, \ldots, D_n^{\text{cont}}\right),\text{ where } D_i^{\text{cont}} = \{s\}\,,\\
&F = \{T_1, \ldots, T_n, \lambda(s_1), \ldots, \lambda(s_n)\},\\
&G = \exp\left(-\left(\frac{(R^{\text{coverage}}-1)^2}{(\sigma^{\text{coverage}})^2}\right)^{f^{\text{coverage}}} - \left(\frac{(R^{\text{time}})^2}{(\sigma^{\text{time}})^2}\right)^{f^{\text{time}}}\right).
\end{aligned}$$

Note that the domains of the search segments are bounded by the MCM search area $s = (x, y, w, h)$.

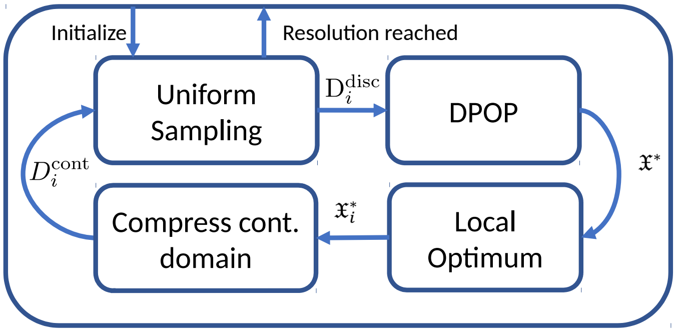


Figure 3.5.: An overview of the C-DCOP algorithm, where the discrete domains of AUV i, $D_i^{\mathrm{disc}}(t)$ at time t are iteratively created from the continuous domains $D_i^{\mathrm{cont}}(t)$. After the optimization, the (locally) optimally assigned values of AUV i ($\mathfrak{X}_i^*$) are used to compress the continuous domains. The algorithm terminates when the resolution of all discrete domains is smaller than a predefined threshold.

3.4. SIMULATION ENVIRONMENT

The performance of the C-DPOP algorithm for the MCM operations is validated through the high-fidelity Unmanned Underwater Vehicle (UUV) simulation environment [15]. The simulator is based on the Gazebo simulator [11] and Robotic Operating System (ROS) [22] and includes hydrodynamics, (underwater) current, underwater sensor functions, and various AUV types. In this chapter, the AUVs are modeled after the A9-S of the ECA Group[1] as it is designed for seabed imagery operations. Figure 3.6 shows a rendering of the A9-S AUV as modeled[2] within the UUV simulator [15].

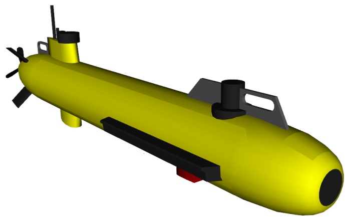


Figure 3.6.: A rendering of the A9-S AUV of the ECA Group.

Within the simulation environment, all AUVs are considered to be identical and therefore share all parameters such as maximum scan and transit velocities, sonar range, and turn radius. The parameters are chosen following the operational specification of the ECA A9-S AUV. The required PoD is set such that the sonar range is equal to 150 m, as defined within the specifications of the ECA A9-S. The maximum time is set corresponding to the time a single AUV would require to cover the entire MCM search area.

[1] https://www.ecagroup.com/en/solutions/a9-s-auv-autonomous-underwater-vehicle
[2] https://github.com/uuvsimulator/eca_a9

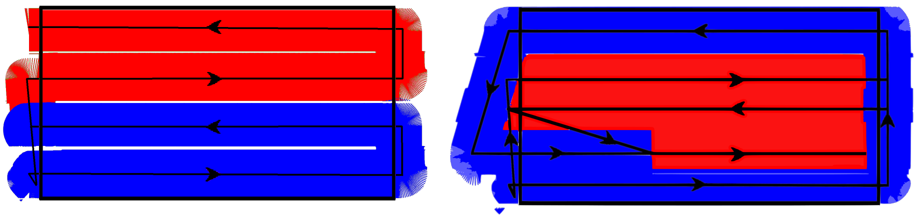
Two approaches are compared to assess the performance of the dynamic application of the C-DPOP algorithm. The first is a static approach in which the C-DPOP algorithm is executed once (at the start of the operation). The resulting segmentation is not updated during the operation. The second is a dynamic approach in which the segmentation is updated based on the triggers of the AUVs.

3.4.1. SIMULATION RESULTS

The performance of C-DPOP is assessed and compared to a static segmentation for multiple AUVs based on the nominal sonar performance. The performance is measured according to the global utility function based on the achieved covered area and required operation time. Two scenarios are evaluated in which the sonar range is less than nominal. The first scenario involves two AUVs, of which the sonar performance decreases gradually, and the MCM area covers 2 km × 1 km. The second scenario involves three AUVs, of which one has severely decreased sonar performance, and the MCM area covers 4 km × 4 km. In both scenarios, the AUVs start at the rendezvous location to mimic the deployment by a support vessel.

In the first scenario, the reduction in sonar performance is not compensated by the static approach, which results in a loss of coverage over the entire width of the MCM area. By incorporating the information from the AUVs about their performance decrease, the search segments are optimized by the dynamic approach based on the remaining available time to maximize the global utility.

Figure 3.7 shows the covered segments and trajectories of both approaches. The results in terms of the global utility, time and coverage ratios are shown in Table 3.1.



(a) Results of static approach. **(b)** Results of the dynamic approach.

Figure 3.7.: Results for the first scenario of the static and dynamic C-DPOP approaches. The MCM area is indicated by a white rectangle, the covered segments are color-coded based on the individual AUVs. Within the segments, the lawnmower pattern is shown as lines with contrasting colors. As can be seen in Figure 3.7a, the static approach leaves several sections uncovered, while in Figure 3.7b most of the area is covered by the updated search pattern. Note that, during turns the scanned areas are incomplete, indicating the decreased probability of detection.

Table 3.1.: Results of the first scenario.

Approach	time ratio	coverage ratio	global utility
Static	0.58	0.95	0.93
Dynamic	0.83	0.98	0.98

Figure 3.8 shows the results for the static and dynamic approach in terms of the coverage segments and the trajectories of the AUVs for the second scenario. The results are summarized in Table 3.2. The higher utility is achieved by the dynamic approach as the performance loss of one AUV is compensated by the others. In the static approach, the lack of additional optimization leaves large segments not covered, which increases the risk of undetected mines within that area.

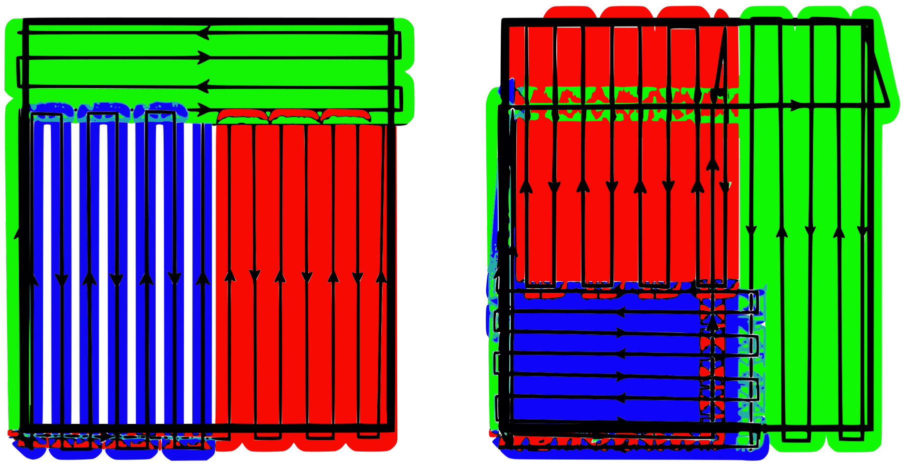

(a) Results of static approach. **(b)** Results of the dynamic approach.

Figure 3.8.: Example of the results for the second scenario of the static and dynamic C-DPOP algorithm. The MCM area is indicated by a white rectangle, the covered segments are color-coded based on the individual AUVs. Within the segments, the lawnmower pattern is shown as lines with contrasting colors. As can be seen in Figure 3.8a, the sonar range of the AUV in the bottom left corner (shown in blue) is severely decreased. In Figure 3.8b, the result of the dynamic optimization can be seen as the AUV decreases the distance between the legs and the other AUVs adjust their trajectories. Note that, the dynamic approach covers several locations multiple times due to the change in orientation of the scan segments.

Table 3.2.: Results of the second scenario.

Approach	time ratio	coverage ratio	global utility
Static	0.88	0.87	0.83
Dynamic	0.95	0.99	0.99

3.4.2. Discussion

In both scenarios, the dynamic C-DPOP approach achieves a higher global utility. The utility increase depends on the size of the sonar performance loss during operation and the maximum available time of the operation. When the performance loss is moderate as in the first scenario, only slight gains can be achieved. However, in the second scenario, considerable performance loss can be overcome when the maximum operational time allows for additional legs within the MCM area. When the available remaining time is limited, this improvement is reduced, since no time is available for additional legs.

The results of the two scenarios can be extended towards larger MCM areas without additional computational resources since no *a priori* discretization of the MCM area is required. Furthermore, due to the assumption that the longest edge of the MCM area is used as scan direction, these results are analogous to MCM areas with other height-to-width ratios. For example, the segmentation for an MCM area of $2\,\text{km} \times 4\,\text{km}$ is similar to the segmentation for an area of $4\,\text{km} \times 2\,\text{km}$. This assumption does not hold in practice for every height-to-width ratio, due to several aspects. Two of the most important aspects are the environmental properties and the positional uncertainty of the AUVs. Important environmental properties are underwater currents and ocean topography since both can severely deteriorate the sonar performance. Positional uncertainty is defined as the error between the actual and estimated position. The source of this uncertainty is due to the attenuation of Global Positioning System (GPS) signals underwater. AUVs are required to estimate their position during scanning instead of interpolating from the GPS signals. The estimation is typically performed through the use of inertial sensors, however, the position error for this estimation method is unbounded [18]. Therefore, the maximum leg length is often restricted based on the growth of the position error. In order to cope with this increasing error, the AUVs interrupt the scanning to acquire a GPS fix by surfacing when the error crosses a (predefined) threshold.

3.5. Conclusions and Future Work

This chapter presents a method to segment a search area of multiple cooperative Autonomous Underwater Vehicles (AUVs) in a distributed manner based on global performance metrics set by an operator. The application of the distributed constraint optimization problem framework is extended towards autonomous operations of multiple AUVs without the need for the discretization of the Mine Counter-Measures (MCM) areas, which allows for flexible modeling and optimization. The optimization of scan segments of the AUVs is initially performed based on nominal sonar performance and repeated when an AUV triggers a re-optimization during the operation. The performance of the Compression-DPOP (C-DPOP) algorithm is compared against static segmentation, which is the de facto standard for current multi-AUV operations. Results show higher achieved utility for the dynamic approach based on the C-DPOP algorithm.

Future work includes adding the position uncertainty of the AUVs during the operation as this results in significant operational limitations in practice [18]. Based on the position uncertainty the scanning trajectories can be adjusted such that the risk of collision between AUVs is minimized. Additionally, to reduce the position uncertainty the implementation of cooperative simultaneous localization and mapping (SLAM) [17] methods will be investigated.

REFERENCES

[1] V. Ablavsky and M. Snorrason. "Optimal search for a moving target: a geometric approach". In: *AIAA Guidance, Navigation and Control Conference*. 2000. DOI: 10.2514/6.2000-4060.

[2] A. Belbachir, F. Ingrand, and S. Lacroix. "A cooperative architecture for target localization using multiple AUVs". In: *Intelligent Service Robotics* 5.2 (2012), pp. 119–132. DOI: 10.1007/s11370-012-0107-1.

[3] R. Bellman. *Dynamic Programming*. Princeton University Press, 1957. ISBN: 9780691079516.

[4] E. Bovio, D. Cecchi, and F. Baralli. "Autonomous underwater vehicles for scientific and naval operations". In: *Annual Reviews in Control* 30.2 (2006), pp. 117–130. DOI: 10.1016/j.arcontrol.2006.08.003.

[5] M. Cramer. *Understanding information uncertainty within the context of a net-centric data model: a mine warfare example*. Tech. rep. Mine warfare Environmental Decision Aids Library (MEDAL), 2008, p. 49. URL: https://www.researchgate.net/publication/235143929.

[6] F. Florin, F. Van Zeebroeck, I. Quidu, and N. Le Bouffant. "Classification performances of mine hunting sonar: theory, practical results and operational applications". In: *the Undersea Defence Technology conference (UDT)*. 2003. URL: https://www.researchgate.net/publication/45341625.

[7] J. Fransman, J. Sijs, H. S. Dol, E. Theunissen, and B. De Schutter. "Distributed constraint optimization for continuous mobile sensor coordination". In: *European Control Conference (ECC)*. Limassol, Cyprus, 2018. DOI: 10.23919/ECC.2018.8550486.

[8] J. Fransman, J. Sijs, H. Dol, E. Theunissen, and B. De Schutter. "Distributed constraint optimization for autonomous multi AUV mine counter-measures". In: *OCEANS MTS/IEEE*. Charleston, SC, USA, 2018. DOI: 10.1109/OCEANS.2018.8604924.

[9] E. Galceran and M. Carreras. "A survey on coverage path planning for robotics". In: *Robotics and Autonomous Systems* 61.12 (2013), pp. 1258–1276. DOI: 10.1016/j.robot.2013.09.004.

[10] S. Giodini, B. Binnerts, and K. Blom. "Can I communicate with my AUV?" In: *Hydro International* Unmanned Systems (2016), pp. 24–27. URL: https://resolver.tudelft.nl/uuid:c4b312ff-1528-4808-9147-e689646bca26.

[11] N. Koenig and A. Howard. "Design and use paradigms for Gazebo, an open-source multi-robot simulator". In: *International Conference on Intelligent Robots and Systems (IROS)*. 2004, pp. 2149–2154. DOI: 10.1109/IROS.2004.1389727.

[12] H. Lebesgue. "Intégrale, Longueur, Aire". PhD thesis. Universitè de Paris, 1902. DOI: 10.1007/BF02420592.

[13] K. H. Low, J. M. Dolan, and P. Khosla. “Adaptive multi-robot wide-area exploration and mapping”. In: *International Conference on Autonomous Agents and Multiagent Systems (AAMAS)*. 2008, pp. 23–30. DOI: 10.1145/1402383.1402392.

[14] K. H. Low, J. M. Dolan, and P. K. Khosla. “Information-theoretic approach to efficient adaptive path planning for mobile robotic environmental sensing.” In: *International Conference on Automated Planing and Scheduling (ICAPS)*. 2009, pp. 233–240. URL: https://ojs.aaai.org/index.php/ICAPS/article/view/13344.

[15] M. M. M. Manhães, S. A. Scherer, M. Voss, L. R. Douat, and T. Rauschenbach. “UUV simulator: a Gazebo-based package for underwater intervention and multi-robot simulation”. In: *OCEANS*. 2016, pp. 1–8. DOI: 10.1109/OCEANS.2016.7761080.

[16] A. Meliou, A. Krause, C. Guestrin, and J. M. Hellerstein. “Nonmyopic informative path planning in spatio-temporal models”. In: *National Conference on Artificial Intelligence (AAAI)*. 2007, pp. 602–607. URL: https://www.aaai.org/Papers/AAAI/2007/AAAI07-095.pdf.

[17] L. Paull, G. Huang, M. Seto, and J. J. Leonard. “Communication-constrained multi-AUV cooperative SLAM”. In: *International Conference on Robotics and Automation (ICRA)*. 2015, pp. 509–516. DOI: 10.1109/ICRA.2015.7139227.

[18] L. Paull, S. Saeedi, M. Seto, and H. Li. “AUV navigation and localization: a review”. In: *Journal of Oceanic Engineering* 39.1 (2014), pp. 131–149. DOI: 10.1109/JOE.2013.2278891.

[19] A. Petcu and B. Faltings. “DPOP: a scalable method for multiagent constraint optimization”. In: *International Joint Conference on Artificial Intelligence (IJCAI)*. 2005, pp. 266–271. URL: https://www.ijcai.org/Proceedings/05/Papers/0445.pdf.

[20] D. Popa, A. Sanderson, R. Komerska, S. Mupparapu, D. Blidberg, and S. Chappel. “Adaptive sampling algorithms for multiple autonomous underwater vehicles”. In: *Autonomous Underwater Vehicles*. 2004, pp. 108–118. DOI: 10.1109/AUV.2004.1431201.

[21] M. Pujol-Gonzalez. “Scaling DCOP algorithms for cooperative multi-agent coordination”. PhD thesis. Universitat Autònoma de Barcelona, 2014. URL: https://dialnet.unirioja.es/servlet/tesis?codigo=120236.

[22] M. Quigley, K. Conley, B. P. Gerkey, J. Faust, T. Foote, J. Leibs, R. Wheeler, and A. Y. Ng. “ROS: an open-source Robot Operating System”. In: *ICRA workshop on open source software*. 2009. URL: https://www.researchgate.net/publication/303138182.

[23] D. P. Williams, F. Baralli, M. Micheli, and S. Vasoli. “Adaptive underwater sonar surveys in the presence of strong currents”. In: *International Conference on Robotics and Automation (ICRA)*. Vol. 6. 2016, pp. 2604–2611. DOI: 10.1109/ICRA.2016.7487418.

[24] R. B. Wynn, V. A. I. Huvenne, T. P. Le Bas, B. J. Murton, D. P. Connelly, B. J. Bett, H. A. Ruhl, K. J. Morris, J. Peakall, D. R. Parsons, E. J. Sumner, S. E. Darby, R. M. Dorrell, and J. E. Hunt. "Autonomous Underwater Vehicles (AUVs): their past, present and future contributions to the advancement of marine geoscience". In: *Marine Geology* 352 (2014), pp. 451–468. DOI: 10.1016/j.margeo.2014.03.012.

[25] M. Yokoo, E. H. Durfee, T. Ishida, and K. Kuwabara. "The distributed constraint satisfaction problem: formalization and algorithms". In: *Transactions on Knowledge and Data Engineering* 10.5 (1998), pp. 673–685. DOI: 10.1109/69.729707.

[26] B. Zhang and G. S. Sukhatme. "Adaptive sampling with multiple mobile robots". In: *International Conference on Robotics and Automation (ICRA)*. 2008. URL: https://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.146.3548.

4

Distributed Bayesian: a continuous Distributed Constraint Optimization Problem solver

In this chapter, the novel Distributed Bayesian (D-Bay) algorithm is presented for solving multi-agent problems within the Continuous Distributed Constraint Optimization Problem (C-DCOP) framework. This framework extends the classical DCOP framework towards utility functions with continuous domains. D-Bay solves a C-DCOP by utilizing Bayesian optimization for the adaptive sampling of variables. We theoretically show that D-Bay converges to the global optimum of the C-DCOP for Lipschitz continuous utility functions. The performance of the algorithm is evaluated empirically based on the sample efficiency. The proposed algorithm is compared to state-of-the-art DCOP and C-DCOP solvers. The algorithm generates better solutions while requiring fewer samples.

This chapter has been submitted to JAIR, 2021 [15].

4.1. Introduction

Many real-world problems can be modeled as multi-agent problems in which agents need to assign values to their variables to optimize a global objective characterized by a utility function. Examples include scheduling [48], mobile sensor coordination [65], hierarchical task network mapping [51], and cooperative search [1]. Even though numerous algorithms exist that solve these problems, applying them in practice is often problematic, as complications arise from limitations in communication, computation, and/or memory.

The Distributed Constraint Optimization Problem (DCOP) framework is well suited to model the above-mentioned problems (as detailed in Gershman *et al.* [19], Meisels [33], Modi *et al.* [40], Petcu *et al.* [43], and Yeoh *et al.* [63]). Within the DCOP framework, a problem is defined based on variables and on utility functions that are aggregated into an objective function. Additionally, agents assign values to all the variables that are allocated to them. Agents are considered neighbors if their variables are arguments of the same utility function. Neighbors cooperatively optimize their utility functions through the exchange of messages. Within the DCOP framework, variables are constrained by their domains. In other words, a domain defines all possible value assignments of a variable. This explicit definition of the domains of the variables is suitable for real-world problems that are (input) constrained. These domains are considered to be finite and discrete within a DCOP, while real-world problems are typically characterized by finite continuous domains. Problems with finite continuous domains can be modeled as a Continuous DCOP (C-DCOP), which is equal to a DCOP except for the domain definition. A common approach to solve a C-DCOP with a DCOP solver is to use equidistant discretization of the domains, such as using a grid overlay to define all possible positions of an agent in an area. This process converts the continuous domains into discrete domains. When discretizing a continuous domain, the quality of the solution will depend on the distance between the values, where a smaller distance will allow for a better solution. This results in a trade-off as more values will increase the cardinality of the discrete domains. The increase in cardinality will result in polynomial growth of the search space where the degree of the polynomial is equal to the number of agents. From the overview articles of Leite *et al.* [27] and Fioretto *et al.* [14], it is clear that the cardinality of the domains is a major restriction to DCOP solvers. Therefore, solving a C-DCOP by discretization can become intractable for DCOP solvers despite a small number of variables.

The underlying reason for the increase in problem size is that DCOP solvers (implicitly) consider all values within a domain as unrelated to each other. Because of this assumption, it is not possible to efficiently sample the search space. In problems with continuous domains, this assumption does not hold since the utility of values that are close is often similar. By explicitly taking such a relation into account, a C-DCOP solver based on efficient sampling methods can be constructed.

Several C-DCOP solvers (CMS [50], C-CoCoA [47], PFD [11], AC-DPOP [22]) have been introduced that, based on initial discretization of the continuous domains, update the discretized values within an iterative optimization method such as local gradient descent. In this chapter, Bayesian optimization [37] will be used as it focuses on efficient sampling during optimization, thereby requiring relatively few iterations to closely approach the optimum. This method eliminates the need for the discretization of the domains.

Overall, the contributions of this chapter are threefold. Firstly, we introduce an efficient algorithm that uses methods found in Bayesian optimization to solve C-DCOPs called Distributed Bayesian (D-Bay). Secondly, we provide theoretical proof of the convergence of the proposed algorithm to the global optimum of the C-DCOP for utility functions with known Lipschitz constants. Lastly, simulation results are given for randomly generated graphs and sensor coordination problems to compare the sample efficiency of D-Bay to state-of-the-art DCOP and C-DCOP solvers.

The remainder of this chapter is organized as follows. Firstly, in Section 4.2 background information about the DCOP framework is given. In Section 4.3 relevant literature regarding DCOP solvers is discussed. The Bayesian optimization algorithm is provided in Section 4.4. Afterward, we present the novel sampling-based C-DCOP solver called D-Bay in Section 4.5. The theoretical properties of D-Bay are analyzed in Section 4.6. Evaluation of D-Bay for sensor coordination problems and random graphs are included in Section 4.7. Finally, Section 4.8 summarizes the results and defines future work.

4.2. Distributed Constraint Optimization Problems

The DCOP framework originates from an extension and generalization of Constraint Satisfaction Problems (CSPs) [53] towards distributed optimization. A solution for a CSP is defined as the assignment of all variables from (finite) discrete domains such that all hard constraints are satisfied. The CSP framework has been extended from a centralized problem framework to an agent-based distributed problem framework in the work of Yokoo *et al.* [64]. Within the Distributed-CSP framework, the variables are allocated to agents and the agents coordinate the value assignments among each other.

Additionally, CSP has been generalized into the Constraint Optimization Problem (COP) framework, where the hard constraints are replaced with soft constraints expressed as utility functions. Utility functions return a cost or reward based on the value assignments. Hard constraints are enforced by the utility functions by returning infinite cost (or infinite negative reward). Instead of constraint satisfaction, the goal of a COP is to find assignments that optimize an objective function.

A final extension is the Distributed Constraint Optimization Problem (DCOP) framework, which provides a unified framework that includes a large class of problems by combining the generalization of Distributed-CSP and the extension of COP. A graphical overview of the relations between the problem frameworks can be seen in Figure 4.1.

In the DCOP framework, the domains are considered to be discrete, which limits its application to problems with continuous domains. In the current chapter, the Continuous

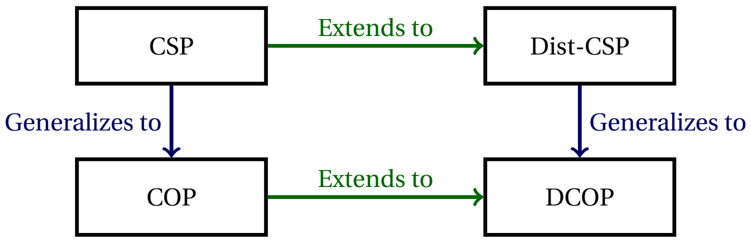


Figure 4.1.: Graphical overview of the relations between the problem frameworks. Adapted from Fioretto *et al.* [14].

DCOP (C-DCOP) framework will be used to overcome this restriction to include real-world problems such as cooperative search.

A DCOP is a problem in which an objective function needs to be optimized in a distributed manner through value assignments for all variables. The objective function consists of the aggregate of utility functions, which define a utility value for the value assignments of the variables. All variables within the DCOP are exclusively allocated to agents. An agent is responsible to assign values to all the variables that are allocated to it. Typically, the number of variables is equal to the number of agents, i.e. every agent is allocated a single variable. The agents cooperate by sending messages to agents with whom they share a utility function. A utility function is shared between agents if their variables are in the arguments of that function. An important aspect of a DCOP is the definition of the domains of the variables. A domain defines all possible values that a variable can be assigned to. In other words, the value assignments are restricted by the domains of the variables.

Following the notation of Fioretto *et al.* [14], a DCOP is defined by $\mathfrak{D} = \langle \mathbf{A}, \mathbf{X}, \mathbf{D}, \mathbf{F}, \alpha, \eta \rangle$ where,

- $\mathbf{A} = \{a_1, \ldots, a_M\}$ is the set of agents, where M is the number of agents.
- $\mathbf{X} = \{x_1, \ldots, x_N\}$ is the set of discrete variables, where $N \geq M$ is the number of variables.
- $\mathbf{D} = \{\mathbf{D}_1, \ldots, \mathbf{D}_N\}$ is the set of domains of all variables, where $\mathbf{D}_i \subseteq \mathbb{R}$ is the finite discrete domain associated with variable x_i. The search space of the DCOP is defined by all possible combinations of all values within the domains as $\Sigma = \prod_{i=1}^{N} \mathbf{D}_i$, where $\prod$ is the set Cartesian product. The search space of a set of variables ($\mathbf{V} \subseteq \mathbf{X}$) is defined as $\Sigma_{\mathbf{V}} = \prod\limits_{i : x_i \in \mathbf{V}} \mathbf{D}_i$.

  An assignment denotes the projection of variables onto their domain as $\rho : \mathbf{X} \to \Sigma$. In other words, for all $x_i \in \mathbf{X}$ if $\rho(x_i)$ is defined, then $\rho(x_i) \in \mathbf{D}_i$. An assignment of a subset of variables is denoted by $\rho_{\mathbf{V}} = \{\rho(x_i) : x_i \in \mathbf{V}\}$.
- $\mathbf{F} = \{f_1, \ldots, f_K\}$ is the set of utility functions, where K is the number of utility functions. The scope of f_n is denoted as $\mathbf{V}_n \subseteq \mathbf{X}$, where $x_i \in \mathbf{V}_n$ when x_i is an argu-

ment of f_n. The optimum of f_n is denoted by $y_n^* = \max_{x \in \Sigma_{\mathbf{V}_n}} f_n(x)$ with input $x_n^* = \arg\max_{x \in \Sigma_{\mathbf{V}_n}} f_n(x)$, where $\Sigma_{\mathbf{V}_n}$ denotes the domain of the utility function.

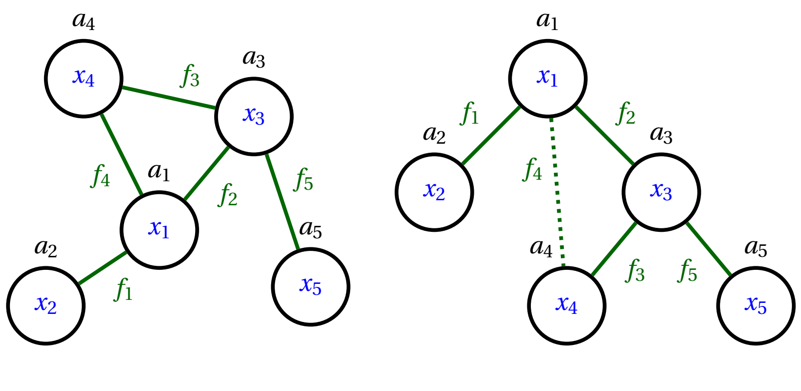
- $\alpha : \mathbf{X} \rightarrow \mathbf{A}$ is a mapping from variables to agents. The agent to which variable x_i is allocated is denoted as $\alpha(x_i)$. A common assumption is that the number of agents is equal to the number of variables, such that $a_i = \alpha(x_i)$ for $i = 1, \ldots, N$. Likewise, the set of agents associated with f_n is denoted by $\alpha(\mathbf{V}_n) = \{\alpha(x) \in \mathbf{A} \mid x \in \mathbf{V}_n\}$.
- η is an operator that combines all utility functions into the objective function. Common options are the *summation* operator $(\sum(\cdot))$ and the *maximum* operator $\left(\max(\cdot)\right)$. The objective function is defined by $G(\rho) = \underset{f_n \in \mathbf{F}}{\eta}\left(f_n(\rho_{\mathbf{V}_n})\right)$. The optimal value assignment is denoted by $\rho^* := \underset{\rho \in \Sigma}{\arg\max}\, G(\rho)$.

Analogous to the DCOP definition, a Continuous DCOP (C-DCOP) can be defined as a tuple $\langle \mathbf{A}, \mathbf{X}, \mathbf{D}, \mathbf{F}, \alpha, \eta \rangle$. The definitions of $\mathbf{A}$, $\mathbf{F}$, α, and η are identical to their definitions in a DCOP. The differences between a DCOP and a C-DCOP are the definition of the domain set and the variables. All variables in the variable set of a C-DCOP, $\mathbf{X}$, are continuous. The corresponding domain set is defined as $\mathbf{D} = \{\mathbf{D}_1, \ldots, \mathbf{D}_N\}$, where the domain for variable x_i is defined by a lower bound $\underline{d}_i$ and an upper bound $\overline{d}_i$ as $\mathbf{D}_i = [\underline{d}_i, \overline{d}_i]$.

The relation between the variables and the utility functions is typically represented as a constraint graph. In this representation, the agents are defined as nodes and the edges implicitly represent the utility functions. A constraint graph is often converted into a pseudo-tree to introduce a hierarchy to the agents. A pseudo-tree is a rooted spanning tree where the subproblems are contained in separate branches. All agents are assigned a single parent, which is an agent higher in the hierarchy. The only exception is the agent on top of the hierarchy, which is denoted as the root of the tree, this agent has no parent. An agent can have multiple children and the agents without children are denoted as the leaves of the tree.

In addition to parent/child relations, the pseudo-tree defines pseudo-parent/pseudo-child relations to indicate relations between agents over multiple hierarchy levels. Typically, the pseudo-tree is used as a communication structure, where agents only communicate between parent and child. In these cases, the pseudo relation allows for (indirect) interaction between pseudo-parent and pseudo-children. A graphical example of the two DCOP representations is given in Figure 4.2.

Producing a pseudo-tree from a constraint graph was introduced in the work of Freuder *et al.* [16]. Various methods exists for the generation of a pseudo-tree (e.g. pseudo-tree ordering [9], MLSP tree generation [31], and BFS construction [10]). A commonly used method is to produce a depth-first search (DFS) tree from the constraint graph by a DFS traversal. A DFS tree [43] is a special case of a pseudo-tree where the number of edges is equal to the number of edges in the constraint graph. This property ensures that all agents within the DFS tree are also connected in the constraint graph. If the DFS tree is used as a communication structure, this property ensures that agents only communicate if they share a utility function. Various algorithms exist in the literature that creates a DFS tree through a distributed procedure. The interested reader is referred to the works of



(a) Constraint graph **(b)** Pseudo-tree

Figure 4.2.: Graphical example for two representations of the same DCOP. A node represents an agent a_i and the edges indicate the utility functions f_n based on the variables x_i. In Figure 4.2a the nodes are unstructured. In Figure 4.2b the hierarchy is indicated by horizontal layers. The agent at the top (a_1) is the root of the tree, and the bottom agents (a_2, a_4, a_5) are the leaves. The edge (dotted line) between a_1 and a_4 specifies a pseudo-parent/pseudo-child relation.

Gallager *et al.* [17], Barbosa [4], Hamadi *et al.* [21], and Awerbuch [3] for implementation details.

4.3. Background of DCOP solvers

In the literature numerous solvers for DCOPs have been proposed; for a detailed overview, the reader is referred to Cerquides *et al.* [8] and Leite *et al.* [27]. As noted by Modi *et al.* [40], optimally solving a DCOP is NP-hard with regard to the number of variables and the cardinality of their domains. For this reason, complete (optimal) DCOP solvers are often not used in practice. In the literature, a diverse range of incomplete (near-optimal) DCOP solvers exist that trade off solution quality against computational requirements. Such solvers perform well for benchmark problems with domains with low cardinalities, such as graph coloring problems [40].

DCOP solvers are unable to directly solve C-DCOPs due to the definition of the domains. However, a C-DCOP can be discretized into a DCOP. One typically discretizes all domains of the C-DCOP using a grid-based approach that converts the continuous domains into discrete domains. This process can arbitrarily increase the cardinality of the domains, thereby rendering the use of DCOP solvers intractable.

For this reason, the development of C-DCOP solvers has recently gained a lot of attention within the literature. Initially, Stranders *et al.* [50] introduced the extension of the DCOP framework towards continuous variables (and domains) and proposed a continuous max-sum based (CMS) algorithm to solve C-DCOPs with continuous piecewise linear utility functions. The motivation behind this class of functions was that it can approximate all continuous functions arbitrarily close. In this approach, the domain discretization is replaced with a function approximation that has the same trade-off between resolution and solution quality. In a follow-up paper, Voice *et al.* [60] proposed a hybrid continuous max-sum (HCMS) algorithm without the function approximation. The HCMS algorithm extends the max-sum algorithm by incorporating a continuous non-linear optimization method. Within the algorithm, the domains are discretized and during optimization, the values are updated at every iteration based on a gradient descent method that depends on local utility values. An important parameter of the algorithm is the step size or learning factor with which the values are updated. The authors note that the step size parameter of the algorithm must be adjusted for the given problem, as small values will require numerous iterations while high values could result in overshooting of the optimum.

The concept of discretization of the continuous domains and the iteratively updating of their values is used to extend DCOP solvers of various classes. Based on the taxonomy introduced by Yeoh *et al.* [62] DCOP solvers can be divided into three classes:

Search-based solvers perform a distributed search over the local search space of the agents. These solvers are based on centralized search techniques such as best-first and depth-first to *reduce the search space* of the problem by exchanging messages between the agents. Examples are ADOPT [40], CoCoA [54], AFB [19], DSA [24], and DBA [61].

Inference-based solvers communicate accumulated information among agents to *reduce the problem size after every message* through dynamic programming methods. Well-known examples of this class of solvers are DPOP [43], the max-sum based algorithm [45], and action GDL [59].

Sampling-based solvers coordinate the sampling of the global search space guided by probabilistic measures. The probabilistic measures are calculated based on (all) preceding samples to *balance exploration and exploitation* of the global search space. At the time of writing, two sampling-based solvers are found in the literature: DUCT [42] and Distributed Gibbs [41].

The inference-based DPOP algorithm [43] is extended by Hoang *et al.* [22] into several algorithms, where AC-DPOP and the memory-limited variant CAC-DPOP can be applied to C-DCOPs without requirements on the utility functions. The search-based CoCoA algorithm [54] is extended in a similar manner by Sarker *et al.* [47] into C-CoCoA. Both Stranders *et al.* [50] and Sarker *et al.* [47] note that the local gradient descent approach cannot guarantee convergence to a global optimum. Initial domain discretization thus remains an important factor in the solution quality. An alternative approach to gradient-based optimization is presented by Choudhury *et al.* [11]. The proposed Particle Swarm Based F-DCOP (PFD) is based on Particle Swarm Optimization (PSO) [23]. PSO is a stochastic optimization technique in which multiple particles are assigned a random position and velocity. The positions represent domain values and the velocities contain implicit derivative information. The PFD algorithm guarantees convergence to a local optimum when the velocity of the best particle is reduced to zero. In conclusion, local gradient-descent-based and PSO-based algorithms will arguably find higher quality solutions compared to their discrete counterparts, but they remain dependent on the initial discretization of the domains and do not guarantee convergence to the global optimum.

In the literature, alternative approaches exist that are capable of extending the inference-based or search-based solvers toward continuous domains. A prominent example is the adaption of dynamic programming for Markov Decision Problems (MDPs) with continuous domains toward C-DCOPs. The interested reader is referred to the work of van Hasselt [55], and the references therein, for a reinforcement learning approach for learning policies for MDPs with continuous state and action spaces. Additionally, the work of Vianna *et al.* [58] extends symbolic dynamic programming techniques to solve discrete and continuous state MDPs.

Sampling-based solvers coordinate the sampling of the global search space guided by probabilistic measures, where samples are referred to as (partial) value assignments. Note that in the field of stochastic optimization samples are typically only defined in the context of functions perturbed by noise. The probabilistic measures are used to quantify the probability of finding samples that correspond with function outputs with high utility values. Sampling-based solvers iteratively select samples while taking previous iterations into account to balance exploration and exploitation of the global search space. Sampling-based solvers have not been extended towards the application to C-DCOPs, however, the iterative process allows for the selection of a sample from a continuous domain directly. The elimination of the discretization of the domains combined with the balanced search of the global search space makes sampling-based solvers highly promising to efficiently solve C-DCOPs. Therefore, in Section 4.4, the sample selection for C-DCOPs is addressed and a novel sampling-based solver for C-DCOPs is introduced in Section 4.5.

4.4. SAMPLE SELECTION FOR CONTINUOUS DCOPS

As mentioned in Section 4.3, a sampling-based solver will be introduced such that the relation between the value assignment and the corresponding utility will be taken into account within the optimization process. During the optimization process, a sample is defined as a value assignment of a variable. Samples of agents are combined and used as inputs for the utility functions to calculate the corresponding utility values. Within the literature several methods exist for sample generation:

- Upper Confidence Bound (UCB) sampling [2], developed for the reduction of regret for K-armed bandit problems [5].
- Gibbs sampling [18], constructed to approximate joint probability distributions in a Markov random field.
- Bayesian sampling [38], designed for the optimization of utility functions that are computationally expensive to evaluate.

UCB sampling has been applied to DCOPs by Ottens *et al.* [42] based on the UCB Applied to Trees (UCT) [25] and HOO [7]. The resulting algorithm, Distributed UCT (DUCT), generates samples based on previously returned values and the uncertainty (or confidence) over these values. All possible values (of the discrete domains) are sampled at least once before sampling is based on the confidence bounds. Gibbs sampling has been extended by Nguyen *et al.* [41] for the optimization of DCOPs based on the mapping of a DCOP to a maximum *a* posteriori (MAP) estimation problem. The MAP is found by approximating a joint probability distribution over all the variables. Samples are generated to approximate the joint probability distribution in a distributed manner. Note that several initial samples are required to accurately represent the desired distribution.

Both UCT and Gibbs sampling share a drawback compared to Bayesian sampling: these methods do not allow for the inclusion of *a* priori knowledge about the utility functions. This allows for Bayesian sampling to generate samples more efficiently than UCT and Gibbs. For this reason, in this chapter, Bayesian sampling is extended toward the application of C-DCOPs.

Bayesian sampling is based on Bayesian optimization which is a method to find the global optimum of a function in a sample-efficient manner, i.e. it minimizes the number of required samples. Bayesian optimization consists of two elements: a probabilistic model to approximate a (utility) function $f(\cdot)$, and an acquisition function $q(\cdot)$ to optimally select a new sample x_s, where s denotes the sample index. These two elements are discussed in more detail in Sections 4.4.1 and 4.4.2. Every input/output pair, $O_s = (x_s, y_s)$, is included in the ordered observations set $\mathcal{O}_S = \{O_1, \dots, O_S\}$, where S is the number of observations and $y_s = f(x_s)$ is the function (utility) value. The observations are used to update the probabilistic model, such that after every new observation the approximation is refined. Based on observations, the probabilistic model is used to estimate a mean function $\mu(x,\mathcal{O}) = \mathbb{E}\left[f(x)|\mathcal{O}\right]$ and the corresponding variance function $\sigma^2(x,\mathcal{O}) = \mathbb{E}\left[\left([f(x)|\mathcal{O}] - \mu(x,\mathcal{O})\right)^2\right]$. An overview of the Bayesian optimization algorithm is given in Algorithm 1.

4.4.1. Probabilistic model

The Gaussian process (GP) is a widely used probabilistic model to represent acquired knowledge about a function. More elaborate models exist, but these will often not share the computational benefit of the Gaussian process (GP) model. Using the Gaussian process (GP) model, a function $f(\cdot)$ is modeled based on a prior mean function $m(x) = \mathbb{E}\left[f(x)\right]$ and a kernel $\kappa(x, x') = \mathbb{E}\left[\left(f(x) - m(x)\right)\left(f(x') - m(x')\right)^{\mathrm{T}}\right]$. The kernel represents the cross-correlation between two values of a variable x, x'. The prior mean function and the kernel contain all (prior) knowledge of $f(\cdot)$. Typically, no prior information about the function is available and the zero function ($m(x) = 0$ for all x) is used as the prior mean function. In such cases, the modeling of the function depends mostly on the choice of the kernel.

Algorithm 1: Bayesian optimization [38]

```
Input  : $f(\cdot)$, $q(\cdot)$, $S$
Output: $\mu(x,\mathcal{O})$, $\sigma^2(x,\mathcal{O})$
/* Initialize the observation set                                  */
$\mathcal{O}_0 := \emptyset$;
for $s = 1, 2, \ldots, S$ do
    /* Select the next sample based on acquisition function        */
    $x_s := \arg\max_x q(x|\mathcal{O}_{s-1})$;
    /* Sample the utility function                                  */
    $y_s := f(x_s)$;
    /* Augment (and reorder) the observation set                    */
    $\mathcal{O}_s := \mathcal{O}_{s-1} \cup \{(x_s, y_s)\}$;
    /* Calculate the mean function and the variance function       */
    $\mu(x,\mathcal{O}) = \mathbb{E}\left[f(x)|\mathcal{O}\right]$;
    $\sigma^2(x,\mathcal{O}) = \mathbb{E}\left[\left([f(x)|\mathcal{O}] - \mu(x,\mathcal{O})\right)^2\right]$;
end
```

The Gaussian process (GP) model is combined with the observations to construct the joint Gaussian distribution over the function. From the joint Gaussian distribution, the posterior (distribution) can be found by using the Sherman-Morrison-Woodbury formula [49]:

$$P(f(x)|\mathcal{O}) = \mathcal{N}\left(\mu(x|\mathcal{O}), \sigma^2(x|\mathcal{O})\right), \tag{4.1}$$

where

$$\mu(x|\mathcal{O}) = \boldsymbol{k}(x|\mathcal{O})^{\mathrm{T}} \boldsymbol{K}^{-1}(\mathcal{O})\, \boldsymbol{y}(\mathcal{O}) \tag{4.2}$$

$$\sigma^2(x|\mathcal{O}) = \kappa(x,x) - \boldsymbol{k}(x|\mathcal{O})^{\mathrm{T}} \boldsymbol{K}^{-1}(\mathcal{O})\, \boldsymbol{k}(x|\mathcal{O}) \tag{4.3}$$

and $\mathcal{N}$ denotes the normal distribution, $\boldsymbol{K}(\mathcal{O})$ is the Gramian matrix of the kernel, defined by $(\boldsymbol{K})_{i,j} = \kappa(x_i, x_j)$ for all $i, j \in \{1, \ldots, S\}$, $\boldsymbol{k}(x|\mathcal{O}) = [\kappa(x_1, x), \ldots, \kappa(x_S, x)]^{\mathrm{T}}$ denotes the *cross-correlation* vector between the observations and x, and $\boldsymbol{y}(\mathcal{O}) = [y_1, \ldots, y_S]^{\mathrm{T}}$ denotes the observation value vector. The (posterior) mean and variance functions of the probabilistic model are denoted as $\mu(\cdot)$ and $\sigma^2(\cdot)$, respectively. Note that the posterior distribution contains the estimate of the function based on both the prior knowledge and the observations.

A wide range of kernels for Gaussian processes exist in the literature and the interested reader is referred to the work of Duvenaud *et al.* [13] for an overview of constructing kernels. An important kernel property is the ability to estimate every continuous function up to a required resolution given a sufficient number of observations. A kernel that possesses this property is called a universal kernel. In the work of Micchelli *et al.* [34], the conditions for a kernel to be universal in terms of properties of its features are given. The most commonly used universal kernel is the squared exponential kernel. A general description of the kernel and its properties is given by Vert *et al.* [57]. A drawback of the squared exponential kernel is that it can result in over-smooth approximations. For this reason, the Matérn kernel [35] is often used, since it can trade off differentiability and smoothness. In practice, the choice for a kernel depends on the properties of the function that needs to be approximated. All kernels have parameters that can be used to adjust their properties, such as smoothness and scaling. If information about $f(\cdot)$ is available, this should be incorporated in the selection of the kernel and its parameters. Typically, it is assumed that no information about $f(\cdot)$ is available, and then, as noted by Rasmussen *et al.* [44], the selection of the parameters is non-trivial. For this reason, the selection of parameters is often treated as a separate optimization problem [29]. It is commonly solved by using the maximum likelihood problem for which automatic relevance detection [30] is a widely used algorithm.

4.4.2. Acquisition function

The selection of the next sample is the result of the optimization of an acquisition function $q(\cdot)$, defined by

$$x_s = \arg\max_x q(x|\mathcal{O}).$$

The acquisition function depends on the posterior distribution in Equation (4.1) and thereby on all previous observations. Two commonly used acquisition functions are the probability of improvement function [26] and the expected improvement function [39]. The probability of improvement function considers the *probability* of finding a sample of which its value is larger than the maximum observed value. The maximum observed value is defined as

$$y^+ = \max\{y_s : (x_s, y_s) \in \mathcal{O}\}.$$

The corresponding maximum sample is defined as $x^+ = \{x_s : (x_s, y_s) \in \mathcal{O} | y_s = y^+\}$. As noted by Brochu *et al.* [6], the probability of improvement function focuses solely on the exploitation of already observed samples. To balance the exploration of the search space and exploitation of the observations, the expected improvement function will be used in this chapter. The expected improvement function chooses the sample based on the *expected value* of the next observation. The interested reader is referred to Brochu *et al.* [6] for a comparison of the two acquisition functions and more details. The expected improvement function can be written in closed form in terms of the mean and the deviation function of the probabilistic model as

$$q_{\mathrm{EI}}(x,\xi|\mathcal{O}) = \begin{cases} z(x,\xi|\mathcal{O})\Phi\left(\frac{z(x,\xi|\mathcal{O})}{\sigma(x|\mathcal{O})}\right) + \sigma(x|\mathcal{O})\phi\left(\frac{z(x,\xi|\mathcal{O})}{\sigma(x|\mathcal{O})}\right) & \text{if } \sigma(x|\mathcal{O}) > 0 \\ 0 & \text{if } \sigma(x|\mathcal{O}) = 0 \end{cases} \tag{4.4}$$

$$z(x,\xi|\mathcal{O}) = \mu(x|\mathcal{O}) - \left(y^+ + \xi\right) \tag{4.5}$$

where $\Phi(\cdot)$ is the Gaussian cumulative distribution function, $\phi(\cdot)$ is the Gaussian probability density function, and ξ is a design parameter. The design parameter can be used to trade off exploration and exploitation. As noted by Lizotte *et al.* [28], even a value as low as $\xi = 0$ will not result in a solely exploiting sampling behavior.

4.5. The Distributed Bayesian algorithm

In the previous sections, background information has been given about the DCOP framework and Bayesian optimization. In this section, the novel sampling-based C-DCOP solver Distributed Bayesian (D-Bay) is presented. This solver is capable of directly solving C-DCOPs without discretization of the domains. D-Bay uses Bayesian optimization as the probabilistic measure to optimize the sample selection. The overall procedure is similar to state-of-the-art sampling-based solvers, e.g. DUCT [42] and Sequential Distributed Gibbs (SD-Gibbs) [41].

Sampling-based solvers coordinate the sampling of the global search space guided by probabilistic measures to balance exploration and exploitation. The general outline of sampling-based solvers is as follows. Based on a pseudo-tree representation of the C-DCOP, the variables and utility functions are allocated to the agents. Afterward, two consecutive phases are iteratively repeated until a termination condition is satisfied. The first phase, the sampling phase, is top-down and starts from the root agent. The root starts the sampling phase by selecting a sample for all its variables. A sample can be viewed as a temporary value assignment of a variable. The sample is sent as a **sample** message to all the children of the agent. Upon receiving this message, an agent samples its variables and adds these samples to the **sample** message before sending it to its children. This process continues until the leaf agents are reached.

When the leaf agents are reached, the utility phase is initiated. This second phase is bottom-up and starts from the leaf agents. Based on the allocated utility functions, the agents calculate the utility (value) based on the **sample** message and the assignments of their variables. This utility is encoded within a **utility** message and sent to the parent of the agent. Upon receiving a **utility** message, an agent calculates the utility of its allocated utility functions. The resulting utility value is aggregated with the utility value of the received message before sending a **utility** message to its parent. This phase finishes when the root agent received a **utility** message from all its children. This moment marks the end of a single iteration. At this time, all agents have obtained the utility value associated with the sample of their variables. This information is used by the agents to update their probabilistic models and thereby the selection of their sample at the next iteration.

The main difference between sample-based solvers is in the method of selecting additional samples. The probabilistic measure in DUCT is based on confidence bounds of the utility of the samples and selects samples to improve these bounds. The agents store the utility for all previous samples during optimization. The Distributed Gibbs algorithm selects samples based on joint probability distributions and only keeps track of the differences between the utility values of the samples as a termination criterion. This makes Distributed Gibbs more memory efficient compared to DUCT.

Both algorithms are DCOP solvers and have a runtime complexity that is linear in the cardinality of the largest domain [14, Table 4]. Therefore, both Distributed Gibbs and DUCT suffer from the discretization of continuous domains and are not suitable for continuous DCOPs.

An additional disadvantage of both solvers is the non-determinism with regard to the utility value of a sample. This is caused by the consecutive sampling and utility phases since within an iteration all agents sample a single value from their local search space. In other words, the same **sample** message can result in different **utility** messages when the children of an agent select different samples for their variables.

To remove the non-determinism, the sampling and utility phase in D-Bay will be restricted to parents and children instead of the entire pseudo-tree. To be more precise, when a child receives a **sample** message it will first iterate between its children before sending a **utility** message to its parent. This will guarantee that the **utility** message in response to a **sample** message will always be identical.

D-Bay as described in Algorithm 2 (Appendix A) involves four phases:

(1) Pseudo-tree construction: The agents create a pseudo-tree from the constraint graph of the C-DCOP. Afterwards, every agent a_i knows its parent/children sets ($\mathbf{P}_i/\mathbf{C}_i$) and pseudo-parents/pseudo-children sets ($\mathbf{PP}_i/\mathbf{PC}_i$), where $\mathbf{P}_i, \mathbf{PP}_i, \mathbf{C}_i, \mathbf{PC}_i \subset \mathbf{A}$. The pseudo-tree is used as the communication structure in which agents only communicate with agents with whom they share a parent/child relation. Note that the agents only have local information on the pseudo-tree.

(2) Allocation of utility functions: Similar to the allocation of variables, all utility functions in $\mathbf{F}$ are exclusively allocated to the agents. Every agent a_i constructs two separate function sets based on the variables of the agent and the variables of its (pseudo-)parents. Firstly, the utility function set $\mathbf{F}_{a_i} = \{f_n \in \mathbf{F} : \alpha(\mathbf{V}_n) = \{a_i\}\}$, which only depends on the agent itself. Secondly, the shared utility function set, $\mathbf{F}_{\mathbf{P}_i} = \{f_n \in \mathbf{F} : (a_i \in \alpha(\mathbf{V}_n)) \wedge (\alpha(\mathbf{V}_n) \cap (\mathbf{P}_i \cup \mathbf{PP}_i) \neq \emptyset)\}$, involves the agent and its (pseudo-)parents. These two function sets are combined as $\mathbf{F}_i = \mathbf{F}_{a_i} \cup \mathbf{F}_{\mathbf{P}_i}$.

(3) Sample propagation: In this phase, every agent optimizes its variables through the Bayesian optimization method and exchanges **sample** and **utility** messages. By doing so, the assignments of the variables will converge to the global optimum of the objective function as will be shown in Section 4.6.2. The variables of a_i are defined as $\mathbf{X}_i = \{x_j \in \mathbf{X} \mid a_i \in \alpha(x_j)\}$. The variables are optimized based on the aggregate utility of all functions in set $\mathbf{F}_i$ and the functions of its children ($f_n \in \mathbf{F}_k$ for all $a_k \in \mathbf{C}_i$). Consequently, the aggregate utility values obtained by the root agent hold the utility values of the objective function.

Since a sample from (pseudo-)parents is required to calculate the utility of the functions in $\mathbf{F}_{\mathbf{P}_i}$, every agent a_i waits for a **sample** message from its parent. The phase is therefore initiated by a **sample** message from the root agent. The sample propagation phase finishes when a convergence threshold is reached by the root agent. Upon receiving **sample** message $\mathfrak{S}_j$ from its parent a_j, agent a_i samples its variables with respect to the functions in set $\mathbf{F}_i$. The samples are selected through the optimization of an acquisition function. Note that the acquisition function depends on both a kernel and all preceding samples. If the agent is a leaf agent, the agent can optimize its variables without considering the impact of its assignments on other agents. The agents with children augment the **sample** message of their parent with their sample as $\mathfrak{S}_i = \mathfrak{S}_j \cup \{\rho_{\mathbf{X}_i}\}$ and send this message to their children.

The agent then waits until it has received all **utility** messages from its children. Only then is the agent able to compute the aggregate utility and return a **utility** message to its parent. Note that the aggregate utility represents the optimal utility for the sample of the agent and all its (pseudo-)children. A **utility** message is defined as $\mathfrak{U}_i^j = \eta\left(\mathfrak{U}_i, \hat{\mathfrak{U}}_i\right)$, where $\mathfrak{U}_i = \min\limits_{\rho \in \Sigma_{\mathbf{X}_i}} \underset{f_n \in \mathbf{F}_i}{\eta} \left(f_n(\rho_{\mathbf{V}_n} \mid \mathfrak{S}_j)\right)$ and $\hat{\mathfrak{U}}_i = \underset{a_k \in \mathbf{C}_i}{\eta} \left(\mathfrak{U}_k^i\right)$ define the utility and the aggregated child utility, respectively. A graphical overview of the sample propagation phase is shown in Figure 4.3. Additionally, a partial trace of the Bayesian optimization is shown in Figure 4.4.

(4) Assignment propagation: The final phase is the assignment propagation phase, in which the root agent a_1 sends the final assignment of all its variables to its children as a **final** message $\hat{\mathfrak{S}}_1 = \{\hat{\rho}_{\mathbf{X}_1}\}$. Based on these assignments, the children assign their variables to the value corresponding to the optimal utility value. Afterwards, every agent adds its assignments to the **final** message as $\hat{\mathfrak{S}}_i = \{\hat{\rho}_{\mathbf{X}_i}\} \cup \hat{\mathfrak{S}}_j$. After the leaf agents have received a **final** message, all agents have completed their local assignments $\hat{\rho}_{\mathbf{X}_i}$. Note that typically no agent has information of the complete assignment, $\hat{\rho} = \{\hat{\rho}_{\mathbf{X}_i} : i = 1, \ldots, M\}$.

Proposition 1. The memory complexity of each agent in D-Bay is $O(S)$, where S is the number of samples at every iteration.

Proof. The optimization of the variables of an agent is restarted every time a **sample** message from the parent is received. An agent only needs to store the utility of the values based on the current (local) iteration to send the *best* utility value back to its parent, thereby restricting the memory requirement per agent to $O(S)$. $\square$

Proposition 2. The maximal message size for all messages in D-Bay is $O(t)$, where t is the maximal depth of the tree.

Proof. The **utility** message has a fixed size of $O(1)$ as it contains a single utility value related to the current sample message $\mathfrak{S}$. Both the **sample** message and the **final** message are appended with the sample of the agent before it is sent to the children of the agent. This limits the size of these messages to $O(t)$. $\square$

Proposition 3. The total number of messages sent by an agent in D-Bay is $O(cS^t)$, where t is the depth of the tree, S is the number of samples and c denotes the largest number of children.

Proof. When agent a_i receives a **sample** message from its parent it generates S samples that are sent to all its children $\mathbf{C}_i$. This process continues until the leaves of the pseudo-tree are reached and will bind the number of messages for an agent to $O(cS^t)$. $\square$

Proposition 4. The maximal runtime complexity of each agent in D-Bay is $O(S^t)$, where t is the depth of the tree.

Proof. For every **sample** message an agent receives it optimizes the value of its next sample through Bayesian optimization a total of S times. The runtime complexity of the agents is greatest for the leaf agents and therefore the maximum number of samples for an agent is $O(S^t)$. $\square$

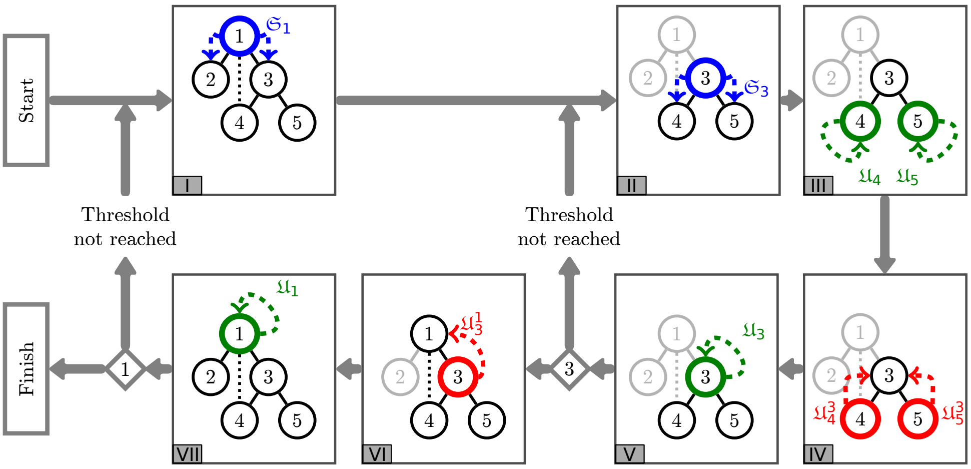
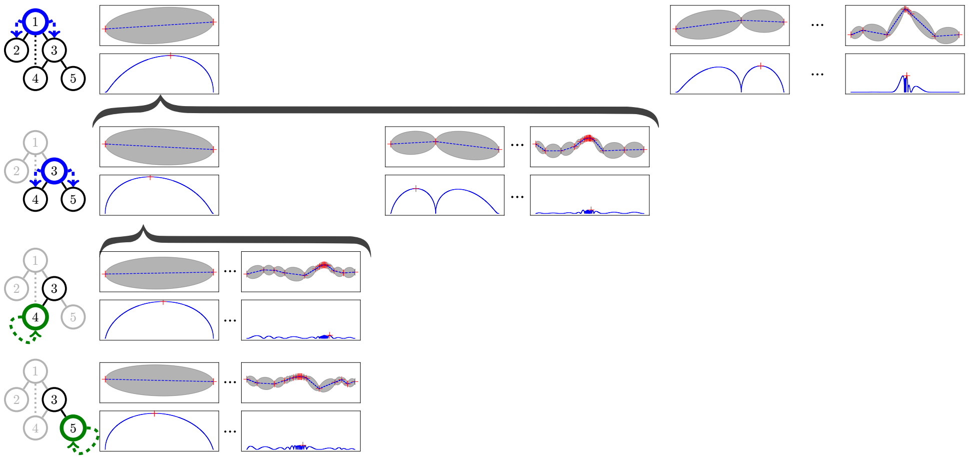
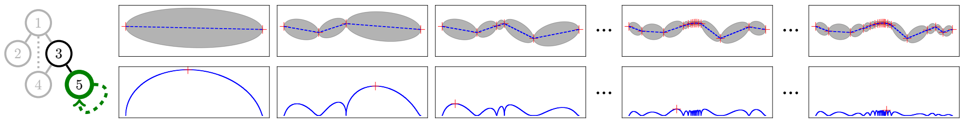


Figure 4.3.: Graphical overview of the sample propagation phase of D-Bay. Agents are indicated by circles labeled with an agent index, and utility functions are shown as black lines. Starting from the root a_1 (I), **sample** message $\mathfrak{S}_1$ is sent to its children (a_2, a_3). Subsequently, agent a_3 will send **sample** message $\mathfrak{S}_3$ to its children (II). After iterating between its children and calculating its local utility (III), agent a_3 combines all utilities (IV) and checks its threshold (V). The check for the threshold is indicated by an annotated grey diamond. If the threshold is reached the agent a_3 sends **utility** message $\mathfrak{U}_3^1$ to its parent (VI). This process is repeated when the root a_1 sends another **sample** message and finishes when a_1 the convergence threshold or the number of samples is reached (VII). Note that the interactions between a_1 and a_2 are not illustrated for brevity.



(a) Partial trace for simple DCOP during sample propagation phase. During the optimization, the children sample their variables based on the **sample** message of a parent. This dependency is indicated by the curly brackets, where the samples associated with a parent sample are contained within the curly bracket.



(b) Detailed view for agent a_5.

Figure 4.4.: Graphical examples of a partial trace of the Bayesian optimization process. The agent associated with the trace is shown on the left. The trace consists of the utility function approximation (top) and the acquisition function (bottom). The function approximation shows the samples (+), the mean (--), and the standard deviation (▨). The acquisition function shows both the values (—) and its optimum (+). The optimum of the acquisition function determines the next sample of the objective function.

In the next section, the convergence of D-Bay to the global optimum of a C-DCOP is analyzed. D-Bay utilizes Bayesian optimization for the sample selection within the sample propagation phase. For that reason, the performance of D-Bay depends highly on the properties of the Bayesian optimization method. As mentioned in Section 4.4, Bayesian optimization consists of the combination of a kernel and an acquisition function. Therefore, the analysis is focused on the selection of the kernel, the acquisition function, and their parameters.

4.6. THEORETICAL ANALYSIS OF D-BAY

This section analyses the convergence of D-Bay to the global optimum of a C-DCOP in two parts. Firstly, in Section 4.6.1, the convergence to the global optimum of the utility functions within the sampling propagation phase is proven. It shows that if the Lipschitz constant of the utility functions is known, the convergence to the global optimum can be guaranteed through the appropriate selection of the kernel and the acquisition function. In this chapter, all utility functions are assumed to be Lipschitz continuous with a known Lipschitz constant. A utility function $f(\cdot)$ is Lipschitz continuous with Lipschitz constant L_f if

$$|f(x_i) - f(x_j)| \leq L_f |x_i - x_j| \qquad \forall x_i, x_j \in \mathrm{dom}(f) \tag{4.6}$$

where $\mathrm{dom}(f)$ denotes the domain of the utility function.

Secondly, in Section 4.6.2, the convergence of D-Bay to the global optimum of the objective function based on the global optima of the utility functions is proven. This analysis focuses on the assignment propagation phase. The two parts of the analysis are combined to prove the convergence of D-Bay to the global optimum of C-DCOPs with utility functions with known Lipschitz constants.

4.6.1. CONVERGENCE OF BAYESIAN OPTIMIZATION BASED ON LIPSCHITZ CONTINUOUS FUNCTIONS

As shown by Törn *et al.* [52], the convergence to the global optimum of a function by Bayesian optimization can only be guaranteed through *dense* sampling of the domain of the function. For this reason, within the Bayesian optimization method, the acquisition function will need to produce dense samples. In the work of Vazquez *et al.* [56, Theorem 6], the expected improvement acquisition function, given by Equation (4.4), is proven to produce dense observations within its search region. The search region is defined in Definition 4.1.

Definition 4.1. The search region of the acquisition function $q_{\mathrm{EI}}(\cdot)$ (based on $f(\cdot)$ and $\mathscr{O}$) is defined by

$$\mathscr{S} = \{x \in \mathrm{dom}(f) : q_{\mathrm{EI}}(x, \xi | \mathscr{O}) > 0\}.$$

As a consequence of the dense sample generation property, $\mathscr{S}$ will converge to an empty set when the number of samples goes to infinity. Therefore, since the next sample is chosen from the search region ($x_s \in \mathscr{S}$), the global optimum (x^*) will be sampled for $s \to \infty$ if $x^* \in \mathscr{S}$. In other words, if the optimum inclusion ($x^* \in \mathscr{S}$) property holds, then global convergence is guaranteed.

Definition 4.2. The upper bound function $\bar{f}(x|\mathscr{O})$ of (a Lipschitz continuous) function $f(\cdot)$ over all observations in $\mathscr{O}$ is defined by

$$\bar{f}(x|\mathscr{O}) = \min\{L_f|x - x_s| + y_s : (x_s, y_s) \in \mathscr{O}\} \qquad \forall x \in \mathrm{dom}(f).$$

Definition 4.3. The upper bound region of $f(\cdot)$ holds all values of x for which the upper bound function $\bar{f}(\cdot)$ (Definition 4.2) is larger than the maximum observed value y^+ and is defined by

$$\mathscr{U} = \{x \in \mathrm{dom}(f) : \bar{f}(x|\mathscr{O}) > y^+\}.$$

To find the conditions for which the optimum inclusion holds, the upper bound region set is introduced. The upper bound region set $\mathscr{U}$ (Definition 4.3) is based on the upper bound function $\bar{f}(\cdot)$ (Definition 4.2). Note that by definition, $\bar{f}(x) \geq f(x)$ for all x and $\mathscr{U}$ does not include any observations in $\mathscr{O}$ since $\bar{f}(x|\mathscr{O}) = y_s \leq y^+$ for all x_s. As shown in Lemma 4.1, this region is guaranteed to include the global optimum if the optimum has not already been observed. A graphical example of sets $\mathscr{U}$ and $\mathscr{S}$, and $\bar{f}(\cdot)$ can be seen in Figure 4.5.

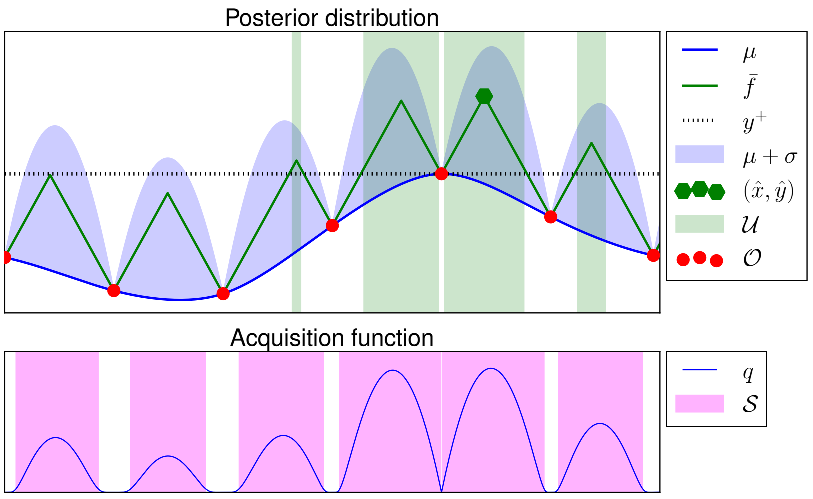


Figure 4.5.: Graphical overview of the sets $\mathscr{U}$, $\mathscr{S}$, and the upper bound function $\bar{f}(\cdot)$ based on the observations $\mathscr{O}$.

Lemma 4.1. *The upper bound region $\mathscr{U}$ includes the optimum sample x^* if it has not been observed. Formally, if $x^+ \neq x^*$, then $x^* \in \mathscr{U}$.*

Proof. By Definition 4.2 and Equation (4.6), the value of the upper bound function at the optimal sample is larger or equal to the optimal value. Formally, $\bar{f}(x^*|\mathscr{O}) \geq y^*$. If the optimal sample has not been observed ($x^* \neq x^+$), then $y^* > y^+$. Consequently, $\bar{f}(x^*|\mathscr{O}) > y^+$. Therefore, by definition of the upper bound region (Definition 4.3) the optimal sample is included ($x^* \in \mathscr{U}$). □

Based on the definition of the upper bound region, the optimum inclusion is satisfied when the set inclusion $\mathscr{U} \subseteq \mathscr{S}$ holds. The conditions for the set inclusion are given in two parts. Firstly, in Lemma 4.2 it is shown that if for all samples where the sum of the mean and deviation function is greater or equal to the highest sampled value, the sample is included in the search region set. Secondly, by Definition 4.3, the upper bound function defines all samples that are within the upper bound region set. By combining both conditions, we find that if the sum of the mean and deviation function is greater than the upper bound function, then $\mathscr{U} \subseteq \mathscr{S}$, as shown in Lemma 4.3.

Lemma 4.2. *If $\mu(x|\mathscr{O}) + \sigma(x|\mathscr{O}) \geq y^+ + \xi$ and $\sigma(x|\mathscr{O}) > 0$ then $x \in \mathscr{S}$.*

Proof. By Definition 4.1, $x \in \mathscr{S}$ if $q_{\mathrm{EI}}(x,\xi|\mathscr{O}) > 0$. Let $\sigma(x|\mathscr{O}) > 0$ and define $w(x,\xi|\mathscr{O}) = \frac{z(x,\xi|\mathscr{O})}{\sigma(x|\mathscr{O})}$ and through substitution rewrite Equation (4.4) as

$$q_{\mathrm{EI}}(x,\xi|\mathscr{O}) = z(x,\xi|\mathscr{O})\Phi\Big(w(x,\xi|\mathscr{O})\Big) + \sigma(x|\mathscr{O})\phi\Big(w(x,\xi|\mathscr{O})\Big). \tag{4.7}$$

Since $\sigma(x|\mathscr{O}) > 0$, we find $q_{\mathrm{EI}}(x,\xi|\mathscr{O}) > 0$ if $\frac{q_{\mathrm{EI}}(x,\xi|\mathscr{O})}{\sigma(x|\mathscr{O})} > 0$ where

$$\frac{q_{\mathrm{EI}}(x,\xi|\mathscr{O})}{\sigma(x|\mathscr{O})} = w(x,\xi|\mathscr{O})\Phi\Big(w(x,\xi|\mathscr{O})\Big) + \phi\Big(w(x,\xi|\mathscr{O})\Big).$$

Define $h(w) = w\Phi(w) + \phi(w)$. Then since $\Phi'(w) = \phi(w)$ and $\phi(w) = \frac{1}{\sqrt{2\pi}}\mathrm{e}^{-\frac{1}{2}w^2}$, we find $h'(w) = \Phi(w) + w\phi(w) - w\phi(w) = \Phi(w)$. Let $\mu(x|\mathscr{O}) + \sigma(x|\mathscr{O}) \geq y^+ + \xi$, then $z(x,\xi|\mathscr{O}) \geq -\sigma(x|\mathscr{O})$ and $w(x,\xi|\mathscr{O}) \geq -1$. For w in the interval $(-1,\infty]$ we find

$$h(w) = \int_{-1}^{w} h'(v)\mathrm{d}v + h(-1) = \int_{-1}^{w} \Phi(v)\mathrm{d}v - \Phi(-1) + \phi(-1) > 0,$$

since $\Phi(w) > 0$ for finite inputs, and $-\Phi(-1) + \phi(-1) > 0$. Therefore, if $\mu(x|\mathscr{O}) + \sigma(x|\mathscr{O}) \geq y^+ + \xi$ and $\sigma(x|\mathscr{O}) > 0$, then $q_{\mathrm{EI}}(x,\xi|\mathscr{O}) > 0$ and $x \in \mathscr{S}$. □

Lemma 4.3. *If $\mu(x|\mathscr{O}) + \sigma(x|\mathscr{O}) \geq \bar{f}(x|\mathscr{O}) + \xi$ for all $x \in \mathrm{dom}(f)$, then $\mathscr{U} \subseteq \mathscr{S}$.*

Proof. As shown in Lemma 4.2, if $\mu(x|\mathscr{O}) + \sigma(x|\mathscr{O}) \geq y^+ + \xi$ and $\sigma(x|\mathscr{O}) > 0$, then $x \in \mathscr{S}$. By definition of $\mathscr{U}$, for all $x \in \mathscr{U}$ we find $\bar{f}(x|\mathscr{O}) > y^+$. Additionally, for all $x \in \mathscr{U}$ we find $\sigma(x|\mathscr{O}) > 0$, since $\sigma^2(x|\mathscr{O}) = 0$ only if $x = x_s$ and by definition $x_s \notin \mathscr{U}$. Therefore, if $\mu(x|\mathscr{O}) + \sigma(x|\mathscr{O}) \geq \bar{f}(x|\mathscr{O}) + \xi$, then $x \in \mathscr{S}$ for all $x \in \mathscr{U}$. □

Note that $\mu(\cdot)$ and $\sigma(\cdot)$ depend on the kernel and $\bar{f}(\cdot)$ depends on the Lipschitz constant. This raises the question of which (type of) kernel is capable of explicitly associating its mean function and variance function to the Lipschitz constant.

An answer can be found in the work of Ding *et al.* [12], where Markovian class kernels are introduced as kernels that possess a Markovian posterior distribution. The value of a Markovian posterior distribution for a certain input value only depends on the observations surrounding that value. This property is beneficial as the upper bound function, which is directly related to the Lipschitz constant, possesses the same property. An additional benefit of this class of kernels is that the elements of $\boldsymbol{K}^{-1}(\mathscr{O})$ can be expressed analytically. This removes the need of inversion of a matrix of which the size grows with the number of observations, since $\boldsymbol{K}(\mathscr{O}) \in \mathbb{R}^{S\times S}$. As noted by Rasmussen *et al.* [44], this inversion is considered a major restriction to the practical application of Bayesian optimization. In general, a Markovian class kernel is defined by

4

$$\kappa(x_i, x_j) = \lambda^2\Big(p(x_i)g(x_j)\mathbb{I}_{x_i\le x_j} + p(x_j)g(x_i)\mathbb{I}_{x_i>x_j}\Big)$$

for some function $p(\cdot)$ and $g(\cdot)$, where $\mathbb{I}(\cdot)$ is the indicator function and λ is the kernel scale parameter. The observations are (re)ordered after every new observation, such that for scalar arguments $x_1 \le x_2 \le \cdots \le x_S$. The mean function $\mu_s(\cdot)$ and the variance function $\sigma_s^2(\cdot)$ of the posterior on the interval between observations for a kernel of this class is defined by,

$$\mu_s(x|\mathscr{O}) = \boldsymbol{\kappa}_s^{\mathrm{T}}(x,\mathscr{O})\boldsymbol{K}_s^{-1}(\mathscr{O})\boldsymbol{y}_s(\mathscr{O}) \tag{4.8}$$

$$\sigma_s^2(x|\mathscr{O}) = \kappa(x,x) - \boldsymbol{\kappa}_s^{\mathrm{T}}(x,\mathscr{O})\boldsymbol{K}_s^{-1}(\mathscr{O})\boldsymbol{\kappa}_s(x,\mathscr{O}) \tag{4.9}$$

for $x \in [x_{s-1}, x_s]$, where

$$\begin{aligned}\boldsymbol{\kappa}_s(x,\mathscr{O}) &= \begin{bmatrix}\kappa(x_1,x) & \dots & \kappa(x_{s-1},x) & \kappa(x_s,x) & \kappa(x_{s+1},x) & \dots & \kappa(x_S,x)\end{bmatrix}^{\mathrm{T}},\\ \boldsymbol{y}_s(\mathscr{O}) &= \begin{bmatrix}y_1 & \dots & y_{s-1} & y_s & y_{s+1} & \dots & y_S\end{bmatrix}^{\mathrm{T}},\end{aligned}$$

and $\boldsymbol{K}_s^{-1}(\mathscr{O})$ is a tridiagonal matrix of appropriate dimensions where the tridiagonal elements of $\boldsymbol{K}_s^{-1}(\mathscr{O})$, for $S \ge 3$ and if $\boldsymbol{K}_s(\mathscr{O})$ is nonsingular, are given by

$$(\boldsymbol{K}_s^{-1}(\mathscr{O}))_{s,s} = \begin{cases} \dfrac{\lambda^{-2}p(x_2)}{p(x_1)\big(p(x_2)g(x_1)-p(x_1)g(x_2)\big)}, & \text{if } s=1,\\[2ex] \dfrac{\lambda^{-2}\big(p(x_{s+1})g(x_{s-1})-p(x_{s-1})g(x_{s+1})\big)}{\big(p(x_s)g(x_{s-1})-p(x_{s-1})g(x_s)\big)\big(p(x_{s+1})g(x_s)-p(x_s)g(x_{s+1})\big)}, & \text{if } s\in\{2,\dots,S-1\},\\[2ex] \dfrac{\lambda^{-2}g(x_{S-1})}{g(x_S)\big(p(x_S)g(x_{S-1})-p(x_{S-1})g(x_S)\big)}, & \text{if } s=S,\end{cases}$$

and

$$(\boldsymbol{K}_s^{-1}(\mathscr{O}))_{s-1,s} = (\boldsymbol{K}_s^{-1}(\mathscr{O}))_{s,s-1} = \frac{-\lambda^{-2}}{\big(p(x_s)g(x_{s-1}) - p(x_{s-1})g(x_s)\big)},\quad s = 2,\dots,S.$$

All other elements of $\boldsymbol{K}_s^{-1}(\mathscr{O})$ are equal to zero.

Next, we show that for the Dirichlet kernel, as introduced by Ding *et al.* [12], the inequality of Lemma 4.3 will hold for all observations if the kernel scale λ is chosen appropriately. This kernel is selected over other Markovian class kernels because of its simplicity. The Dirichlet kernel defined by

$$\kappa_{\mathrm{d}}(x_i, x_j) = \lambda^2 \min(x_i, x_j)(1 - \max(x_i, x_j)) \tag{4.10}$$

for $x_i, x_j \in [0,1]$. Note that for κ_{d}, we find that $p(x) = x$ and $g(x) = (1-x)$. The mean function, given by Equation (4.8), and the variance function, given by Equation (4.9), corresponding to the Dirichlet kernel in the interval $[x_{s-1}, x_s]$ can be written as

$$\mu_s(x|\mathcal{O}) = \frac{y_{s-1}(x_s - x) + y_s(x - x_{s-1})}{x_s - x_{s-1}}, \tag{4.11}$$

$$\sigma_s^2(x|\mathcal{O}) = \lambda^2 \frac{-(x_s - x)(x_{s-1} - x)}{x_s - x_{s-1}}. \tag{4.12}$$

The derivation of Equations (4.11) and (4.12) can be found in Appendix B. It is important to note that both the mean function and the variance function on the interval $[x_{s-1}, x_s]$ only depend on the observations at the boundaries of the interval.

Based on Equations (4.11) and (4.12), the inequality of Lemma 4.3 holds if

$$\mu_s(x|\mathcal{O}) + \sigma_s(x|\mathcal{O}) \geq \bar{f}(x|\mathcal{O}) + \xi \text{ for } x \in [x_{s-1}, x_s] \tag{4.13}$$

for all $s \in \{1, \dots, S\}$, given $x_1 = 0$ and $x_S = 1$. In other words, by using the Dirichlet kernel, instead of analyzing the inequality in Lemma 4.3 over the entire domain of the function, it is sufficient to analyze Equation (4.13) on the intervals between the observations.

Now that the acquisition function and the kernel have been selected, we need to find their parameters (ξ and λ) such that the inequality of Equation (4.13) holds. These parameters can be appropriately chosen based on the Lipschitz constant as given in Theorem 4.4.

Theorem 4.4. *For a function $f(\cdot)$ with known Lipschitz constant L_f and* $\mathrm{dom}(f) = [0,1]$, *the Dirichlet kernel κ_{d}, and $S \geq 3$, where $x_1 = 0$ and $x_S = 1$, will yield $\mu(x|\mathcal{O}) + \sigma(x|\mathcal{O}) \geq \bar{f}(x|\mathcal{O})$ for all $x \in \mathrm{dom}(f)$ if $\lambda \geq L_f$.*

Proof. Let the functions $\mu_s(\cdot)$ and $\sigma_s(\cdot)$ be as defined by Equations (4.11) and (4.12), respectively. At the observations ($x = x_s$ for $s \in \{1, \dots, S\}$), the inequality $\mu(x|\mathcal{O}) + \sigma(x|\mathcal{O}) \geq \bar{f}(x|\mathcal{O})$ is satisfied, since $\mu_s(x_s|\mathcal{O}) + \sigma_s(x_s|\mathcal{O}) = \bar{f}(x_s|\mathcal{O}) = y_s$. Therefore, by letting $x_1 = 0$ and $x_S = 1$, only the closed intervals $x \in [x_{s-1}, x_s]$ for all $s \in \{2, \dots, S\}$ need to be examined. The proof will focus on these closed intervals next.

Based on Equations (4.11) and (4.12) the inequality $\mu(x|\mathcal{O}) + \sigma(x|\mathcal{O}) \geq \bar{f}(x|\mathcal{O})$ for the Dirichlet kernel at the closed intervals can be rewritten as

$$\mu_s(x|\mathcal{O}) + \sigma_s(x|\mathcal{O}) \geq \bar{f}(x|\mathcal{O}) \text{ where } x \in [x_{s-1}, x_s] \text{ for all } s \in \{2, \dots, S\}. \tag{4.14}$$

For the benefit of the analysis, we normalize the function input for every interval by defining a normalized function argument as

$$\tau_s(x|\mathcal{O}) = \frac{x - x_{s-1}}{\Delta x_s}. \tag{4.15}$$

Where $\Delta x_s = x_s - x_{s-1}$ is the size of the interval, and $\tau_s(x|\mathcal{O}) \in [0,1]$ for $x \in [x_{s-1}, x_s]$. All possible critical points of $\bar{f}(\cdot)$, $\hat{\tau}_s$, can be written as a function of $v_s(\mathcal{O})$ as

$$\hat{\tau}_s(v_s|\mathcal{O}) = \frac{1}{2} + \frac{\left(y_s - y_{s-1}\right)}{2L_f \Delta x_s} \tag{4.16}$$

$$= \frac{1}{2}\left(1 + v_s(\mathcal{O})\right), \tag{4.17}$$

where $v_s(\mathcal{O}) \in [-1,1]$ is defined as

$$v_s(\mathcal{O}) = \frac{y_s - y_{s-1}}{L_f \Delta x_s}.$$

Likewise, the upper bound function $\bar{f}(\cdot)$ (Definition 4.2) can be rewritten based on the normalized interval as

$$\bar{f}_s(x|\mathcal{O}) = \begin{cases} L_f \Delta x_s \tau_s(x|\mathcal{O}) + y_{s-1} & \text{if } 0 \le \tau_s(x|\mathcal{O}) < \hat{\tau}_s(v_s|\mathcal{O}), \\ L_f \Delta x_s (1 - \tau_s(x|\mathcal{O})) + y_s & \text{if } \hat{\tau}_s(v_s|\mathcal{O}) \le \tau_s(x|\mathcal{O}) \le 1. \end{cases} \tag{4.18}$$

A graphical overview of the normalized interval and the corresponding functions can be seen in Figure 4.6.

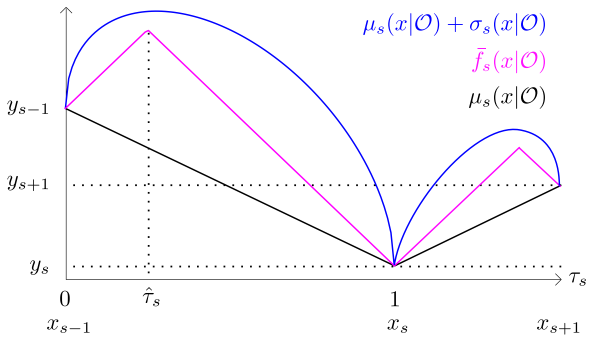


Figure 4.6.: Graphical overview of the normalized function argument $\tau_s(x|\mathcal{O})$ over its domain $[0,1]$. The domain corresponds to the interval $[x_{s-1}, x_{s+1}]$ where $\mu_s(\cdot)$ and $\sigma_s(\cdot)$ are based on the Dirichlet kernel and $\bar{f}_s(\cdot)$ is based on L_f and Δx_s. Based on the upper bound function, the domain can be divided into two intervals $[0, \hat{\tau}_s(v_s|\mathcal{O}))$ and $[\hat{\tau}_s(v_s|\mathcal{O}), 1]$.

After defining these variables, two separate intervals can be considered, $[0, \hat{\tau}_s(v_s|\mathcal{O}))$ and $[\hat{\tau}_s(v_s|\mathcal{O}), 1]$. Both these intervals will be investigated next.

Interval $[0, \hat{\tau}_s(v_s|\mathcal{O}))$**:** Let $\tau_s(x|\mathcal{O}) \in [0, \hat{\tau}_s(v_s|\mathcal{O}))$. The mean function, given by Equation (4.11), on the interval can be rewritten based on the normalized function argument as

$$\begin{aligned}\mu_s(x|\mathcal{O}) &= \frac{y_{s-1}(x_s - x) + y_s(x - x_{s-1})}{x_s - x_{s-1}}\\ &= y_{s-1}(1 - \tau_s(x|\mathcal{O})) + y_s\tau_s(x|\mathcal{O})\\ &= y_{s-1} + \tau_s(x|\mathcal{O})L_f\Delta x_s v_s(\mathcal{O}).\end{aligned} \tag{4.19}$$

Likewise, the variance function in Equation (4.12) can be rewritten as the deviation function as

$$\begin{aligned}\sigma_s(x|\mathcal{O}) &= \lambda\sqrt{\frac{-(x_s - x)(x_{s-1} - x)}{x_s - x_{s-1}}}\\ &= \lambda\sqrt{(1 - \tau_s(x|\mathcal{O}))\tau_s(x|\mathcal{O})\Delta x_s}.\end{aligned} \tag{4.20}$$

Substitution of Equations (4.18) to (4.20) into Equation (4.14) yields

$$\begin{aligned}\mu_s(x|\mathcal{O}) + \sigma_s(x|\mathcal{O}) &\geq \bar{f}_s(x|\mathcal{O})\\ \lambda &\geq (1 - v_s(\mathcal{O}))\sqrt{\frac{\tau_s(x|\mathcal{O})}{(1 - \tau_s(x|\mathcal{O}))}}L_f\sqrt{\Delta x_s}.\end{aligned} \tag{4.21}$$

Note that Equation (4.21) gives an explicit expression of the value for the kernel scale λ and all possible values of input/output pairs of the observations through the auxillary variable $v_s(\mathcal{O})$.

To analyze the values of λ for which the inequality of Equation (4.21) holds, the upper bound of the right-hand side is determined.

Since $\sqrt{\tau_s(x|\mathcal{O})/(1 - \tau_s(x|\mathcal{O}))}$ is monotonically increasing with respect to $\tau_s(x|\mathcal{O})$, we find for $\tau_s(x|\mathcal{O})$ in the interval $[0, \hat{\tau}_s(v_s|\mathcal{O}))$,

$$\begin{aligned}(1 - v_s(\mathcal{O}))\sqrt{\frac{\tau_s(x|\mathcal{O})}{(1 - \tau_s(x|\mathcal{O}))}} &\leq (1 - v_s(\mathcal{O}))\sqrt{\frac{\hat{\tau}_s(v_s|\mathcal{O})}{(1 - \hat{\tau}_s(v_s|\mathcal{O}))}}\\ &\leq (1 - v_s(\mathcal{O}))\sqrt{\frac{\frac{1}{2}(1 + v_s(\mathcal{O}))}{(1 - \frac{1}{2}(1 + v_s(\mathcal{O})))}}\\ &\leq \sqrt{1 - v_s(\mathcal{O})^2}\\ &\leq 1.\end{aligned}$$

Furthermore, since $\Delta x_s \leq \Delta x \leq 1$, we find through substitution of the upper bounds in Equation (4.21) that if $\lambda \geq L_f$, then Equation (4.21) is satisfied for all possible observations.

Interval $[\hat{\tau}_s(\nu_s|\mathcal{O}),1]$: For this interval the same approach is applied. Let $\tau_s(x|\mathcal{O}) \in [\hat{\tau}_s(\nu_s|\mathcal{O}),1]$; then substitution of Equations (4.18) to (4.20) in Equation (4.14) yields

$$\mu_s(x|\mathcal{O}) + \sigma_s(x|\mathcal{O}) \geq \bar{f}_s(x|\mathcal{O})$$
$$\lambda \geq (1+\nu_s(\mathcal{O}))\sqrt{\frac{(1-\tau_s(x|\mathcal{O}))}{\tau_s(x|\mathcal{O})}} L_f\sqrt{\Delta x_s}. \tag{4.22}$$

Since $\sqrt{(1-\tau_s(x|\mathcal{O}))/\tau_s(x|\mathcal{O})}$ is monotonically decreasing with respect to $\tau_s(x|\mathcal{O})$, we find

$$(1+\nu_s(\mathcal{O}))\sqrt{\frac{(1-\tau_s(x|\mathcal{O}))}{\tau_s(x|\mathcal{O})}} \leq 1. \tag{4.23}$$

Therefore, we conclude that for the interval $[\hat{\tau}_s(\nu_s|\mathcal{O}),1]$ if $\lambda \geq L_f$ then Equation (4.22) is satisfied for all possible observations.

In conclusion, if $\lambda \geq L_f$, we find that the inequality of Equation (4.14) will hold for the intervals $[x_{s-1}, x_s]$ for $s \in \{1,\ldots,S\}$. Since $x_1 = 0$ and $x_S = 1$, Equation (4.13) hold for all $x \in (0,1)$. Subsequently, $\mu(x|\mathcal{O}) + \sigma(x|\mathcal{O}) \geq \bar{f}(x|\mathcal{O})$ will holds for all $x \in [0,1]$. $\square$

According to Theorem 4.4, if $\lambda \geq L_f$ we find $\mu(x|\mathcal{O}) + \sigma(x|\mathcal{O}) \geq \bar{f}(x|\mathcal{O})$ for all $x \in \text{dom}(f)$. Applying Lemma 4.3 and setting $\xi = 0$, yields $\mathcal{U} \subseteq \mathcal{S}$ where $x^* \in \mathcal{S}$ for all observations. Subsequently, the Bayesian optimization will converge to the global optimum of the function. Note that for all functions without a normalized domain, the Lipschitz constant should be scaled according to the scaling required for the normalization of the domain.

4.6.2. Convergence of D-Bay based on global optima of utility functions

As shown in Section 4.6.1, all agents can find the global optimum of the aggregate utility of their utility functions and the utility functions of their children through Bayesian optimization. In this section, it will be shown that given the global optima of the utility functions, D-Bay will find the global optimum of the objective function.

During the sample phase of D-Bay, none of the agents can optimize their variables without interaction with other agents. The interaction involves the sending of (top-down) **sample** messages $\mathfrak{S}$ and (bottom-up) **utility** messages $\mathfrak{U}$. Therefore, for the leaf agents, the optimization depends on their utility functions and the **sample** message of their parent, $\mathfrak{S}_j$, as

$$\hat{\rho}_{\mathbf{X}_i} = \underset{\rho\in\Sigma_{\mathbf{X}_i}}{\arg\min}\,\eta\left(\mathfrak{U}_i\right) = \underset{\rho\in\Sigma_{\mathbf{X}_i}}{\arg\min}\,\eta\left(\underset{f_n\in\mathbf{F}_i}{\eta}\left(f_n(\rho_{\mathbf{V}_n} \mid \mathfrak{S}_j\right)\right) \qquad \forall a_i : \mathbf{C}_i = \emptyset. \tag{4.24}$$

When the kernel and acquisition function are selected as detailed in Section 4.6.1, the assignment $\hat{\rho}_{\mathbf{X}_i}$ is optimal with respect to the assignments of the (pseudo-)parents of agent a_i, since $\mathfrak{S}_j = \hat{\rho}_{\mathbf{P}_i} \cup \hat{\rho}_{\mathbf{PP}_i}$. Consequently, the optimal assignment results in the optimal value for the **utility** message $\mathfrak{U}_i^j$ given the **sample** message $\mathfrak{S}_j$.

This optimal value is sent as a **utility** message to the parents of the leaf agents and results in the following assignment for the other agents:

$$\hat{\rho}_{\mathbf{X}_i} = \underset{\rho \in \Sigma_{\mathbf{X}_i}}{\arg\min}\, \eta\left(\mathfrak{U}_i, \hat{\mathfrak{U}}_i\right) = \underset{\rho \in \Sigma_{\mathbf{X}_i}}{\arg\min}\, \eta\left(\mathop{\eta}_{f_n \in \mathbf{F}_i}\left(f_n(\rho_{\mathbf{V}_n} \mid \mathfrak{S}_j\right), \hat{\mathfrak{U}}_i\right) \qquad \forall a_i : \mathbf{C}_i \neq \emptyset. \tag{4.25}$$

The aggregated **utility** message $\hat{\mathfrak{U}}_i = \mathop{\eta}_{a_k \in \mathbf{C}_i}\left(\mathfrak{U}_k^i\right)$ is the aggregate of the optimal **utility** messages of all children given an assignment of a_i. Therefore, agent a_i can calculate the optimal assignment with regard to its parent **sample** message.

This process is repeated until the root agent a_1 has received all **utility** messages from its children. As the root agent does not have any (pseudo-)parents, Equation (4.25) can be rewritten as

$$\hat{\rho}_{\mathbf{X}_1} = \underset{\rho \in \Sigma_{\mathbf{X}_1}}{\arg\min}\, \eta\left(\mathfrak{U}_1, \hat{\mathfrak{U}}_1\right) = \underset{\rho \in \Sigma_{\mathbf{X}_1}}{\arg\min}\, \eta\left(\mathop{\eta}_{f_n \in \mathbf{F}_1}\left(f_n(\rho_{\mathbf{V}_n})\right), \hat{\mathfrak{U}}_1\right) = \rho^*_{\mathbf{X}_1}. \tag{4.26}$$

Note that $\hat{\mathfrak{U}}_1$ holds the aggregate utility value of all other agents based on the sample of the root agent. For that reason, if the root agent finds the optimal assignment $\hat{\rho}_{\mathbf{X}_1}$ it is equal to the optimum of the objective function $\rho^*_{\mathbf{X}_1}$.

After the root agent has found the optimal assignment of its variables it starts the final phase of D-Bay. In this phase the root agent sends the optimal assignment as a **final** message to its children, $\hat{\mathfrak{S}}_1 = \{\rho^*_{\mathbf{X}_1}\}$. Based on that optimal sample all agents are able to determine their optimal assignments, as shown in Equation (4.25), and append their optimal assignment to the final message before sending the final message to their children, i.e. $\hat{\mathfrak{S}}_i = \{\rho^*_{\mathbf{X}_i}\} \cup \hat{\mathfrak{S}}_j$. This process is repeated until the leaf agents are reached and all agents have assigned the globally optimal values to their variables.

4.6.3. Summary

In Section 4.6.1 it was shown that, based on the Lipschitz constant of a utility function (or aggregate of functions), the kernel and acquisition function (and their parameters) can be appropriately selected to guarantee convergence to the global optimum of the utility function. Subsequently, Section 4.6.2 has shown that, if the agents can find the global optimum of the aggregate of the utility functions, D-Bay will converge to the global optimum of the objective function. Combining these results proves the convergence of D-Bay to the global optimum of the objective function for utility functions with known Lipschitz constants.

4.7. SIMULATION RESULTS

In this section, the performance of the D-Bay algorithm is compared to DCOP solvers and C-DCOP solvers. The sampling-based DCOP solvers Sequential Distributed Gibbs (SD-Gibbs) [41] and DUCT [42] are chosen to compare the performance of the D-Bay algorithm with discrete solvers of the same class. Additionally, DPOP [43] is added to the comparison to represent the optimal solution of the DCOP solvers, since these solvers operate on the same domains and DPOP is a complete solver. The C-DCOP solvers AC-DPOP [22] and PFD [11] are selected as both achieve higher performance than the HCMS algorithm of Voice *et al.* [60].

The implementation of DPOP is included within the pyDCOP library [46]. All other algorithms have been included in the pyDCOP library and made available publicly[1]. The simulations are conducted on a 2.1 GHz Intel Xeon Gold 6152 CPU machine with sufficient memory for the requirements of all solvers and the computation time is limited to one hour. All algorithms are evaluated on two types of problems: random graphs and sensor coordination problems.

The hyperparameters of the solvers are fixed for all experiments and their values are listed in Table 4.1. If available, the values are taken equal to the listed values in the original works.

Table 4.1.: Hyperparameters of DCOP solvers used during simulations.

Algorithm	Hyperparameters
DPOP	-
SD-Gibbs	iterations = 20
DUCT	$\epsilon = 0.6, \delta = 0.1$
AC-DPOP	iterations = 100, $\delta = 0.001, \alpha = 0.01$
PFD	particles = 2000, $w = 0.9, c_1 = 0.9, c_2 = 0.1, \max_{f_c} = 5, \max_{s_c} = 15$
D-Bay	$\lambda = L_f$

4.7.1. RANDOM GRAPHS

For the generation of the random graph experiments, the NetworkX [20] generator, embedded within the pyDCOP library, is used. Based on the randomly created graph, a C-DCOP is generated by allocating a variable (and agent) to every node and defining utility functions for all edges. The bi-modal Bird function [36] (shown in Figure 4.7) is used as utility function. The Bird function was created as a test function for global optimization and contains multiple local optima at different values.

The experiments are defined based on three parameters: number of constraint checks, number of agents $|\mathbf{A}|$, and density of the graph p_1. The density of the graph is defined as

[1] https://gitlab.com/jfransman/pyDcop/

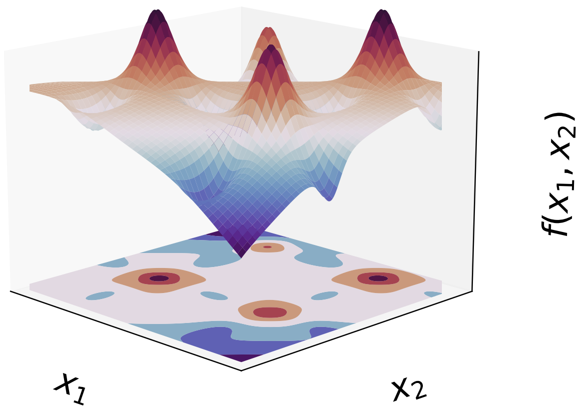


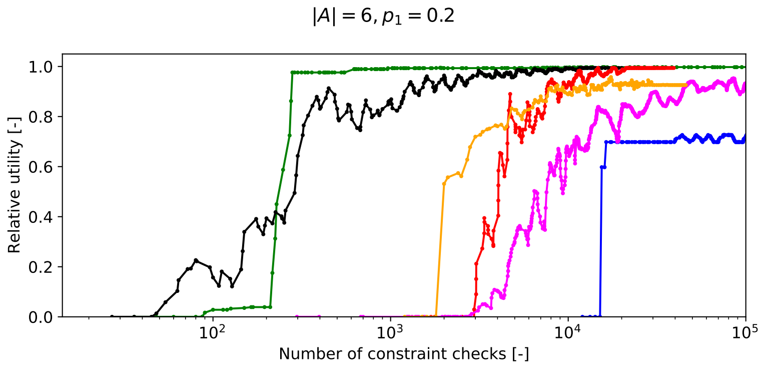
Figure 4.7.: Combined view of the function values and a contour plot of the Bird function [36] on the domain $x_i \in [-2\pi, 2\pi]$ for $i = 1,2$.

the ratio between the number of edges and the maximal number of possible edges. The number of constraint checks is used as a parameter to compare the DCOP and C-DCOP solvers based on computational efficiency. The efficiency of the solver is an important measure for problems for which the utility functions are computationally expensive to evaluate. Note that when comparing solvers for problems in which communication is the bottleneck non-concurrent constraint checks [32] are more suitable.

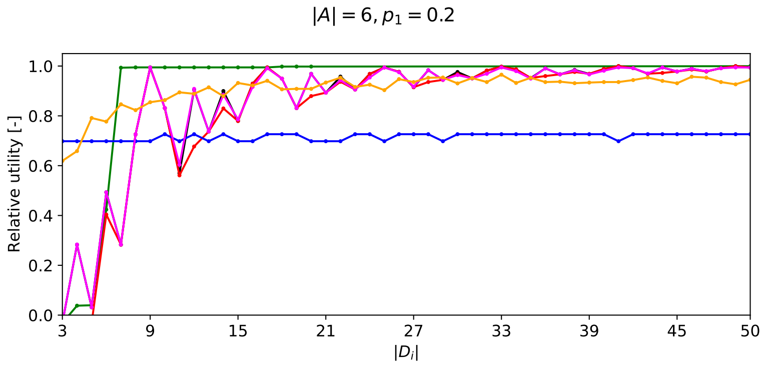
For all DCOP solvers, the continuous domains are uniformly discretized in a preprocessing step to convert the C-DCOP into a DCOP. The domain cardinality of the generated DCOPs is related to the constraint checks during the solving procedure. More values within a domain will constitute more possible evaluations of the utility functions. The C-DCOP solvers operate directly on the continuous domains; however, most solvers have a parameter that is analogous to the domain cardinality. The AC-DPOP algorithm requires the discretization of the domains within a preprocessing step. As Hoang *et al.* [22] provides no method for defining the optimal level of discretization, the discretization is set equal to that of the DCOP. The PFD algorithm initiates by selecting a random value for every particle using a uniform distribution from the domains and updates the values during every iteration. The D-Bay algorithm samples the continuous domains dynamically without discretization.

Based on the three defined parameters, numerous experiments are conducted; the number of agents is varied from 3 to 10, and the graph density is varied from 0.1 to 0.4 with increments of 0.02. All experiments are repeated 50 times and the median of the most illustrious results are shown in Figure 4.8.

In Figures 4.8a and 4.8b the relative utility is calculated based on the utility found by a centralized exhaustive search-based algorithm on densely discretized domains. The relative utility shows several important properties of the solvers. In Figure 4.8a all solvers show an increase of relative utility that converges when the number of constraint checks



(a) Relative utility for varying numbers of constraint checks.



(b) Relative utility for varying domain cardinality $|D_i|$.

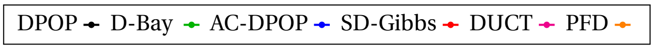


Figure 4.8.: Experimental results for the random graph problems.

is increased. The performance of the C-DCOP solvers differs significantly. The AC-DPOP solver shows a nearly constant performance but requires a large number of constraint checks. This is expected as the AC-DPOP solver initially starts with domain values that are equal to that of the discrete solvers and then the agents update their values based on a local gradient descent method. Upon further investigating the cause of the constant performance, we found that most of the domain values converge to the same local optima. This effectively reduces the number of domain values that are evaluated. During optimization, the utility values at the local optima of the Bird function are sent to the parents of the agents after which these values are interpolated. This (linear) interpolation results in a large overestimation of the utility values between the optima, therefore the solver does not escape the local optima and the performance does not improve when the constraint checks in increased.

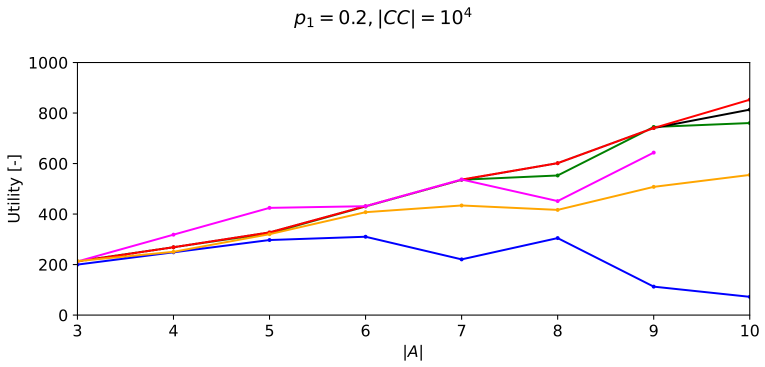
The performance of the PFD solver steadily increases for larger numbers of constraint checks. This behavior is expected since the domain values are updated during optimization based on the shared particles. The particles represent (partial) allocations of the variables within the C-DCOP. The relatively large number of particles (2000) increases the chance of producing particles that represent high utility allocations. These particles influence other particles during their value updates, thereby improving performance. This effect also holds for particles that are initialized near local optima. These particles influence other particles by driving them away from regions with high global utility.

Compared to other solvers, D-Bay shows the most consistent performance, in terms of achieved relative utility. It shows a (near) monotonic increase in performance. This can be explained by the property of the Bayesian optimization. Bayesian optimization combined with the appropriate choice of the kernel based on the Lipschitz constant ensures that all samples are selected such that the largest amount of information about the optima is gained. This allows for the exclusion of large regions of the domains, which effectively focuses the sampling on high-utility areas of the search space.

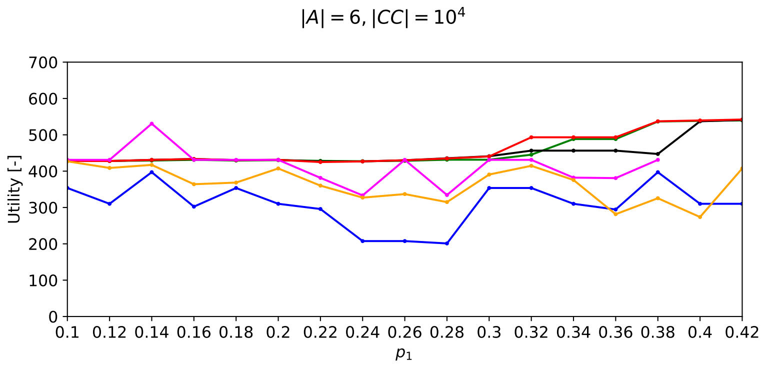
The results of the DCOP solvers (DPOP, SD-Gibbs, DUCT) show increasing utility that approaches the optimum. Compared to the C-DCOP solvers, the utility does not increase as smoothly when the number of constraint checks is increased. Upon further investigation, the performance of the DCOP solvers shows a clear dependency on the selection of the domain values. This is visible by the irregular increase of the relative utility for increasing domain cardinality as shown in Figure 4.8b. For a domain cardinality of 9, the discretization resulted in an excellent performance. This highlights a drawback of solvers that are dependent on discretization, as increasing the cardinality will not always guarantee better performance. In other words, there is no monotonic relation between the cardinality of the domains and the performance. The results of the DCOP solvers (DPOP, SD-Gibbs, DUCT) can be seen to overlap significantly. The DPOP algorithm yields the optimal solution of the discretized C-DCOP. This indicates that both SD-Gibbs and DUCT show close-to-optimal performance with high consistency. DUCT outperforms SD-Gibbs for low values of the domain cardinality however, for all higher values, SD-Gibbs achieves close to optimal performance. However, these algorithms require more constraint checks as can be seen in Figure 4.8a.

The performance of the D-Bay algorithm is very close to the optimum for 5×10^2 constraint checks. Similar performance is consistently achieved by the DCOP solvers at 1×10^4 constraint checks. To show D-Bay has a high sample efficiency independent of the number of agents and the density of the graph, these values are used in the comparison for the experiments with a varying number of agents and graph density. The results are shown in Figure 4.9.

Figure 4.9a shows the results for problems with a graph density of 0.2, where the number of agents is varied from 3 to 10. In Figure 4.9b the results for a 6 agent problem are shown, where the graph density is varied from 0.1 to 0.4 with increments of 0.02. Within both figures, the performance for all solvers is shown to be closely related to the results from Figure 4.8b. This demonstrates the fact that the sample efficiency of D-Bay holds for numerous random graphs. In other words, D-Bay achieves high performance with



(a) Solution quality for varying numbers of agents.



(b) Solution quality for varying graph density.

DPOP ●	D-Bay ●	AC-DPOP ●	SD-Gibbs ●	DUCT ●	PFD ●

Figure 4.9.: Comparison of performance based on a varying number of agents and graph density.

a limited number of samples and this performance is achieved by the compared algorithms only for a larger number of (evaluated) domain values.

4.7.2. SENSOR COORDINATION PROBLEM

The sensor coordination problem is an optimization problem in which every agent needs to orient its sensor to observe targets as accurately as possible. A real-world analogy would be the optimization of the orientation of multiple cameras based on image recognition. This problem is modeled within the DCOP framework as a distributed problem. The image recognition process can require significant computational effort depending on the image quality and the type of target(s). It is therefore computationally intensive to check every orientation of the camera, especially for a centralized approach. Even for a relatively small number of agents, a DCOP representation of these problems will result in a large search space.

Within the sensor coordination problem, all sensors are modeled identically in terms of their sensor range l and angle of view β. These properties, combined with the position of the sensor, determine the observation domain of the sensor. This domain defines all locations that could be observed by the sensor. The orientation ω_i of the sensor of agent a_i determines the observed area within the observation domain. A target is detected when it is located within this area. For every detected target, a (positive) utility value is allocated to the agent. The maximum utility is allocated when the sensor is oriented directly at the target. The utility value decreases linearly towards the edges of the observation area. The optimal utility is determined by the optimal solution of the centralized optimization approach with 720 samples (0.5 degree resolution) for every domain. The parameters of the problem are the number of targets T, the number of sensors N, the sensor range l, the angle of view β of the sensors, and the arrangement of the sensors. The sensors are arranged in an equally distanced rectangular grid. Various configurations are simulated, where a configuration indicates the number of rows and columns of the grid. The sensors are positioned such that the combined observation domains of all sensors are maximized without allowing unobservable areas between the sensors. For this reason, the distance between the sensors of the same row or column is $\sqrt{2}l$. The locations of the targets t are uniformly distributed within the combined observation domains of the sensors. In the experiments, the problems are generated with 6 sensors, 12 targets, and with identical sensor properties, where the sensor range is set to $l = 1$ and the angle of view is set to $\beta = 36°$. A graphical example of the simulated sensor coordination problems can be seen in Figure 4.10. The sensor coordination problem is described within the C-DCOP framework as follows:

- $\mathbf{A} = \{a_1, \dots, a_M\}$ is the set of agents, where M is the number of agents. The position of agent i is denoted as $p_i \in \mathbb{R}^2$.
- $\mathbf{X} = \{\omega_1, \dots, \omega_N\}$ is the set of sensor orientations, where $N = M$.
- $\mathbf{D} = \{\mathbf{D}_1, \dots, \mathbf{D}_N\}$, where $\mathbf{D}_i = (-180°, 180°)$ for all $i = 1, \dots, N$ indicating all possible values of sensor orientation ω_i.
- $\mathbf{F} = \{f_n\}_{n=1}^{T}$ is the set of utility functions associated with the observation of the targets. The number of targets is denoted by $T \in \mathbb{N}$. Target n is located at position

$t_n \in \mathbb{R}^2$. The utility functions of the targets are described as $f_n = \max_{i=1,\ldots,N}\left(f_{n,i}\right)$ for $n = 1,\ldots,T$, where

$$f_{n,i} = \begin{cases} 1 - |\omega_i - \angle\overrightarrow{p_i t_n}|/\beta & \text{if } \|\overrightarrow{p_i t_n}\| \leq l \text{ and } |\omega_i - \angle\overrightarrow{p_i t_n}| \leq \beta \\ 0 & \text{otherwise} \end{cases}$$

and $\overrightarrow{p_i t_n}$ denotes the vector between the location of the target t_n and the position of the agent a_i. Figure 4.11 shows an example of the utility value as a function of the angle of view.

- $\alpha(\omega_i) = a_i$ for $i = 1,\ldots,N$ allocating a single sensor to every agent.
- $\eta = \sum(\cdot)$, resulting in the goal function $G(\cdot) = \sum_{f_n \in F} f_n(\cdot)$.

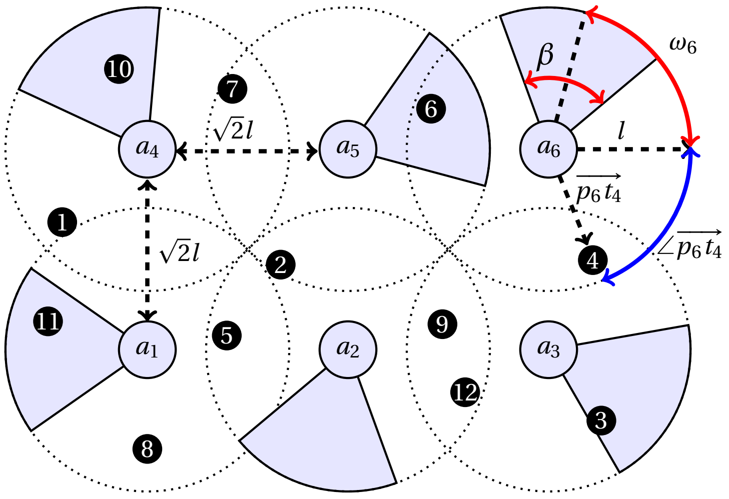


Figure 4.10.: Graphical example of a sensor coordination problem with 6 sensors and 12 targets. The sensors a_i are arranged in an equally distanced rectangular grid. The distance between the sensors is based on the sensor range l. The observation domain is indicated by a dotted circle centered around the position of the sensor. The observed area of the sensors is shown as shaded areas and is based on the angle of view β and the orientation ω_i. The targets are shown as annotated black circles.

In this section, the performance of D-Bay is empirically evaluated using the achieved relative utility as a function of the number of samples. This metric is important to consider if the evaluation of the utility functions by the agents is computationally expensive. The relative utility allows for the comparison of the results over various randomly generated problems. The achieved relative utility is defined as the achieved utility divided by the optimal utility generated by centralized exhaustive search based on densely discretized domains. The number of samples is defined as the maximum number of domain values checked by an agent and allows for a comparison based on sample efficiency per agent instead of the algorithm as a whole. For the DCOP solvers, the number of samples (per

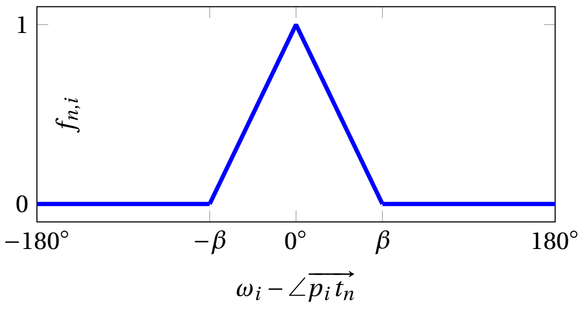


Figure 4.11.: Utility function $f_{n,i}$ indicates the utility of agent a_i for the observation of target t_n if the target is within the sensor range l ($\|\overrightarrow{p_i t_n}\| \leq l$). The angle of view of the sensor is denoted as β. On the horizontal axis the difference between the sensor orientation ω_i and the angle between the position of the agent and the position of the target $\angle\overrightarrow{p_i t_n}$ is given.

agent) is equal to the domain cardinality of the variables of the discretized C-DCOPs. The C-DCOP solvers iteratively update the domain values by either local gradient descent (AC-DPOP), particle velocity update (PFD), or sampling (D-Bay). In order to compare these solvers, the number of updates of the value of the root agent is used.

The performance results of D-Bay compared to the DCOP and C-DCOP solvers are given in Figure 4.12 for various configurations of the sensor coordination problem. Similar to the results of D-Bay in Figure 4.8a, the sample efficiency of D-Bay enables it to outperform both the DCOP solvers as well as the C-DCOP solvers. The performance of PFD slightly surpasses D-Bay for the 1x2 configuration but falls behind all other solvers for the 2x2, 2x3, and 3x2 configurations. This indicates that the performance of PFD largely depends on the random initialization of the particles.

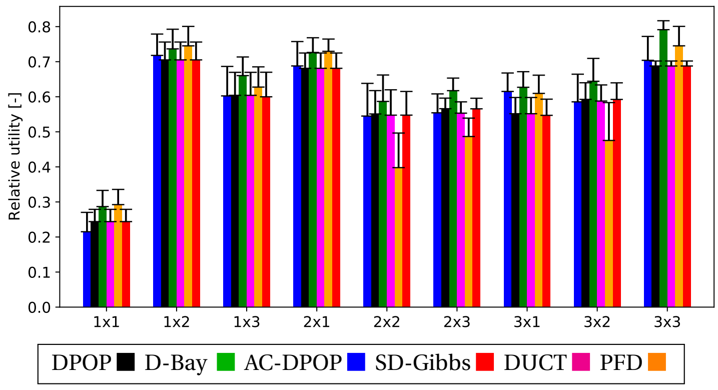
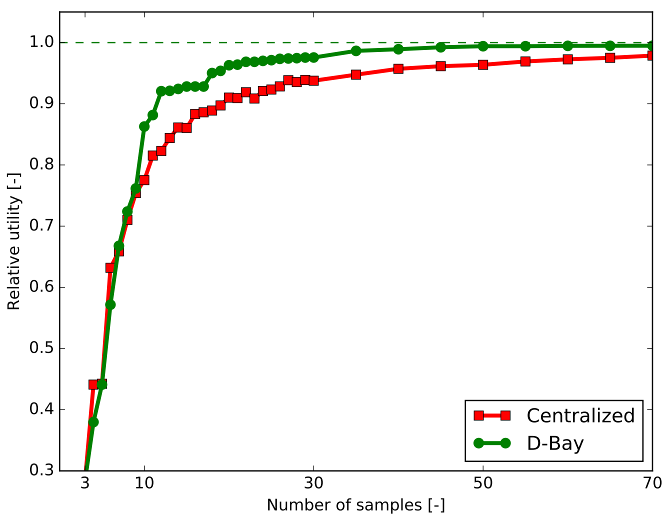
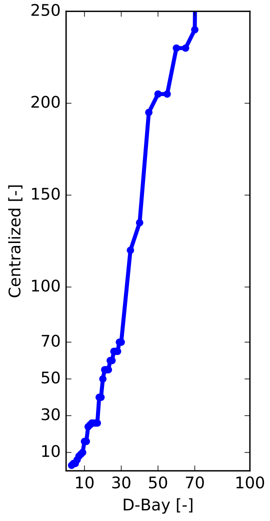


Figure 4.12.: Sensor configurations for 10 targets and 10 number of samples.

The performance results of D-Bay compared to the centralized approach are presented in Figure 4.13. This figure shows the results for 30 randomly generated problems for 6 sensors and 12 targets. The results show an increase in the achieved relative utility of D-Bay compared to the centralized approach based on the number of samples. The difference in achieved utility can be explained by investigating the sampling strategies. The centralized approach samples the sensor orientations equidistantly. Therefore, as the number of samples is increased, the resolution of the samples increases uniformly for the centralized approach. D-Bay samples dynamically to balance exploration and exploitation based on all previously acquired observations. Consequently, D-Bay will initially focus on exploration and eventually focus on exploitation. This behavior is clearly visible in Figure 4.13a in the range between samples 3 and 10. Within this range, D-Bay samples the sensor orientations equidistantly focussing on exploration. The sampling behavior is identical to the centralized approach, which can be seen in the similarity in achieved utility. For more than 11 number of samples, the achieved utility of D-Bay increases substantially. This can be explained based on the angle of view of 36° of the sensors during the experiments. At 10 samples the entire observation domain of a sensor is observed. Afterward, the switch to the exploitation of the observations increases the achieved utility more than the continued exploration of the centralized approach. The advantage is even more prominent when comparing the number of samples required by the centralized approach to achieve equal utility to D-Bay, as shown in Figure 4.13b. This clearly shows the advantage of the dynamic sampling of D-Bay over equidistant sampling.



(a) Achieved relative utility.



(b) Required number of samples for equal utility.

Figure 4.13.: Simulation results for randomly generated sensor coordination problems with 6 sensors and 12 targets. The figures show the average result of 30 randomly generated problems. Figure 4.13a shows the achieved utility of D-Bay and the centralized approach relative to the optimum. Note that the achieved utility for both algorithms is equal for 3 samples since the first 3 samples are equidistantly spaced for D-Bay. Figure 4.13b shows the number of samples required by the centralized approach to achieve the same utility as D-Bay.

4.8. CONCLUSIONS

In this chapter, the novel algorithm called Distributed Bayesian (D-Bay) has been introduced to solve Continuous Distributed Constraint Optimization Problems (C-DCOPs). Within D-Bay, the continuous domains are sampled based on Bayesian optimization. This removes the need for the discretization of the domains and balances exploration and exploitation of the global search space by incorporating knowledge about the utility functions within the kernels of the probabilistic models. Compared to DCOP solvers, which require discretization of the C-DCOP, it results in a reduction of the computational and memory demands of the individual agents. For utility functions with known Lipschitz constants, D-Bay is proven to converge to the global optimum solution of the C-DCOP.

Random graphs and sensor coordination problems have been used to evaluate the performance of D-Bay. The results show that D-Bay outperforms a centralized approach as well as state-of-the-art DCOP and C-DCOP solvers based on the achieved utility as a function of the required number of samples. This sample efficiency is a result of the application of Bayesian optimization with the D-Bay algorithm. Implementations of the proposed algorithm and the state-of-the-art DCOP and C-DCOP solver have been added to the open-source software library pyDCOP [46] and made available publicly[2].

In future work, D-Bay will be extended towards dynamic DCOPs [14] in which the agents need to optimize a dynamic problem at every time step. An extension will increase the applicability of the proposed algorithm to dynamic real-world problems in which tracking of targets is an important factor, such as multi-agent surveillance. In a dynamic adaptation of the sensor coordination problem, the locations of the targets change over time based on the target properties, such as velocity and turn radius.

ACKNOWLEDGMENT

I would like to thank dr. Peyman Mohajerin Esfahani of the Delft Center for Systems and Control at the Delft University of Technology for his insightful comments.

[2]https://gitlab.com/jfransman/pyDcop/

REFERENCES

[1] J. J. Acevedo, B. C. Arrue, I. Maza, and A. Ollero. "Cooperative large area surveillance with a team of aerial mobile robots for long endurance missions". In: *Journal of Intelligent and Robotic Systems: Theory and Applications* 70 (2013), pp. 329–345. DOI: 10.1007/s10846-012-9716-3.

[2] P. Auer, N. Cesa-Bianchi, and P. Fischer. "Finite-time analysis of the multiarmed bandit problem". In: *Machine learning* 47.2-3 (2002), pp. 235–256. DOI: 10.1023/A:1013689704352.

[3] B. Awerbuch. "A new distributed depth-first-search algorithm". In: *Information Processing Letters* 20.3 (1985), pp. 147–150. DOI: 10.1016/0020-0190(85)90083-3.

[4] V. C. Barbosa. *An introduction to distributed algorithms*. Mit Press, 1996. URL: https://mitpress.mit.edu/books/introduction-distributed-algorithms.

[5] J. A. Bather, D. A. Berry, and B. Fristedt. *Bandit Problems: Sequential Allocation of Experiments*. Vol. 149. 3. 1986, p. 271. DOI: 10.2307/2981558.

[6] E. Brochu, V. M. Cora, and N. de Freitas. "A tutorial on Bayesian optimization of expensive cost functions, with application to active user modeling and hierarchical reinforcement learning". In: *arXiv* (2010). arXiv: 1012.2599.

[7] S. Bubeck, R. Munos, G. Stoltz, and C. Szepesvári. "X-armed bandits". In: *Journal of Machine Learning Research (JMLR)* 12.5 (2011), pp. 1655–1695. URL: https://jmlr.org/papers/volume12/bubeck11a/bubeck11a.pdf.

[8] J. Cerquides, A. Farinelli, P. Meseguer, and S. D. Ramchurn. "A tutorial on optimization for multi-agent systems". In: *The Computer Journal* 57.6 (2014), pp. 799–824. DOI: 10.1093/comjnl/bxt146.

[9] A. Chechetka and K. Sycara. "A decentralized variable ordering method for distributed constraint optimization". In: *International Conference on Autonomous Agents and Multiagent Systems (AAMAS)*. 2005, pp. 1307–1308. DOI: 10.1145/1082473.1082746.

[10] Z. Chen, Z. He, and C. He. "An improved DPOP algorithm based on breadth first search pseudo-tree for distributed constraint optimization". In: *Applied Intelligence* 47.3 (2017), pp. 607–623. DOI: 10.1007/s10489-017-0905-4.

[11] M. Choudhury, S. Mahmud, and M. M. Khan. "A particle swarm based algorithm for functional distributed constraint optimization problems". In: vol. 34. 05. Apr. 2020, pp. 7111–7118. DOI: 10.1609/aaai.v34i05.6198.

[12] L. Ding and X. Zhang. "Scalable stochastic kriging with Markovian covariances". In: *arXiv* (2018). arXiv: 1803.02575.

[13] D. K. Duvenaud, H. Nickisch, and C. E. Rasmussen. "Additive Gaussian processes". In: *Advances in Neural Information Processing Systems (NIPS)*. 2011, pp. 226–234. DOI: 10.2495/DSHF120121.

[14] F. Fioretto, E. Pontelli, and W. Yeoh. “Distributed constraint optimization problems and applications: a survey”. In: *Journal of Artificial Intelligence Research (JAIR)* 61 (2018), pp. 623–698. DOI: 10.1613/jair.5565.

[15] J. Fransman, J. Sijs, H. Dol, E. Theunissen, and B. De Schutter. “Distributed Bayesian: a continuous distributed constraint optimization problem solver”. In: *Submitted to JAIR* (2021).

[16] E. C. Freuder and M. J. Quinn. “Taking advantage of stable sets of variables in constraint satisfaction problems”. In: *International Joint Conference on Artificial Intelligence (IJCAI)*. 1985, pp. 1076–1078. URL: https://www.ijcai.org/Proceedings/85-2/Papers/082.pdf.

[17] R. G. Gallager, P. a. Humblet, and P. M. Spira. “A distributed algorithm for minimum-weight spanning trees”. In: *Transactions on Programming Languages and Systems (TOPLAS)* 5.1 (1983), pp. 66–77. DOI: 10.1145/357195.357200.

[18] S. Geman and D. Geman. “Stochastic relaxation, Gibbs distributions, and the Bayesian restoration of images”. In: *Transactions on Pattern Analysis and Machine Intelligence* 6 (1984), pp. 721–741. DOI: 10.1109/TPAMI.1984.4767596.

[19] A. Gershman, A. Meisels, and R. Zivan. “Asynchronous forward bounding for distributed COPs”. In: *Journal of Artificial Intelligence Research (JAIR)* 34 (2009), pp. 61–88. DOI: 10.1613/jair.2591.

[20] A. Hagberg, D. Schult, and P. Swart. *NetworkX*. 2020. URL: https://networkx.github.io/.

[21] Y. Hamadi and J. Quinqueton. “Backtracking in distributed constraint networks”. In: *European Association for Artificial Intelligence (ECAI)*. 1998, pp. 219–223. URL: https://jmvidal.cse.sc.edu/library/hamadi98a.pdf.

[22] K. D. Hoang, W. Yeoh, M. Yokoo, and Z. Rabinovich. “New algorithms for continuous distributed constraint optimization problems”. In: *International Conference on Autonomous Agents and Multiagent Systems (AAMAS)*. 2020, pp. 502–510. URL: https://dl.acm.org/doi/10.5555/3398761.3398823.

[23] J. Kennedy and R. Eberhart. “Particle swarm optimization”. In: *International Conference on Neural Networks (IJCNN)*. Vol. 4. 1995, pp. 1942–1948. DOI: 10.1007/978-3-642-37846-1_3.

[24] S. Kirkpatrick, C. D. Gelatt, and M. P. Vecchi. “Optimization by Simulated Annealing”. In: *Science* 220.4598 (1983), pp. 671–680. DOI: 10.1126/science.220.4598.671.

[25] L. Kocsis and C. Szepesvári. “Bandit based Monte-Carlo planning”. In: *European Conference on Machine Learning (ECML)*. 2006, pp. 282–293. DOI: 10.1007/11871842_29.

[26] H. J. Kushner. “A new method for locating the maximum point of an arbitrary multipeak curve in the presence of noise”. In: *Journal of Basic Engineering* 86.1 (1964), pp. 97–106. DOI: 10.1115/1.3653121.

[27] A. R. Leite, F. Enembreck, and J.-P. A. Barthès. "Distributed constraint optimization problems: review and perspectives". In: *Expert Systems with Applications* 41.11 (2014), pp. 5139–5157. DOI: 10.1016/j.eswa.2014.02.039.

[28] D. J. Lizotte, R. Greiner, and D. Schuurmans. "An experimental methodology for response surface optimization methods". In: *Journal of Global Optimization* 53.4 (2012), pp. 699–736. DOI: 10.1007/s10898-011-9732-z.

[29] D. J. C. MacKay. "Bayesian interpolation". In: *Neural Computation* 4.3 (1992), pp. 415–447. DOI: 10.1162/neco.1992.4.3.415.

[30] D. J. MacKay. "Bayesian nonlinear modeling for the prediction competition". In: *American Society of Heating, Refrigerating and Air-Conditioning Engineers Transactions* 100.2 (1994), pp. 1053–1062. DOI: 10.1007/978-94-015-8729-7_18.

[31] R. Maheswaran, M. Tambe, E. Bowring, J. Pearce, and P. Varakantham. "Taking DCOP to the real world: efficient complete solutions for distributed multi-event scheduling". In: *International Conference on Autonomous Agents and Multiagent Systems (AAMAS)*. 2004. DOI: 10.1109/AAMAS.2004.10067.

[32] A. Meisels, E. Kaplansky, I. Razgon, and R. Zivan. "Comparing performance of distributed constraints processing algorithms". In: *AAMAS workshop on Distributed Constraint Reasoning (DCR)*. 2002, pp. 86–93. URL: https://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.57.8606.

[33] A. Meisels. *Distributed search by constrained agents: algorithms, performance, communication*. Springer Science & Business Media, 2007. DOI: 10.1007/978-3-642-24013-3_2.

[34] C. A. Micchelli, Y. Xu, and H. Zhang. "Universal kernels". In: *Journal of Machine Learning Research (JMLR)* 7 (2006), pp. 2651–2667. URL: https://www.jmlr.org/papers/volume7/micchelli06a/micchelli06a.pdf.

[35] B. Minasny and A. B. McBratney. "The Matérn function as a general model for soil variograms". In: *Geoderma* 128 (2005), pp. 192–207. DOI: 10.1016/j.geoderma.2005.04.003.

[36] S. K. Mishra. *Some new test functions for global optimization and performance of repulsive particle swarm method*. Tech. rep. 2006. DOI: 10.2139/ssrn.926132.

[37] J. Mockus. *Bayesian approach to global optimization: theory and applications*. Kluwer Academic Publishers, 1989. DOI: 10.1007/978-94-009-0909-0.

[38] J. Mockus. "The Bayesian approach to global optimization". In: *System Modeling and Optimization* 38 (1982), pp. 473–481. DOI: 10.1007/BFb0006170.

[39] J. Mockus, V. Tiesis, and A. Zilinskas. "The application of Bayesian methods for seeking the extremum". In: *Towards Global Optimisation* 2 (1978), pp. 117–129. URL: https://www.researchgate.net/publication/248818761.

[40] P. J. Modi, W. M. Shen, M. Tambe, and M. Yokoo. "Adopt: Asynchronous distributed constraint optimization with quality guarantees". In: *Artificial Intelligence* 161.1-2 (2005), pp. 149–180. DOI: 10.1016/j.artint.2004.09.003.

[41] D. T. Nguyen, W. Yeoh, H. C. Lau, and R. Zivan. “Distributed Gibbs: a linear-space sampling-based DCOP algorithm”. In: *Journal of Artificial Intelligence Research (JAIR)* 64 (2019), pp. 705–748. DOI: 10.1613/jair.1.11400.

[42] B. Ottens, C. Dimitrakakis, and B. Faltings. “DUCT: An Upper Confidence Bound Approach to Distributed Constraint Optimization Problems”. In: *ACM Transactions on Intelligent Systems and Technology* 8.5 (2017). DOI: 10.1145/3066156.

[43] A. Petcu and B. Faltings. “DPOP: a scalable method for multiagent constraint optimization”. In: *International Joint Conference on Artificial Intelligence (IJCAI)*. 2005, pp. 266–271. URL: https://www.ijcai.org/Proceedings/05/Papers/0445.pdf.

[44] C. E. Rasmussen and C. K. I. Williams. *Gaussian processes for machine learning*. MIT Press, 2006. ISBN: 026218253X. URL: http://www.gaussianprocess.org/gpml/chapters/RW.pdf.

[45] A. Rogers, A. Farinelli, R. Stranders, and N. R. Jennings. “Bounded approximate decentralised coordination via the max-sum algorithm”. In: *Artificial Intelligence* 175.2 (2011), pp. 730–759. DOI: 10.1016/j.artint.2010.11.001.

[46] P. Rust, G. Picard, and F. Ramparany. “pyDCOP, a DCOP library for IoT and dynamic systems”. In: *International workshop on Optimisation in Multi-Agent Systems (OptMAS)*. Montréal, Canada, 2019. URL: https://hal.archives-ouvertes.fr/hal-02098294.

[47] A. Sarker, A. B. Arif, M. Choudhury, and M. M. Khan. “C-CoCoA: a continuous cooperative constraint approximation algorithm to solve functional DCOPs”. In: *arXiv* (2020). arXiv: 2002.12427.

[48] D. M. Sato, A. P. Borges, P. Márton, and E. E. Scalabrin. “I-DCOP: train classification based on an iterative process using distributed constraint optimization”. In: *Procedia Computer Science* 51 (2015), pp. 2297–2306. DOI: 10.1016/j.procs.2015.05.391.

[49] J. Sherman and W. J. Morrison. “Adjustment of an inverse matrix corresponding to a change in one element of a given matrix”. In: *The Annals of Mathematical Statistics* 1 (1950), pp. 124–127. DOI: 10.1214/aoms/1177729893.

[50] R. Stranders, A. Farinelli, A. Rogers, and N. R. Jennings. “Decentralised coordination of continuously valued control parameters using the max-sum algorithm”. In: *International Conference on Autonomous Agents and Multiagent Systems (AAMAS)*. 2009, pp. 601–608. URL: https://dl.acm.org/doi/10.5555/1558013.1558097.

[51] E. A. Sultanik, P. J. Modi, and W. W. C. Regli. “On modeling multiagent task scheduling as a distributed constraint optimization problem”. In: *International Joint Conference on Artificial Intelligence (IJCAI)*. 2007, pp. 247–253. URL: https://www.ijcai.org/Proceedings/07/Papers/247.pdf.

[52] A. Törn and A. Žilinskas. *Global optimization*. Vol. 350. Springer, 1989. DOI: 10.1007/3-540-50871-6.

[53] E. Tsang. *Foundations of Constraint Satisfaction.* London: Academic Press, 1993. DOI: 10.1016/C2013-0-07627-X.

[54] C. J. Van Leeuwen. "CoCoA: a non-iterative approach to a local search (A)DCOP solver". In: *Association for the Advancement of Artificial Intelligence.* 2017, pp. 3944–3950. URL: https://ojs.aaai.org/index.php/AAAI/article/view/11125/.

[55] H. van Hasselt. "Reinforcement learning in continuous state and action spaces". In: *Adaptation, Learning, and Optimization.* Vol. 12. 2012, pp. 207–251. DOI: 10.1007/978-3-642-27645-3_7.

[56] E. Vazquez and J. Bect. "Convergence properties of the expected improvement algorithm with fixed mean and covariance functions". In: *Journal of Statistical Planning and Inference (SPI)* 140.11 (2010), pp. 3088–3095. DOI: 10.1016/j.jspi.2010.04.018.

[57] J.-P. Vert, K. Tsuda, and B. Schölkopf. "A primer on kernel methods". In: *Kernel Methods in Computational Biology* 47 (2004), pp. 35–70. DOI: 10.7551/mitpress/4057.003.0004.

[58] L. G. R. Vianna, S. Sanner, and L. N. de Barros. "Continuous real time dynamic programming for discrete and continuous state MDPs". In: *Brazilian Conference on Intelligent Systems (BRACIS).* Vol. 3. 1. IEEE, 2014, pp. 134–139. DOI: 10.1109/BRACIS.2014.34.

[59] M. Vinyals, J. A. Rodríguez-Aguilar, and J. Cerquides. "Generalizing DPOP: Action-GDL, a new complete algorithm for DCOPs". In: *International Conference on Autonomous Agents and Multiagent Systems (AAMAS).* Vol. 1. 2009, pp. 1239–1240. URL: https://www.researchgate.net/publication/221456786.

[60] T. Voice, R. Stranders, A. Rogers, and N. R. Jennings. "A hybrid continuous max-sum algorithm for decentralised coordination". In: *Frontiers in Artificial Intelligence and Applications* 215 (2010), pp. 61–66. DOI: 10.3233/978-1-60750-606-5-61.

[61] L. Wittenburg and W. Zhang. "Distributed breakout algorithm for distributed constraint optimization problems – DBArelax". In: *International Conference on Autonomous Agents and Multiagent Systems (AAMAS).* 2003, p. 1158. URL: https://dl.acm.org/doi/10.1145/860575.860844.

[62] W. Yeoh, A. Feiner, and S. Koenig. "BnB-ADOPT: an asynchronous branch-and-bound DCOP algorithm". In: *Journal of Artificial Intelligence Research (JAIR)* 38 (2010), pp. 85–133. DOI: 10.1613/jair.2849.

[63] W. Yeoh and M. Yokoo. "Distributed problem solving". In: *AI Magazine* 33 (2012), pp. 53–65. DOI: 10.1609/aimag.v33i3.2429.

[64] M. Yokoo, E. H. Durfee, T. Ishida, and K. Kuwabara. "The distributed constraint satisfaction problem: formalization and algorithms". In: *Transactions on Knowledge and Data Engineering* 10.5 (1998), pp. 673–685. DOI: 10.1109/69.729707.

[65] R. Zivan, T. Parash, and Y. Naveh. “Applying max-sum to asymmetric distributed constraint optimization”. In: *International Joint Conference on Artificial Intelligence (IJCAI)*. 2015, pp. 432–439. URL: https://dl.acm.org/doi/abs/10.5555/2832249.2832309.

5

The Distributed Bayesian algorithm: simulation and experimental results for a cooperative multi UAV search use case

In this chapter, the Distributed Bayesian (D-Bay) algorithm is applied to an autonomous search use case. Within the use case multiple unmanned aerial vehicles equipped with cameras cooperatively search an area and minimize the required time. The use case is modeled within the continuous Distributed Constraint Optimization Problem (DCOP) framework. This framework extends the (discrete) DCOP framework by allowing variables with continuous domains. Compared to similar DCOP solvers, the characteristics of the D-Bay algorithm are well-suited for the use case and allow for the implementation of autonomous vehicles with limited resources (computational power, memory, and communication bandwidth). Experimental results are given and these results are used to validate a simulation environment. Within the simulation environment, various scenarios are implemented. The D-Bay algorithm was able to find solutions within 3.5 % *of the optimal solution with a limited number of samples per agent.*

This chapter has been published in the proceedings of the OptLearnMAS workshop of the AAMAS conference, 2020 [10].

5.1. Introduction

Autonomous vehicles are used for various tasks to increase situational awareness. Examples include surveillance, search, patrolling, observing, and pursuit-evasion. These tasks are often represented as a mobile sensor coordination problem [33], a cooperative search problem [3], or a patrolling problem [27].

In the literature, numerous types of autonomous vehicles have been used to perform cooperative searches. The reader is referred to the work of Veres *et al.* [29] for a systematic overview of methodologies used for the main classes of autonomous vehicles. This chapter focuses on the cooperative search problem in an outdoor environment performed by Unmanned Aerial Vehicles (UAVs). For outdoor environments UAVs are generally used thanks to their ability to traverse a large number of types of terrain. In typical real-world use cases, several UAVs need to search a predefined region in as little time as possible to find particular objects or victims. In these use cases, a single operator should be able to control all UAVs in an easy and straightforward manner. Therefore, a UAV should be able to optimize its own path with respect to the paths of the other UAVs. For this reason, search problems have been modeled within various frameworks, such as the task allocation framework, Markov decision processes, and game theory. The reader is referred to the work of Robin *et al.* [25] for a unifying taxonomy of search-related problems and a comprehensive survey of the application of a wide range of problem frameworks. The usage of UAVs introduces an implementation problem for the algorithms used to optimize the individual paths as the complications arise from limitations in communication, computation, and/or memory. For UAVs, these complications become apparent as the required hardware needs to be small and lightweight to reduce negative effects on endurance. Additionally, due to the distance between the UAVs during real-world applications, the communication capabilities to a central system are often limited. In other words, UAVs can communicate with nearby UAVs but not with a central system. This makes it impractical to use a centralized approach.

In order to take both considerations into account, in the current chapter, the Distributed Constraint Optimization Problem (DCOP) framework is used to model the distributed cooperative search problem. As noted by Fioretto *et al.* [8], this framework is well suited since it enables the exploitation of the structure of the problem to create both efficient algorithms and communication strategies. The DCOP framework has been applied to a wide range of problems. Some notable examples include the works of Meisels [17], Modi *et al.* [18], Petcu *et al.* [21], Gershman *et al.* [12], and Yeoh *et al.* [31]. A problem modeled as a DCOP is based on an objective function that quantifies the collaborative goal of agents. Thanks to the focus on the distributed nature of the underlying problems, the solvers for DCOP are focused on local computation and explicit communication strategies. This makes the DCOP framework ideal for modeling and solving the distributed cooperative search task.

The contributions of this chapter are twofold. Firstly, the cooperative search problem is modeled as a continuous DCOP and optimized by the D-Bay algorithm [9]. It will be shown that the characteristics of the D-Bay algorithm make it suitable for the application to UAVs with limited resources. Secondly, experimental results are given for a cooperative search use case with multiple UAVs to illustrate the application of the D-Bay algorithm in a real-world scenario. In the use case, a predefined area needs to be searched in as little time as possible. Additional simulation results are given in order to evaluate the performance of the D-Bay algorithm for a heterogeneous group of UAVs.

The remainder of this chapter is organized as follows. In Section 5.2 the cooperative search problem is described and modeled within the DCOP framework. Section 5.3 introduces the D-Bay algorithm and evaluates its characteristics. In Section 5.4 the overview of the hardware and the software of the UAVs used in the experiments are presented. Afterward, experimental evaluation of the D-Bay algorithm with multiple UAVs is included in Section 5.5. Finally, Section 5.6 summarizes the results and defines future work.

5.2. PROBLEM DEFINITION

We consider a cooperative search use case with multiple autonomous quad-rotor vehicles. These vehicles are selected thanks to their versatility and high maneuverability. As autonomous vehicles will distributively optimize their trajectories, they are referred to as (autonomous) agents. Typically, these search operations are performed by autonomous vehicles equipped with one or more cameras. While surveying, the images from the cameras are evaluated by image-processing software to detect and identify objects of interest. The solution to the cooperative search problem involves an optimal division of the search region between all agents. The optimal division is defined as the division that will require the least amount of time for all vehicles to complete the entire search and return to their initial position.

More specifically, we consider a (rectangular) search region defined by its width and height as $R = (R_\mathrm{w}, R_\mathrm{h})$, where $R_\mathrm{w} \geq R_\mathrm{h}$. It is assumed that there are no obstacles and that the size of the region is known by all agents. Every agent has a single variable x_i it can assign, where i is the agent index. The variables of the agents are related to the size of their individual segments $R_i = (R_{i,\mathrm{w}}, R_\mathrm{h})$ and $R_\mathrm{w} = \sum_{i=1}^{N} R_{i,\mathrm{w}}$.

The search problem is considered to be two-dimensional, as the altitude for all UAVs is kept constant during the search. The altitude results from a trade-off between the required resolution for successful detection of the images and the size of the area imaged by the camera. Higher-resolution images will therefore enable a larger observed area.

In addition to the time spent searching, the travel times of the UAVs towards the start of their segments ($p_{i,\mathrm{s}}$), and from the end of the segment ($p_{i,\mathrm{f}}$) back to their initial positions (p_i) are taken into account. The UAVs will traverse a series of equidistance parallel legs within their search segments, where a *leg* indicates a straight line over which scanning is performed. This search pattern is often referred to as a lawnmower pattern. As shown by Ablavsky *et al.* [2], the lawnmower pattern is optimal for a rectangular search area and UAVs of which the sweep width is larger than the turn radius. For quad-rotor vehicles, this holds for any sweep width as these platforms are holonomic. The sweep width is based on the width of the camera image ($l_{i,\mathrm{w}}$) on the ground. To ensure complete coverage, the distance between the legs is based on the sweep width such that consecutive tracks interleave. An additional distance (l_t) is added before the start of the leg to ensure that oscillations (caused by the cornering) are eliminated to ensure that the image quality is constant while searching. The velocity of the UAVs during scanning and transit is denoted by v_s, and v_t, respectively.

An overview of the search segment of a single agent can be seen in Figure 5.1.

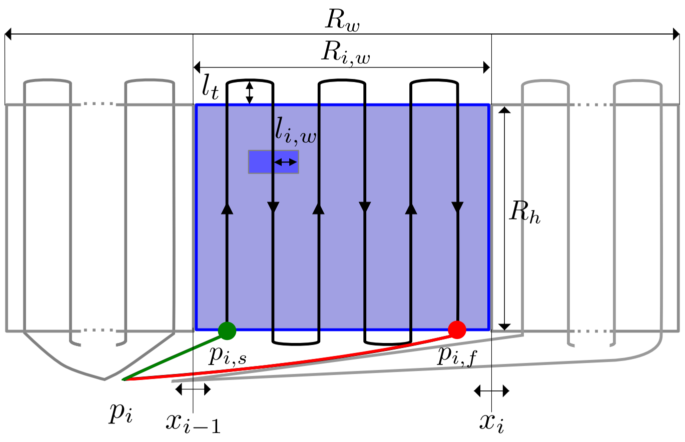


Figure 5.1.: Overview of the search segment of agent a_i.

For agent a_i the number of legs of its lawnmower pattern is defined by

$$N_{i,\mathrm{l}}(x_{i-1}, x_i) = \left\lceil \frac{R_{i,\mathrm{w}}}{2l_{i,\mathrm{w}}} \right\rceil = \left\lceil \frac{x_i - x_{i-1}}{2l_{i,\mathrm{w}}} \right\rceil,$$

where $\lceil\cdot\rceil$ is the ceiling operator. The start and finish positions of the pattern of agent a_i can be defined according to its neighboring agent a_{i-1} as

$$p_{i,\mathrm{s}}(x_{i-1}) = (x_{i-1} + l_{i,w}, 0),$$
$$p_{i,\mathrm{f}}(x_i) = \begin{cases} (x_i - l_{i,w}, 0) & \text{if } N_{i,\mathrm{l}} \text{ is even,} \\ (x_i - l_{i,w}, R_\mathrm{h}) & \text{otherwise.} \end{cases}$$

Based on these values and the properties of the search area, the required time to scan a segment (including transit) for a_i is calculated as

$$f_i(x_{i-1}, x_i) = T_{i,\mathrm{s}}(x_{i-1}) + T_{i,\mathrm{sc}}(x_{i-1}, x_i) + T_{i,\mathrm{f}}(x_i), \tag{5.1}$$

where the transit times ($T_{i,\mathrm{s}}(\cdot)$ and $T_{i,\mathrm{f}}(\cdot)$) and the time spent scanning ($T_{i,\mathrm{sc}}(\cdot)$) are defined as

$$\begin{aligned}T_{i,\mathrm{s}}(x_{i-1}) &= \frac{|p_i - p_{i,\mathrm{s}}(x_{i-1})|^2}{v_\mathrm{s}},\\ T_{i,\mathrm{f}}(x_i) &= \frac{|p_i - p_{i,\mathrm{f}}(x_i)|^2}{v_\mathrm{t}},\\ T_{i,\mathrm{sc}}(x_{i-1}, x_i) &= T_{i,\mathrm{c}} + T_{i,\mathrm{l}}.\end{aligned}$$

The time spent scanning depends on both the time required for traversing the legs, defined as

$$T_{i,\mathrm{l}} = \frac{N_{i,\mathrm{l}}(x_{i-1}, x_i)R_\mathrm{h}}{v_\mathrm{s}},$$

and on cornering between the legs, defined as

$$T_{i,\mathrm{c}} = \frac{2(l_\mathrm{t} + l_{i,\mathrm{w}})\left(N_{i,\mathrm{l}}(x_{i-1}, x_i) - 1\right)}{v_\mathrm{s}} + 2N_{i,\mathrm{l}}(x_{i-1}, x_i)\tau,$$

where τ is the time required by the UAV to make a turn. In the next section, the search problem will be modeled within the DCOP framework.

5.2.1. PROBLEM FORMULATED AS A DCOP

A Distributed Constraint Optimization Problem (DCOP) is a problem framework that is based on a global objective function that needs to be optimized in a distributed manner. The global objective function is defined as the aggregate of utility functions. Typically, the summation operator ($\sum(\cdot)$) or the maximum operator ($\max(\cdot)$) are used as the aggregation operator. The agents within the DCOP are defined based on variables. Each agent can assign a value to its variables. Instead of the global objective function, an agent only knows two properties of the problem; (1) the local utility functions of which its variables are used as an argument, (2) its neighboring agents, which are defined as agents that *share* a utility function, i.e. if there exists a utility function of which its arguments include variables of both agents. This local view of the problem creates the need for the agents to cooperate to optimize the assignment of their variables in terms of the global objective function.

The value assignments are restricted by the domains of the variables. This feature of DCOP makes it suitable for problems with bounded inputs such as many real-world problems. However, conventionally the domains are defined as finite discrete sets, while many real-world problems (including cooperative search) are best described using finite continuous sets. The reason for the discrete set definition is based on the origin of the DCOP framework. DCOP originates from the Constraint Satisfaction Problem (CSP) framework [28], which is mainly used to model problems such as graph coloring problems and meeting scheduling problems. These problems inherently have finite and discrete domains. The DCOP framework emerged from extending the CSP framework to agent-based distributed optimization by Yokoo *et al.* [32] and generalization to utility functions instead of constraint satisfaction checking. Within this process of extending and distributing, the domains have not been updated to include continuous values.

To address this issue, in this chapter, a continuous version of DCOP is used to model the cooperative search problem. Following the notation of Fioretto *et al.* [8], a continuous DCOP is defined by $\mathfrak{D} = \langle \mathbf{A}, \mathbf{X}, \mathbf{D}, \mathbf{F}, \alpha, \eta \rangle$ where:

- $\mathbf{A} = \{a_1, \ldots, a_M\}$ is the set of agents, where M is the number of agents.
- $\mathbf{X} = \{x_1, \ldots, x_N\}$ is the set of variables, where $N \geq M$ is the number of variables.
- $\mathbf{D} = \{\mathbf{D}_1, \ldots, \mathbf{D}_N\}$ is the set of domains of all variables, where $\mathbf{D}_i \subseteq \mathbb{R}$ is the (continuous) domain associated with variable x_i.

  An assignment denotes the projection of variables onto their domain as $\rho : \mathbf{X} \rightarrow \Sigma$. In other words, for all $x_i \in \mathbf{X}$ if $\rho(x_i)$ is defined, then $\rho(x_i) \in \mathbf{D}_i$. An assignment of a subset of variables is denoted by $\rho_{\mathbf{V}} = \{\rho(x_i) : x_i \in \mathbf{V}\}$.
- $\mathbf{F} = \{f_1, \ldots, f_K\}$ is the set of utility functions, where K is the number of utility functions.
- $\alpha : \mathbf{X} \rightarrow \mathbf{A}$ is a mapping from variables to agents. The agent to which variable x_i is allocated is denoted as $\alpha(x_i)$.
- η is an operator that combines all utility functions into the objective function.

The global objective of a DCOP is to minimize the objective function, defined by $G(\rho) = \underset{f_n \in \mathbf{F}}{\eta} \left(f_n(\rho_{\mathbf{V}_n}) \right)$. Based on this definition the cooperative search problem can be cast into a DCOP as,

- $\mathbf{A} = \{a_1, \ldots, a_N\}$, where N is the number of UAVs,
- $\mathbf{X} = \{x_1, \ldots, x_N\}$, since every agent has a single variable ($M = N$),
- $\mathbf{D} = \{\mathbf{D}_1, \ldots, \mathbf{D}_N\}$, where $\mathbf{D}_i = [0, R_\text{w}]$ is related to the search region,
- $\mathbf{F} = \{f_i(x_{i-1}, x_i) : i \in \{1, \ldots, N\}\}$, where $f_i(x_{i-1}, x_i)$ is defined by Equation 5.1 with $x_0 = 0$,
- $\alpha = \{x_i \rightarrow a_i : i \in \{1, \ldots, N\}\}$, where every agent is assigned a single variable,
- $\eta = \max(\cdot)$, in order to minimize the maximum time required for all agents.

Apart from the definition of the continuous DCOP as given above, a DCOP is typically represented in the form of a graph. Two frequently used forms are the (undirected) constraint graph and the (directed) pseudo-tree [11]. In both graphs, the agents are shown as nodes and neighboring agents are connected through an edge. Note that a constraint graph can be converted into a pseudo-tree by means of various procedures, such as depth-first-search [4]. A benefit of the pseudo-tree over the constraint graph is that the pseudo-tree introduces hierarchy to the variables and thereby divides the problem into subproblems. The hierarchy creates an implicit communication structure that is used to send messages between the agents without requiring all-to-all communication. The subproblems can be exploited by algorithms in order to efficiently solve the DCOP.

5.3. ALGORITHM OVERVIEW

Due to the limited (computational and communication) resources that are available to the UAVs, not all DCOP solvers will be suitable to solve the (continuous) DCOP. Classical DCOPs solvers tend to be intractable for continuous DCOPs, despite a limited number of variables, due to the cardinality of the domains. As noted by Leite *et al.* [15] and Fioretto *et al.* [8], the runtime complexity increases exponentially for almost all solvers that are guaranteed to find the optimal solution. Even for near-optimal solvers, the complexity increase is linear based on the cardinality of the largest domain.

The main reason is that the majority of the DCOP solvers are created for DCOPs with discrete domains. Therefore, to apply these solvers, the continuous domains need to be discretized. A straightforward approach is to use equidistant discretization for all variables. This allows for the creation of domains with a cardinality of arbitrary size. As for most problems, the quality of the solution depends on the resolution of the variables, where a high resolution allows for better solutions. This creates a trade-off between solution quality and computational and memory requirements.

5.3.1. DESCRIPTION OF D-BAY

In a previous chapter, the Distributed Bayesian (D-Bay) algorithm [9] was introduced to solve a continuous DCOP without the need for the discretization of the continuous domains. The D-Bay algorithm involves four sequential phases:

(1) Pseudo-tree construction The agents create a pseudo-tree from the constraint graph of the DCOP, by performing a depth-first search traversal.

(2) Allocation of utility functions Similar to the allocation of variables, all utility functions are exclusively allocated to the agents.

(3) Sample propagation In this phase, every agent optimizes its local variables through the Bayesian optimization method and the exchange of **sample** and **utility** messages. This phase is initiated by a **sample** message from the root agent. The phase finishes when a termination criterion is reached by the root agent. The samples are selected through the optimization of an acquisition function. A graphical overview of the sample phase is shown in Figure 5.2.

(4) Assignment propagation The final phase is the assignment propagation phase, in which the root agent sends the final assignment of all its variables to its children as a **final** message. Based on these assignments the children can assign their own variables to the value corresponding to the optimal utility value.

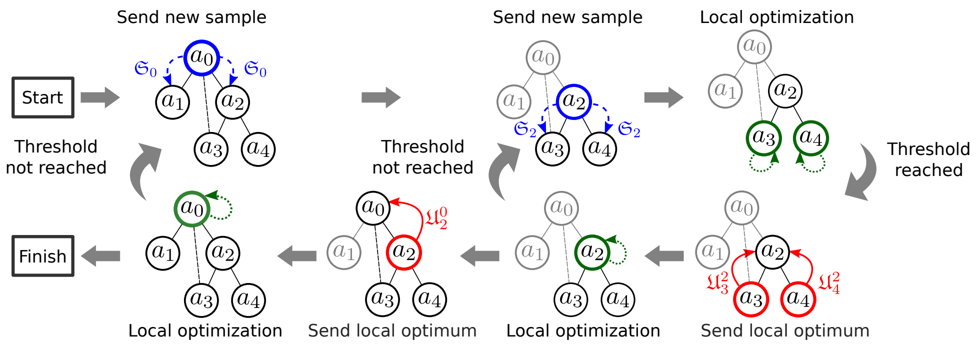


Figure 5.2.: Graphical overview of the sample phase of D-Bay. Agents are indicated by circles labeled with an agent index, and utility functions are shown as black lines. Starting from the root a_0 (top-left), a **sample** message $\mathfrak{S}_0$ is sent to its children (a_1, a_2). After iterating between its children and calculating its local utility, agent a_2 combines all local utilities and sends a **utility** message $\mathfrak{U}_2^0$ to its parent.

Based on the taxonomy introduced by Yeoh *et al.* [30], the D-Bay algorithm can be classified as a sample-based solver. This class of solvers uses probabilistic measures to coordinate the sampling of the global search space. The probabilistic measures are used to balance exploration and exploitation of the search space. The two main differences with existing sample-based solvers (DUCT [20] and Sequential Distributed Gibbs (SD-Gibbs) [19]) are the sample selection and its iterative approach. Firstly, within the D-Bay algorithm, the Bayesian optimization method is used for the selection of the samples. The Bayesian optimization method consists of two elements: a *probabilistic model* to approximate an unknown (utility) function, and an *acquisition function* to optimally select a new sample. This method is also referred to as an active learning approach as the acquisition function makes use of previously sampled values and the probabilistic model to *learn* as much about the function in a sample-efficient manner. The Gaussian process is used as the probabilistic model to represent acquired knowledge about unknown functions. This model is widely used to efficiently approximate functions through a predefined cross-correlation function, commonly referred to as a kernel. The values of the function are approximated based on the inverse of the covariance matrix, which is generated based on the kernel and samples of the function. As noted by Rasmussen *et al.* [23], an important part of the computational burden of Bayesian optimization is the Cholesky factorization used to calculate the inverse of the covariance matrix. To significantly reduce the computational load, a Markovian class kernel [6] is used within the D-Bay algorithm. A Markovian class kernel possesses the property that the corresponding covariance matrices can be inverted analytically. Secondly, instead of iterating over the entire pseudo-tree, agents iterate between parents and children only. While this typically requires more messages than DUCT and Sequential Distributed Gibbs (SD-Gibbs), it ensures determinism concerning the utility value of a sample. In other words, a sample will always return the same utility value.

5.3.2. Characteristics of D-Bay

In this section, the characteristics of the D-Bay algorithm are compared to the related DCOP solvers, DUCT and Sequential Distributed Gibbs (SD-Gibbs).

Similar to DUCT and Sequential Distributed Gibbs (SD-Gibbs), D-Bay is an anytime algorithm since the optimization can be stopped after every iteration and the solution will always improve compared to the previous iteration. Most other characteristics are related to the number of iterations during the optimization, denoted by I. Within the D-Bay algorithm, the number of iterations is used as a termination criterion for all agents. In every iteration, an agent optimizes the value of its next sample through Bayesian optimization. Therefore the runtime complexity of D-Bay is $O(I)$. The optimization of the local variables is restarted every time a new sample message from the parent is received. An agent only needs to store the utility of the values based on the current (local) iteration to send the *best* utility value back to its parent, thereby restricting the memory requirement per agent to $O(I)$. The size of the utility messages is fixed at $O(1)$ while the size of the sample messages is proportional to the maximal depth of the tree t as $O(t)$. The number of messages scales with $O(CI^t)$, where C denotes the largest number of children. Note that this will result in $O(I^N)$ in the worst-case scenario. In this scenario, the problem structure cannot be exploited to split the problem into subproblems. In other words, the resulting pseudo-tree has a single branch (also known as a pseudo-chain).

The related DCOP solvers (DUCT and Sequential Distributed Gibbs (SD-Gibbs)) need to discretize the domains and have a runtime complexity corresponding to the cardinality of the largest domain as $O(I\eta d)$. Here d denotes the largest cardinality of the domains and η denotes the largest number of neighboring agents. This highlights the fact that by discretizing the continuous domains, the runtime complexity will significantly increase. Furthermore, for DUCT there are large memory requirements $O(d^t)$ as it needs to store all the best costs for all messages. SD-Gibbs is much more memory efficient and merely requires $O(\eta)$. Concerning the message characteristics, DUCT sends fewer messages than SD-Gibbs, but they are larger. A comparison of the D-Bay algorithm with its closest related solvers for these key characteristics is shown in Table 5.1.

Table 5.1.: Comparison of algorithm characteristics (based on Fioretto *et al.* [8, Table 4]).

	Runtime		Message	
Algorithm	Complexity	Memory	Number	Size
DUCT	$O(I\eta d)$	$O(d^t)$	$O(IN)$	$O(N)$
D-Gibbs	$O(I\eta d)$	$O(\eta)$	$O(IN\eta)$	$O(1)$
D-Bay	$O(I)$	$O(I)$	$O(CI^t)$	$O(t)$

The characteristics of the D-Bay algorithm are favorable for agents with limited resources as it only requires the communication of small messages without large computational or memory requirements per message. Furthermore, the requirements are independent of the discretization of the domains, which solves an important problem of current DCOP solvers.

Note that these requirements are related to the iterations per agent, not the entire pseudo-tree as in DUCT and SD-Gibbs. This difference is the result of the definition of an *iteration* within the execution of the algorithm. After every iteration, the algorithm could be stopped and return a solution to the problem. For both DUCT and SD-Gibbs, this requires all agents to send, receive, and process messages. In D-Bay every agent optimizes its value based on the latest sample of their parents. For that reason, an update of the values of the leaf agents is regarded as an iteration within D-Bay. Due to this difference, non-concurrent constraint checks [16] are generally used to compare DCOP solvers for specific problems in a quantitative manner.

Although the size of the messages is small, the number of messages is largely dependent on the depth of the tree. Therefore, during the experiments, two main aspects of the D-Bay algorithm will be investigated. Firstly, the quality of the solutions for the number of samples determines the number of messages sent during optimization. Secondly, verification of the number of messages required by the algorithm allows for application on a low-bandwidth network.

5.4. Unmanned Aerial Vehicles (UAVs)

In this section, an overview of the Unmanned Aerial Vehicles (UAVs) used for the experiments is given. The hardware, software architecture, and simulation environment are discussed.

5.4.1. Hardware overview

In the experiments, quad-rotor UAVs (3DR-X4) are equipped with a downward-facing camera (GoPro Hero5 Session [13]) that image a small section of the search area while traveling over it while taking snapshots. This camera was selected as it is low-weight and eliminates the vibrations induced by the UAV through software-based image stabilization. The camera is connected to an onboard computer (Raspberry Pi 3B+ [24]), that runs the D-Bay algorithm and handles the communication between the agents. The size, weight, and accessibility of this computer made it highly suitable for the application. Communication between the agents is done through low-power radios (XBee Pro [5]). This radio module can create a meshed network using the ZigBee protocol between all modules for up to 750 m. Note that this holds for an outdoor environment with a line-of-sight between the modules. Within the network messages up to 84 bytes can be sent. During the flight, the UAV is regulated by a flight controller (3DR Pixhawk 1 [1]), which uses measurements from an inertial measurement unit for linear acceleration and angular speed, barometric pressure measurements for height, and GPS measurements for latitude and longitude. This flight controller runs the PX4 autopilot software [7]. An overview of the UAV and its components is given in Figure 5.3.

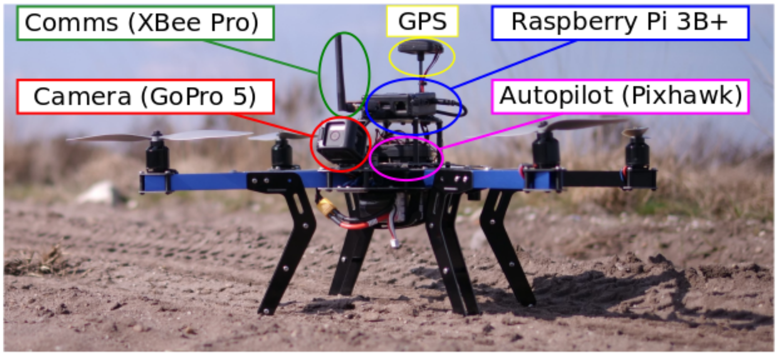
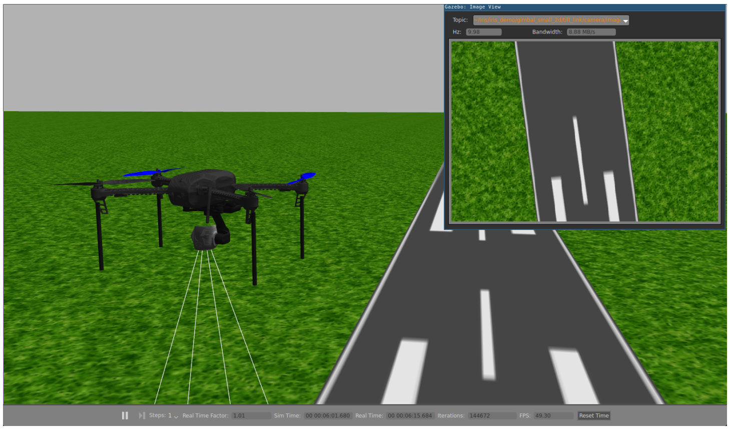


Figure 5.3.: Overview of the 3DR-X4 UAV used during the experiments. Key components required for autonomous flight are highlighted.

5.4.2. SIMULATION ENVIRONMENT

The configuration of the UAVs is modeled in a realistic simulation environment in order to validate the autonomous actions of the UAVs. In this chapter, the Gazebo simulator [14] is used as it offers a high level of flexibility in modeling the environment and the UAVs. There is a large open-source community that actively supports and extends the simulation environment, and creates high-fidelity virtual models for various (autonomous) vehicles. These models require a high level of expert knowledge to construct and validate. For this reason, within our simulations, the community model of the 3DR Iris is used. The 3DR Iris is the commercial version of the 3DR-X4 and is only different in appearance due to the enclosure. An example of the UAV model in the simulation environment is shown in Figure 5.4.



Topic: ~/iris/iris_demo/gimbal_small_2d/tilt_link/camera/image
Hz: 9.98 Bandwidth: 8.88 MB/s

Steps: 1 Real Time Factor: 1.01 Sim Time: 00 00:06:01.680 Real Time: 00 00:06:15.684 Iterations: 144672 FPS: 49.30

Figure 5.4.: Example of the simulation environment (Gazebo) of the UAV (3DR Iris) and its simulated view of the downward facing camera. Adapted from https://www.youtube.com/watch?v=mKt4ZTaE2bk.

5.4.3. Software overview

An additional benefit of the Gazebo simulation environment is its extensive interface with Robotic Operating System (ROS) [22]. This allowed the seamless transition between simulation and experimental execution of the D-Bay algorithm and testing of the cooperative search use case. Apart from an interface to Gazebo and the autopilot, the ROS middleware is also used to connect all other software components within the computer, such as the communication module, the D-Bay algorithm[1], and the camera interface. A schematic overview of the software architecture is shown in Figure 5.5.

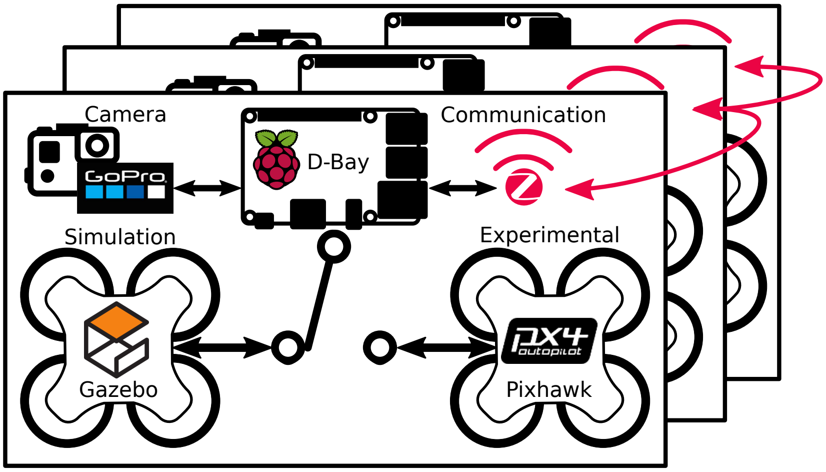


Figure 5.5.: Schematic overview of the software architecture.

5.5. Experimental evaluation

Preliminary experiments were performed to derive the parameters of the problem of the UAVs. The resolution of the cameras allowed for the UAVs to fly at a height of 10 m. At this altitude the scan width was 5 m, therefore $l_{i,\mathrm{w}} = l_\mathrm{w} = 5\,\mathrm{m}$. A velocity of 3 m/s was found to produce a stable flight without large oscillations in speed and position. Postprocessing of the camera images showed the velocity of the UAVs did not negatively influence the image quality. For this reason, both the velocity during scanning and transition are equally set to $v_\mathrm{t} = v_\mathrm{s} = 3\,\mathrm{m/s}$. Investigating the data collected during the cornering allowed for the determination of the turn length (l_t) and time required per turn (τ). While there were some minor differences in the number of oscillations after a turn, all were damped out after 5 m. Therefore, the turn length was set as $l_\mathrm{t} = 5\,\mathrm{m}$. The average time required per turn was found to be 1.5 s, however, when the UAV was greatly affected by wind, this could increase to as much as 7.5 s. Even though there were no strong winds, all experiments were located at an old airfield that offered no protection from the wind.

[1]available at `https://gitlab.com/jfransman/pyDcop/`

As these gusts of wind occurred only sporadically, the time required per turn was set to $\tau = 1.5\,\mathrm{s}$. Additionally, a search region of $R = (R_\mathrm{w}, R_\mathrm{h}) = (200\,\mathrm{m}, 50\,\mathrm{m})$ was defined to ensure all UAVs would have enough battery power to complete the entire search. The lower left corner of the search region is used as the reference position $(0,0)$. The UAVs were placed near the reference position with (approximately) 5 m intervals to ensure enough spacing between the UAVs for safe takeoff and landing conditions. For the experiments, a group of five UAVs was selected. An initial (reference) experiment was conducted as a baseline for the performance when no optimization of the segments was performed. The search area was equally divided among all UAVs as $R_{i,\mathrm{w}} = R_\mathrm{w}/5$. The required time (for all UAVs to finish their search) for the reference experiment to finish was approximately 240 s. The individual times required by the UAVs to finish scanning varied considerably. This was to be expected as the UAVs had a similar-sized area to scan, but greatly different travel times.

This reference experiment was compared to experiments with optimized segments by the D-Bay algorithm. At the start of the experiment, while all UAVs were located on the ground, the pseudo-tree was constructed through the DFS algorithm [4]. The *middle* UAV initiates the DFS algorithm since this will create a less *deep* tree compared to selecting a UAV that is assigned to the edge of the search area. After the DFS algorithm, the root agent of the pseudo-tree initiates the D-Bay algorithm by sending a sample to its children. When the D-Bay algorithm terminates, the resulting trajectories were sent to a control station. The control station was required to check the validity of the trajectories of the UAVs before takeoff[2]. After validation, the controls station issues a *cleared for takeoff* command to all UAVs. All actions afterward were autonomously executed by the UAVs.

For the experiments, the termination criterion (number of samples for the agents) was altered and the total required time for all UAVs to finish their search was evaluated. The comparison between the optimized segments and the reference experiment is shown in Table 5.2. The trajectories of the UAVs for 18 samples are shown in Figure 5.6.

Table 5.2.: Experimental results of optimized segments compared to reference segmentation.

		Total time difference	
# samples	Result [s]	Percentage [%]	Absolute [s]
6	221	−7.9	−19
12	218	−9.2	−22
18	208	−13.3	−32

[2]Manual validation of the search trajectories was a (hard) requirement for permission to fly multiple UAVs autonomously at the airport.

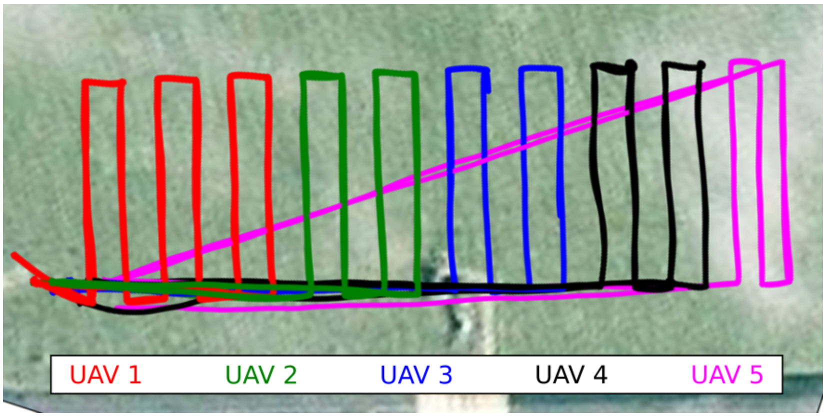


Figure 5.6.: Experimental results of the color-coded GPS tracks of the UAVs. The initial locations of the UAVs are in the bottom left corner.

5.5.1. Validation of simulations

An important motivation for carrying out the experiments was to validate the simulation environment and the (implicit) assumptions in the modeling of the cooperative search use case. The scenario of the experiments was modeled within the simulation environment and a comparison of the total required time based on a varying number of samples is shown in Table 5.3.

Table 5.3.: Comparison of experimental and simulated results.

	Result [s]		Total time difference	
# samples	Experimental	Simulated	Percentage [%]	Absolute [s]
6	221	220.8	−0.1	−0.2
12	218	209.7	−3.8	−8.2
18	208	206.0	−1.0	−2.0

The comparison shows a close match between the experimental and simulated results for 6 and 18 samples, which indicates a small residual modeling error. The comparison for 12 samples shows a relatively large difference. In that experiment, a single UAV had a longer flight time than its simulated counterpart. Inspection of the flight time of the UAV showed a deviation from the trajectory that can be attributed to a gust of wind during cornering. During preliminary experiments, the time required per turn could be seen to increase by as much as 6 seconds. This observation can explain the outlier in the comparison.

Apart from the influence of the wind, only minor deviations were identified during the comparison between the results within the simulation environment and the experiments.

This achievement can be attributed to the high quality of the UAV model and the integration of all important components during simulation (e.g. autopilot software, D-Bay algorithm). One of the major deviations between the initial simulations and the experiments can be attributed to the influence of the wind. These deviations are most clearly visible in the trajectory of UAV 4 in Figure 5.6 as this UAV deviated most from the intended trajectory. During cornering the UAVs were affected most as the waypoints of the trajectory are located closely together. A gust of wind can push an UAV off course, thus needing additional time to recover. Note that not all UAVs were affected to the same extent.

5.5.2. Evaluation of communication

As detailed in Section 5.3, during optimization the UAVs sent several small-sized messages between the parents and children. For this reason, it is appropriate to evaluate the communication between the agents. The data shows that the required time of optimization was severely affected by the latency within the XBee network. The latency is caused by the meshed topology of the network and the low-power design of the radios which disables the radio periodically to reduce energy consumption. The XBee radios create a meshed network that will relay messages from source to destination. During optimization (on the ground) all UAVs were within communication range, therefore instead of one-to-one communication, all messages were received by all radios. When a radio received a message that was not directed to it, the radio would not discard the message but would try to relay the message to its destination. This process is repeated by all radios (that are not the target of the message) and caused a message flood in which the same message was relayed by multiple radios. The flooding was limited by the maximum *hop count* of the messages but resulted in numerous messages being resent within the network. This effect increased the time for the messages to reach their final destinations significantly.

In addition to the optimization messages, the UAVs communicated their telemetry information through the XBee network during flight. We noticed that not all messages were successfully received, this is an indicator that the network was congested. During the optimization, messages were resent, however during the flight they were dropped after a few tries since the message buffers of the radios can only hold four messages. The congestion could be confirmed by increasing the interval after which the telemetry messages were sent. This increased the time for a single message to successfully reach the control computer. The update resulted in a large reduction in the number of dropped messages. Moreover, the relaying of messages was a clear indication that the range of the radios was far less than specified. Field tests showed an average range of 60 m during flight, while the datasheet specified up to the 750 m. Related to the size of the search area during the experiments, the 60 m range was sufficient for neighboring agents to pass messages. However, if similar experiments were to be conducted for a significantly larger search area the communication capabilities should allow for neighbors to be in direct contact with each other to ensure reliable execution of the algorithm.

Despite these issues, we can conclude that the XBee network was capable of sending the messages for the D-Bay algorithm. Thereby showing the D-Bay algorithm can be successfully implemented on a low-bandwidth communication network.

5.5.3. ADDITIONAL SIMULATIONS

Within the validated simulation environment additional simulations were performed. Five scenarios were constructed by changing the properties of some UAVs to evaluate the performance of the D-Bay algorithm for a group of heterogeneous UAVs. Scenarios 1-3 alter the scan width of a subset of UAVs to resemble UAVs with cameras of different quality. Scenario 4 varies the scanning velocity of three UAVs. In scenario 5 one UAV has a higher scanning velocity, but a smaller scan width. The (altered) parameters for the scenarios are defined as,

Scenario 1:	$l_{1,w} = 10\,\mathrm{m}$,		
Scenario 2:	$l_{1,w} = 10\,\mathrm{m}$,	$l_{5,w} = 20\,\mathrm{m}$,	
Scenario 3:	$l_{1,w} = 10\,\mathrm{m}$,	$l_{3,w} = 2.5\,\mathrm{m}$,	$l_{5,w} = 15\,\mathrm{m}$,
Scenario 4:	$v_{1,s} = 6\,\mathrm{m/s}$,	$v_{3,s} = 1.5\,\mathrm{m/s}$,	$v_{5,s} = 4.5\,\mathrm{m/s}$,
Scenario 5:	$v_{1,s} = 6\,\mathrm{m/s}$,	$l_{1,w} = 2.5\,\mathrm{m}$.	

In Table 5.4, an overview of the results of the simulations for 20 samples per agent is compared with the (optimal) results of a brute-force optimization method.

Table 5.4.: Simulation results compared to the optimum.

		Difference	
Scenario	Result [s]	Percentage [%]	Absolute [s]
1	179.0	3.2	5.5
2	160.0	0.4	0.7
3	179.9	2.4	4.3
4	181.8	3.5	6.2
5	188.5	2.8	5.2

In Figure 5.7 the simulation results for scenario 3 are shown.

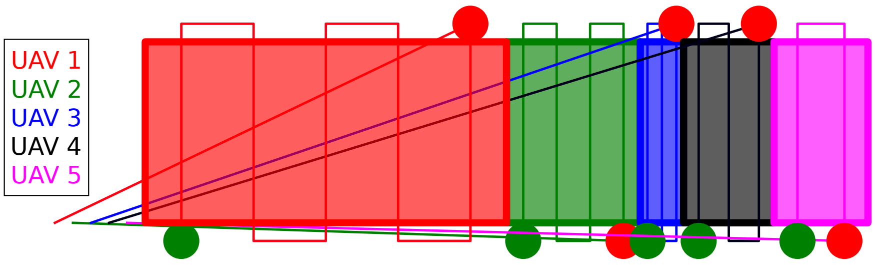


Figure 5.7.: Simulation results of the trajectories of the UAVs for scenario 3. The trajectories are color-coded by UAV.

The simulation results show a close approximation to the optimum for all scenarios. Similar results were obtained when only 10 samples per agent were used. The results for all scenarios (except scenario 1) did not differ more than 2 % from the results with 20 samples. This is an indication that the D-Bay algorithm converges relatively fast even with a very limited number of samples per agent.

5.6. Conclusions

In this chapter, experimental results for a cooperative search use case with multiple Unmanned Aerial Vehicles (UAVs) have been presented. This problem was modeled within the continuous Distributed Constraint Optimization Problem (DCOP) framework and solved with the Distributed Bayesian (D-Bay) algorithm. The D-Bay algorithm is specifically designed for continuous DCOPs and operates directly on the continuous domains. This makes the D-Bay algorithm excellently suitable for the application of real-world problems such as use cases with autonomous vehicles with limited resources (computational power, memory, and communication bandwidth).

Experimental results were used to validate the Gazebo high-fidelity simulation environment and to update the model of the cooperative search problem with UAVs. The simulation results showed that the D-Bay algorithm was able to find solutions for various scenarios that are within 3.5 % of the optimal solution with a limited number of samples per agent. As the algorithm and the hardware proved to be satisfactory for the cooperative search use case, future work will mainly focus on an extension to dynamic optimization based on real-time image collection. Finally, to make the D-Bay algorithm more accessible, we would like to contribute it to the PyDCOP [26] library.

5.7. Acknowledgment

The authors would like to thank Mathijs Lomme from TNO for his help during the experiments.

REFERENCES

[1] 3D Robotics. *Pixhawk 1 Flight Controller.* 2014. URL: https://pixhawk.org/.

[2] V. Ablavsky and M. Snorrason. "Optimal search for a moving target: a geometric approach". In: *AIAA Guidance, Navigation and Control Conference.* 2000. DOI: 10.2514/6.2000-4060.

[3] J. J. Acevedo, B. C. Arrue, I. Maza, and A. Ollero. "Cooperative large area surveillance with a team of aerial mobile robots for long endurance missions". In: *Journal of Intelligent and Robotic Systems: Theory and Applications* 70 (2013), pp. 329–345. DOI: 10.1007/s10846-012-9716-3.

[4] B. Awerbuch. "A new distributed depth-first-search algorithm". In: *Information Processing Letters* 20.3 (1985), pp. 147–150. DOI: 10.1016/0020-0190(85)90083-3.

[5] Digi International. *XBee Pro RF Module.* 2014. URL: https://www.sparkfun.com/datasheets/Wireless/Zigbee/XBee-Datasheet.pdf.

[6] L. Ding and X. Zhang. "Scalable stochastic kriging with Markovian covariances". In: *arXiv* (2018). arXiv: 1803.02575.

[7] Dronecode Project Inc. *PX4 autopilot software.* 2019. URL: https://px4.io/.

[8] F. Fioretto, E. Pontelli, and W. Yeoh. "Distributed constraint optimization problems and applications: a survey". In: *Journal of Artificial Intelligence Research (JAIR)* 61 (2018), pp. 623–698. DOI: 10.1613/jair.5565.

[9] J. Fransman, J. Sijs, H. Dol, E. Theunissen, and B. De Schutter. "Distributed Bayesian: a continuous distributed constraint optimization problem solver". In: *Submitted to JAIR* (2021).

[10] J. E. Fransman, J. Sijs, H. Dol, E. Theunissen, and B. De Schutter. "The Distributed Bayesian algorithm: simulation and experimental results for a cooperative multi UAV search use-case". In: *AAMAS workshop on Optimization and Learning in Multiagent Systems (OptLearnMAS).* 2020.

[11] E. C. Freuder and M. J. Quinn. "Taking advantage of stable sets of variables in constraint satisfaction problems". In: *International Joint Conference on Artificial Intelligence (IJCAI).* 1985, pp. 1076–1078. URL: https://www.ijcai.org/Proceedings/85-2/Papers/082.pdf.

[12] A. Gershman, A. Meisels, and R. Zivan. "Asynchronous forward bounding for distributed COPs". In: *Journal of Artificial Intelligence Research (JAIR)* 34 (2009), pp. 61–88. DOI: 10.1613/jair.2591.

[13] GoPro. *GoPro Session 5.* 2019. URL: https://gopro.com/en/us/yourhero5/session.

[14] N. Koenig and A. Howard. "Design and use paradigms for Gazebo, an open-source multi-robot simulator". In: *International Conference on Intelligent Robots and Systems (IROS).* 2004, pp. 2149–2154. DOI: 10.1109/IROS.2004.1389727.

[15] A. R. Leite, F. Enembreck, and J.-P. A. Barthès. "Distributed constraint optimization problems: review and perspectives". In: *Expert Systems with Applications* 41.11 (2014), pp. 5139–5157. DOI: 10.1016/j.eswa.2014.02.039.

[16] A. Meisels, E. Kaplansky, I. Razgon, and R. Zivan. "Comparing performance of distributed constraints processing algorithms". In: *AAMAS workshop on Distributed Constraint Reasoning (DCR)*. 2002, pp. 86–93. URL: https://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.57.8606.

[17] A. Meisels. *Distributed search by constrained agents: algorithms, performance, communication*. Springer Science & Business Media, 2007. DOI: 10.1007/978-3-642-24013-3_2.

[18] P. J. Modi, W. M. Shen, M. Tambe, and M. Yokoo. "Adopt: Asynchronous distributed constraint optimization with quality guarantees". In: *Artificial Intelligence* 161.1-2 (2005), pp. 149–180. DOI: 10.1016/j.artint.2004.09.003.

[19] D. T. Nguyen, W. Yeoh, H. C. Lau, and R. Zivan. "Distributed Gibbs: a linear-space sampling-based DCOP algorithm". In: *Journal of Artificial Intelligence Research (JAIR)* 64 (2019), pp. 705–748. DOI: 10.1613/jair.1.11400.

[20] B. Ottens, C. Dimitrakakis, and B. Faltings. "DUCT: An Upper Confidence Bound Approach to Distributed Constraint Optimization Problems". In: *ACM Transactions on Intelligent Systems and Technology* 8.5 (2017). DOI: 10.1145/3066156.

[21] A. Petcu and B. Faltings. "DPOP: a scalable method for multiagent constraint optimization". In: *International Joint Conference on Artificial Intelligence (IJCAI)*. 2005, pp. 266–271. URL: https://www.ijcai.org/Proceedings/05/Papers/0445.pdf.

[22] M. Quigley, K. Conley, B. P. Gerkey, J. Faust, T. Foote, J. Leibs, R. Wheeler, and A. Y. Ng. "ROS: an open-source Robot Operating System". In: *ICRA workshop on open source software*. 2009. URL: https://www.researchgate.net/publication/303138182.

[23] C. E. Rasmussen and C. K. I. Williams. *Gaussian processes for machine learning*. MIT Press, 2006. ISBN: 026218253X. URL: http://www.gaussianprocess.org/gpml/chapters/RW.pdf.

[24] Raspberry Pi Foundation. *Raspberry Pi 3B+*. 2019. URL: https://www.raspberrypi.org/products/raspberry-pi-3-model-b-plus/.

[25] C. Robin and S. Lacroix. "Multi-robot target detection and tracking: taxonomy and survey". In: *Autonomous Robots* 40.4 (2016), pp. 729–760. DOI: 10.1007/s10514-015-9491-7.

[26] P. Rust, G. Picard, and F. Ramparany. "pyDCOP, a DCOP library for IoT and dynamic systems". In: *International workshop on Optimisation in Multi-Agent Systems (OptMAS)*. Montréal, Canada, 2019. URL: https://hal.archives-ouvertes.fr/hal-02098294.

[27] R. Stranders, E. Munoz De Cote, A. Rogers, and N. R. Jennings. "Near-optimal continuous patrolling with teams of mobile information gathering agents". In: *Artificial Intelligence* 195 (2013), pp. 63–105. DOI: 10.1016/j.artint.2012.10.006.

[28] E. Tsang. *Foundations of Constraint Satisfaction*. London: Academic Press, 1993. DOI: 10.1016/C2013-0-07627-X.

[29] S. M. Veres, L. Molnar, N. K. Lincoln, and C. P. Morice. “Autonomous vehicle control systems – a review of decision making”. In: *Proceedings of the Institution of Mechanical Engineers, Part I: Journal of Systems and Control Engineering* 225.2 (2011), pp. 155–195. DOI: 10.1177/2041304110394727.

[30] W. Yeoh, A. Feiner, and S. Koenig. “BnB-ADOPT: an asynchronous branch-and-bound DCOP algorithm”. In: *Journal of Artificial Intelligence Research (JAIR)* 38 (2010), pp. 85–133. DOI: 10.1613/jair.2849.

[31] W. Yeoh and M. Yokoo. “Distributed problem solving”. In: *AI Magazine* 33 (2012), pp. 53–65. DOI: 10.1609/aimag.v33i3.2429.

[32] M. Yokoo, E. H. Durfee, T. Ishida, and K. Kuwabara. “The distributed constraint satisfaction problem: formalization and algorithms”. In: *Transactions on Knowledge and Data Engineering* 10.5 (1998), pp. 673–685. DOI: 10.1109/69.729707.

[33] R. Zivan, T. Parash, and Y. Naveh. “Applying max-sum to asymmetric distributed constraint optimization”. In: *International Joint Conference on Artificial Intelligence (IJCAI)*. 2015, pp. 432–439. URL: https://dl.acm.org/doi/abs/10.5555/2832249.2832309.

6 CONCLUSIONS AND FUTURE RESEARCH

This Ph.D. thesis has focused on the development of algorithms that can optimize the actions of autonomous vehicles. The leading use cases in this thesis are underwater search operations with multiple AUVs. These operations suffer from some fundamental restrictions, especially when the search area contains potentially dangerous objects. This will require a relatively large distance between the support vessel and the AUVs. The first restriction is related to the communications caused by the attenuation of radio signals in seawater. The energy of acoustic signals is greatly reduced as the frequency is increased. For that reason, to communicate with the support vessel over large distances, low frequencies need to be used which requires large transducers and offers low data rates. This makes centralized *in-situ* optimization by the support vessel of the trajectories of the AUVs infeasible. The second restriction is the available computational power onboard the AUVs, which makes the centralized *in-situ* optimization by a single AUV (for all other AUVs) unrealistic. For these reasons, a distributed approach is necessary for which all AUVs cooperate through the exchange of messages and share the computations. In this chapter, an overview is given of the contributions of the research described in this thesis, as well as recommendations for future research topics.

6.1. RESEARCH CONTRIBUTIONS

This thesis focuses on the cooperation between autonomous agents to efficiently and effectively complete an objective. It contributes to the state-of-the-art in modeling and cooperative optimization of multi-agent systems. The goal of the research is to quantify the global performance of agents based on local utility functions and inter-agent collaboration. The focus of the research is to further develop C-DCOP solvers that can be applied to various real-world problems.

6.1.1. ALGORITHMS DEVELOPED

The presented Compression-DPOP (C-DPOP) algorithm in Chapter 2, is an extension of Distributed Pseudotree Optimization Procedure (DPOP) [9] algorithm. C-DPOP iteratively discretizes the continuous domains of a C-DCOP and dynamically updates the domains after each iteration based on the found (local) optimum. The computational and memory requirements of the algorithm can be selected using two parameters. These parameters make C-DPOP highly suitable for application to autonomous vehicles that are restricted by processing power and/or communicational bandwidth.

While C-DPOP outperforms DPOP, it does not offer any guarantees about the quality of the solution and could get stuck at local optima. Furthermore, the efficiency of the C-DPOP algorithm could be improved when the properties of the utility functions are taken into account. This would enhance the discretization method from uniform to utility based. Regions of high utility could be estimated by incorporating the gained information from the solution of previous iterations.

The Distributed Bayesian (D-Bay) algorithm, presented in Chapter 4, possesses both these properties. All agents model the influence of their variables on the global utility as a Gaussian process [11]. The properties of the (local) utility functions are captured by selecting an appropriate kernel. Instead of discretization of the continuous domains, the agents iteratively select *samples* from the domains. Sampling refers to the determination of the utility of a value within the domain. The selection of the samples is performed by Bayesian optimization [7] in which an acquisition function balances exploration and exploitation of the search space based on the kernel and the previous samples. Under mild conditions (known Lipschitz constants of the utility functions), D-Bay is proven to converge to the global optimum of the C-DCOP.

Compared to DCOP solvers, which require discretization of the C-DCOP, it results in a reduction of the computational and memory demands of the individual agents. The D-Bay is a versatile algorithm that can be applied to numerous real-world C-DCOPs. The algorithm is sample-efficient in terms of the number of samples required by the agents to approach the optimum. During execution, multiple small-sized messages are exchanged between the agents. This makes the D-Bay algorithm especially suitable for problems that have a relatively small number of agents and computationally demanding utility functions.

6.1.2. EXPERIMENTS PERFORMED

Both the C-DPOP as well as the D-Bay algorithm were applied to benchmark and to real-world problems to evaluate their performance and applicability. This required modeling of the problems within the C-DCOP framework.

The C-DPOP algorithm was applied to a mobile sensor coordination (benchmark) problem [14] in Chapter 2. This problem was selected because it is a well-known real-world problem for mobile sensors. In this problem, multiple autonomous agents equipped with sensors (with limited sensing range) need to coordinate their position on a two-dimensional surface to get the most accurate reading of multiple targets.

Modeling this problem as a C-DCOP, instead of a DCOP, showed the potential of C-DCOPs for real-world problems. Traditionally, modeling this problem as a DCOP would require a trade-off between solution quality and the size of the search space. That is because all possible values need to be defined in the (finite) discrete domains of the variables. Discretization of the two-dimensional surface to a high resolution would allow for accurate positioning of the agents but will increase the size of the search space. A C-DCOP does not have this trade-off, while still being able to set the bounds of the variables through the continuous domains. Taking this into account, modeling real-world problems as C-DCOPs offers great benefits over modeling as (traditional) DCOPs.

Within the mobile sensor coordination problem, the location of the targets is known by the agents and cooperation is focused on finding the optimal sensor locations. A (static) sensor coordination problem was modeled in Chapter 4, in which both the number of targets and the location of the targets are unknown. A real-world analogy of this problem is the optimization of the orientation of multiple cameras that can change their observation angle to observe (and classify) targets.

The (local) utility functions defined within a C-DCOP are continuous. In other words, changes to the inputs of the function will result in comparable changes to the output. This property is not always shared by the utility functions of DCOPs, which are allowed to be discontinuous. Taking advantage of this property, D-Bay was able to balance exploration and exploitation of the search space. The effect of the balancing could be seen by an increase in performance after only a few view angles were sampled. Therefore, by modeling a problem as a C-DCOP instead of a DCOP, algorithms can take advantage of the additional information embedded within the problem definition.

Proceeding towards more realistic problems, the Mine Counter-Measures (MCM) operation was modeled in Chapter 3. The operation was modeled as a distributed segmentation problem for an area for multiple cooperative Autonomous Underwater Vehicles (AUVs). The utility of the segmentation is based on global performance metrics that are set by an operator. The metrics related to the expected time of completion (of the search) and the level of confidence that all mine-like objects within the area have been detected. The MCM operation was simulated within the high-fidelity UUV simulator [6]. The performance of the side-scan sonar sensors of the AUVs was adjusted during operation. After each leg of their trajectories, the AUVs assessed their sonar performance and communicated if it resulted in a change in (global) utility. The modeling as a C-DCOP allowed for *in-situ* optimization of the metrics based on (measured) sonar performance by the agents. The agents jointly optimized their trajectories by communicating the utility of their trajectories, as defined by the metrics, not the actual trajectories themselves. Modeling the MCM operation in terms of utility thereby enabled the usage of heterogenous agents without changing the problem definition.

Finally, a real-world autonomous search use case with multiple Unmanned Aerial Vehicles (UAVs) was modeled in Chapter 5. The use case was modeled within the C-DCOP framework as a segmentation problem that is closely related to the MCM operation. During the experiments, a low-bandwidth meshed network was used to mimic the limited communication capabilities of underwater vehicles. It allowed for a realistic implementation of the D-Bay algorithm and showed the potential of the algorithm for application within an MCM operation. Experiments were used to validate a simulation environment in which additional variations of the problem were performed to assess the performance of the D-Bay algorithm. To the best of my knowledge, this was the first reported application of a C-DCOP algorithm on robotic platforms. Only a few other real-world experiments are presented within the literature, however, these experiments involve DCOP algorithms. The interested reader is referred to Yedidsion *et al.* [13] and Jain *et al.* [3] for two examples involving mobile sensor teams. Essentially, the contrast between a large number of benchmark problems and a small number of real-world experiments highlights the difficulties in the application of both DCOP and C-DCOP algorithms.

In conclusion, there are several benefits to modeling benchmark problems and real-world problems as C-DCOPs. The need for discretization is removed entirely, thereby not restricting the values but allowing the algorithms to sample them through their methods. Furthermore, the algorithms can take advantage of the properties of the utility functions to efficiently solve the problem. Additionally, modeling a real-world problem as a C-DCOP allows for the abstraction of processes that are contained within the agents. This in turn allows for heterogenous agents to collaborate without changing the problem definition. Finally, the experiments performed within the research of this thesis showed the successful real-world application of a C-DCOP algorithm. Hopefully, this will aid fellow researchers to move from benchmark problems toward experiments as well.

6.2. RECOMMENDATIONS

In this section, the two developed algorithms are compared based on their characteristics and recommendations are given related to their application.

6.2.1. ALGORITHM CHARACTERISTICS

In Table 6.1, the two developed algorithms are compared based on their characteristics. Both algorithms can be classified as iterative anytime algorithms and are therefore dependent on the number of iterations (I). The C-DPOP algorithm shares most of its characteristics with DPOP, due to the similarity in message passing. For this reason, the runtime complexity, memory, and message size are highly dependent on the width of the pseudo tree (w) and the cardinality of the domains (d). The number of messages is unaffected by the width of the pseudo tree and scales according to the number of variables (N) of the problem.

The D-Bay algorithm, on the other hand, is mostly affected by the depth of the tree (t) and to a minor extent by the number of children (C) for its characteristics. Specifically, the increase in the number of messages sent during optimization is exponential based on the depth of the tree.

Table 6.1.: Comparison of characteristics of developed algorithms.

	Runtime		Message	
Algorithm	Complexity	Memory	Number	Size
C-DPOP	$O(Id^w)$	$O(d^w)$	$O(IN)$	$O(d^w)$
D-Bay	$O(I)$	$O(I)$	$O(CI^t)$	$O(t)$

The two parameters of the C-DPOP algorithm allow for the trade-off between the message size and the number of messages. The selection of the cardinality of the domains is the most prominent since this directly affects the size of the messages. The compression factor (c) indirectly defines the number of iterations based on the required resolution (r) of the solution and thereby the number of messages. The resolution is the distance between the samples of the domain during an iteration.

In setting the parameters for the C-DPOP algorithm, it is recommended to set the value of the domain cardinality as large as the memory of the agents (and/or the message size) allows. Increasing the domain cardinality will exponentially decrease the number of iterations needed to achieve the required resolution. Additionally, the compression factor should be set as close to 1 to reduce the size of the domain that is discarded after each iteration. This improves the chance of escaping local optima in between the iterations. The C-DPOP algorithm can be stopped after its allowed computational time or the resolution is reached. Therefore, the combination of a large domain cardinality and a high compression factor will sample the domain as finely as possible within the computational boundaries of the platform.

The D-Bay algorithm does not have explicit parameters to set other than the termination criterium. The criterium can be defined based on the number of iterations or the (relative) increase in utility in between samples. The convergence to the global optimum, however, does depend on the Lipschitz constants of the (local) utility functions. When these constants are not known exactly for all utility functions there are several methods to overcome this issue. The first method is to overestimate their values. This ensures the algorithm will not exclude any areas of the domains that could hold the optimum. Consequently, however, areas that certainly do not hold the optimum will be sampled and therefore reduce the efficiency of the algorithm. The second method is to deduce the Lipschitz constants during the sampling of the utility functions. In the literature, several estimation algorithms have been developed. The reader is referred to Jones *et al.* [4] and Wood *et al.* [12] for two examples. While estimating the Lipschitz constant *in situ* can retain the efficiency of the D-Bay algorithm, the estimation of the Lipschitz constant of a function can be computationally intensive [12]. This trade-off would be worthwhile to investigate to improve the sampling within the D-Bay algorithm by choosing samples that strike a balance between finding the optimal value (exploitation) and estimating the Lipschitz constant (exploration).

6.2.2. PROBLEM CHARACTERISTICS

Based on the properties of the developed algorithms as stated above, the following recommendations can be made based on the properties of the platforms and the problems.

Computational constraint platforms In the case of computationally constrained platforms, the choice between the developed algorithms would depend on the complexity of the utility functions. When the utility functions are computationally intensive, the D-Bay algorithm samples the domains more efficiently, thereby requiring fewer function evaluations. If the evaluation of the utility functions is computationally cheap, the C-DPOP algorithm would be preferred as no meta-optimization algorithm is required, which could result in significant computational overhead.

Memory constraint platforms In the case of memory constraint platforms, the D-Bay algorithm would be preferred since only the latest optimized values are stored. While for the reasons stated above, the C-DPOP algorithm requires a significant amount of memory to store (and send) the messages. Decreasing the amount of memory could require the domain cardinality to be set so low as to reduce the efficiency of the C-DPOP algorithm. Decreasing the domain cardinality could cause the sampling of the domains to be very coarse, and thereby increase the chance of getting stuck in local optima.

Communicational constraint platforms When the problem does not allow for large messages to be sent between the platforms, the D-Bay algorithm would be preferred. The D-Bay algorithm sends small messages over the network which is possible in most environments (given that communication is possible at all). Larger messages could become corrupted during transit and would require the resending of the message entirely. The D-Bay algorithm could, depending on the properties of the pseudo-tree, require a large number of messages. If this property is inconvenient for the communication network, the C-DPOP algorithm can be used since its communicational properties can be tuned through its parameters.

Pseudo-tree properties A wide pseudo-tree is the result of a highly coupled problem in which most agents *share* a utility function. In other words, the effect of an agent has a direct effect on that of all others. This hold for most *combinatorial* problems such as area search problems. For problems that result in *wide* pseudo-trees, the D-Bay algorithm is to be preferred as the number of messages within the D-Bay algorithm scales with the *depth* of the tree.

For problems that result in *deep* pseudo-trees, the C-DPOP algorithm is to be preferred, as the properties of the algorithm are mostly related to the width of the tree. For that reason, a problem that results in a deep tree can be optimized efficiently. These problems often relate to loosely coupled problems, in which there is less interdependency between the local utility functions and the global (aggregate) utility.

6.3. FUTURE RESEARCH

During the development of the algorithms presented in this thesis, theoretical and practical questions were raised that can serve as a foundation for future research. In this section, ideas are presented based on these questions related to algorithm design, multi-agent system modeling, and performing multi-agent experimentation.

6.3.1. ALGORITHM DESIGN

In this thesis, algorithms to solve C-DCOPs are developed. Their development highlighted several key points that could be addressed in more detail in future work. The key points can be summarized as:

Develop adaptive communication strategies In real-world scenarios, there will be significant communicational limitations, not only when the bandwidth is constrained due to environmental conditions, but also in some situations, the communication could be unreliable due to a greatly fluctuating bandwidth. These fluctuations can be caused by background noise, variations within the transmission medium, or reflections. The approach taken in this thesis was to limit the size of the messages to allow for the highest chance of the messages arriving. An alternative approach would be to include the dynamic characteristics of the communication capabilities holistically. Using this approach, the algorithms would have to explicitly take the available bandwidth into account and adjust their messaging behavior accordingly. For example, upon detection of high bandwidth agents could send either a few large messages or a lot of smaller messages. When the bandwidth is limited, the algorithms could adjust the size and/or the number of messages or adjust their position to be able to communicate more effectively.

Develop flexible optimization strategies The available computational power for autonomous vehicles is increasing at an impressive rate. Three complementary factors can be identified: 1) the introduction of specialized hardware, such as Graphical Processing Units (GPUs) and Tensor Processing Units (TPUs); 2) the increase in performance per watt; 3) the increase in battery capacity.

These trends allow for more flexibility within the 'computation vs. communication' paradigm. At present, most developed algorithms can be classified according to a scale that ranges from distributed to centralized optimization. Distributed algorithms rely heavily on communication between agents, while centralized algorithms rely on computation. Breaking away from these static categories, algorithms could be designed that dynamically update the balance between computation and communication during optimization. If sufficient communication is available, all agents solve the global problem in a distributed manner. Additionally, the autonomous agents could enquire other agents about their internal models and optimization methods. Afterward, when no communication is available, all autonomous agents could rely on their computational power to solve a centralized problem and choose their actions accordingly. Such strategies would demand that all agents are aware of their communicational and computational potential and act accordingly.

Develop multi-layered hierarchical optimization methods The algorithms developed in this thesis optimize towards the global goal defined within the C-DCOPs. The goal typically refers to objectives that are defined over a set length of time and have a clear finish condition.

In the underwater search operation with AUVs, this is either a deadline (predetermined allocated time for the operation) or an objective (scan a certain area of the seabed). In practice, these tasks are scarcely performed in isolation. For example, after an object is detected on the seabed, a follow-up task would be to classify this object to determine whether the object is hazardous. Currently, the positions of the objects are stored (or transmitted to a support vessel). If the AUVs would be able to collect the visual camera images autonomously, this would greatly reduce the time required for classification.

Extending the C-DCOP framework into a hierarchical framework would separate the total problem into subproblems per level, comparable to separating the global objective into individual utility functions in a C-DCOP. The higher levels could represent tasks within the MCM operation (e.g. detection, classification, identification) while the lower levels represent the execution of these tasks (e.g. trajectory optimization, adjusting sensor parameters, scheduling communication events). The hierarchical structure could be exploited by specialized algorithms to efficiently solve the total problem.

6.3.2. Multi-agent system problems

In this thesis, benchmark problems (sensor coordination problems) and real-world problems (MCM, area segmentation) have been modeled within the C-DCOP framework. The DCOP framework has been extended in order to enable the modeling of numerous problems. The interested reader is referred to the survey of Fioretto *et al.* [2] for an overview. In order to incorporate additional classes of problems, some specific extensions could be investigated:

Hybrid problems A characteristic of many real-world problems is the inclusion of both discrete and continuous variables. The discrete variables typically represent *states* of the agents, such as operational modes. Within these states, the (control) variables of the agents are continuous. Extending the DCOP framework to a combination of both types of domains would allow for integrated modeling of these problems. For the MCM operation, such a framework would support the optimization of the sensor configuration (e.g. sensor type selection, sensor mode optimization).

Semantically modeled problems The Entity-Relationship (ER) model, introduced by Chen [1], was originally developed for database design. It proved to be a powerful method to model both databases and ontologies for usage in artificial intelligence research [8]. The ER model is centered around abstractions of various concepts to encapsulate complexities. The *entities* can represent objects (e.g. humans, objects) and can be defined based on types. The entities are linked to each other through *relationships*. A common description of entities and relationships are nouns and verbs, respectively. For example, when modeling the communication between two agents within the ER model, entities agent A and agent B share a *communicate* relationship. A DCOP can be modeled as agent entities with utility and communication relationships. Leveraging the potential of the versatility of the ER model, real-world problems could be modeled in greater detail. Additionally, algorithms could be designed to take advantage of the structure of the entities and their relationships to efficiently solve the modeled problem.

6.3.3. Experimental

In this thesis, experiments with UAVs were performed for a cooperative area search problem. Preparing and performing the experiments showed the large disparity between benchmark problems and real-world applications. Within the literature, much attention is given to the development of algorithms to solve benchmark problems. While this allows for objective comparisons between algorithms, their implementation potential for real-world problems is often neglected. To aid the application of the developed algorithms in the real world, some topics of research could be addressed:

Development of simulation environments for distributed optimization When developing distributed optimization algorithms researchers often need to set up their simulation environments to test their algorithms. This often results in (highly) custom environments that cannot be utilized by other researchers. Additionally, it takes a substantial amount of time and effort to develop these environments. Open-source projects, such as ROS [10] and Gazebo [5], facilitate the development of generic environments. These projects serve as a foundation for other initiatives to implement use case-specific extensions. A good example is the UUV simulator [6] in which sensors and actuators for underwater vehicles are implemented. Extending such initiatives towards generic distributed optimization would boost the applicability and usability of distributed optimization as a whole. Another benefit is that the open-source development of such initiatives boosts collaboration between researchers. It is important to note that these benefits are not confined to the development of DCOP algorithms.

Development of holistic simulation environments Simulation environments allow for more realistic applications of algorithms, especially for multi-agent problems. While they offer great value in the design and validation of the algorithms, it will depend on the fidelity of the simulation environment if the results will match experimental outcomes.

These effects are well-known for sensor imperfections, such as noise and outliers, which are (almost) always present during experiments. When not taken into account, these imperfections could cause severe problems in the operation of the algorithms. The physical interactions between an autonomous system and the environment are usually modeled with a high level of detail.

However, all interactions with the algorithm (sensor information, (optimization) problem definition, algorithm execution, communication, and actuator control) must be taken into account and implemented realistically. Primarily to discover several types of practical issues during the early stages of algorithm development. An example of one of those issues is the execution time of an algorithm. It can affect the safety of an autonomous system when the execution of the algorithm takes too long, e.g. in handling obstacle avoidance. Long execution times could also result in an autonomous system that is idle for the majority of the experiment. Furthermore, communication issues might lead to dead-lock situations when the algorithm waits until it has received an acknowledgment from another agent. Finally, issues concerning failures within the autonomous system are often neglected within simulation environments. During real-world experiments, all components are subject to failure. Designing an algorithm that can explicitly handle failures is crucial for its reliability. If not handled correctly, a failure of a low-level process could cascade throughout the system and cause the entire system to fail. An example is the availability of sensor information. Moreover, without any checks on the sensor information, expired sensor values could persist and be treated as the latest. If the sensor is used for obstacle avoidance the consequences could be disastrous. In conclusion, the development of a holistic simulation environment will not only improve the fidelity of the simulations but will also improve the reliability of the algorithms during real-world experiments.

REFERENCES

[1] P. P.-S. Chen. "The entity-relationship model—toward a unified view of data". In: *ACM Transactions on Database Systems* 1.1 (Mar. 1976), pp. 9–36. DOI: 10.1145/320434.320440.

[2] F. Fioretto, E. Pontelli, and W. Yeoh. "Distributed constraint optimization problems and applications: a survey". In: *Journal of Artificial Intelligence Research (JAIR)* 61 (2018), pp. 623–698. DOI: 10.1613/jair.5565.

[3] M. Jain, M. Taylor, M. Tambe, and M. Yokoo. "DCOPs meet the real world: exploring unknown reward matrices with applications to mobile sensor networks". In: *International Joint Conference on Artificial Intelligence (IJCAI)*. 2009, pp. 181–186. URL: https://www.ijcai.org/Proceedings/09/Papers/040.pdf.

[4] D. R. Jones, C. D. Perttunen, and B. E. Stuckman. "Lipschitzian optimization without the Lipschitz constant". In: *Journal of Optimization Theory and Applications* 79 (1 Oct. 1993), pp. 157–181. DOI: 10.1007/BF00941892.

[5] N. Koenig and A. Howard. "Design and use paradigms for Gazebo, an open-source multi-robot simulator". In: *International Conference on Intelligent Robots and Systems (IROS)*. 2004, pp. 2149–2154. DOI: 10.1109/IROS.2004.1389727.

[6] M. M. M. Manhães, S. A. Scherer, M. Voss, L. R. Douat, and T. Rauschenbach. "UUV simulator: a Gazebo-based package for underwater intervention and multi-robot simulation". In: *OCEANS*. 2016, pp. 1–8. DOI: 10.1109/OCEANS.2016.7761080.

[7] J. Mockus. "The Bayesian approach to global optimization". In: *System Modeling and Optimization* 38 (1982), pp. 473–481. DOI: 10.1007/BFb0006170.

[8] N. F. Noy. "Semantic integration: a survey of ontology-based approaches". In: *SIGMOD Record* 33.4 (Dec. 2004), pp. 65–70. DOI: 10.1145/1041410.1041421.

[9] A. Petcu and B. Faltings. "DPOP: a scalable method for multiagent constraint optimization". In: *International Joint Conference on Artificial Intelligence (IJCAI)*. 2005, pp. 266–271. URL: https://www.ijcai.org/Proceedings/05/Papers/0445.pdf.

[10] M. Quigley, K. Conley, B. P. Gerkey, J. Faust, T. Foote, J. Leibs, R. Wheeler, and A. Y. Ng. "ROS: an open-source Robot Operating System". In: *ICRA workshop on open source software*. 2009. URL: https://www.researchgate.net/publication/303138182.

[11] C. E. Rasmussen and C. K. I. Williams. *Gaussian processes for machine learning*. MIT Press, 2006. ISBN: 026218253X. URL: http://www.gaussianprocess.org/gpml/chapters/RW.pdf.

[12] G. Wood and B. Zhang. "Estimation of the Lipschitz constant of a function". In: *Journal of Global Optimization* 8 (1 1996), pp. 91–103. DOI: 10.1007/bf00229304.

[13] H. Yedidsion and R. Zivan. "Applying DCOP_MST to a team of mobile robots with directional sensing abilities". In: *International Conference on Autonomous Agents and Multiagent Systems (AAMAS)*. 2016, pp. 1357–1358. URL: https://dl.acm.org/doi/10.5555/2936924.2937158.

[14] R. Zivan, H. Yedidsion, S. Okamoto, R. Glinton, and K. Sycara. “Distributed constraint optimization for teams of mobile sensing agents”. In: *Autonomous Agents and Multi-Agent Systems* 29.3 (2015), pp. 495–536. DOI: 10.1007/s10458-014-9255-3.

A THE DISTRIBUTED BAYESIAN ALGORITHM

In this appendix, the formal description of the distributed Bayesian algorithm introduced in Chapter 4 is presented.

Algorithm 2: Distributed Bayesian (D-Bay) for agent a_i

```
Input  : $\mathbf{P}_i, \mathbf{PP}_i, \mathbf{C}_i, \mathbf{PC}_i, \mathbf{F}_{a_i}, \mathbf{F}_{\mathbf{P}_i}, \mathbf{X}_i, \kappa$
Output: $\hat{\rho}_{\mathbf{X}_i}$
Initialization
    if root agent then
        while not threshold reached do
            $\mathfrak{U}_i^j$ := optimizeLocalVariables($\emptyset$);
        end
        processFinal($\emptyset$);
when received sample $\mathfrak{S}_j$ from parent $\mathbf{P}_i$
    while not threshold reached do
        $\mathfrak{U}_i^j$ := optimizeLocalVariables($\mathfrak{S}_j$);
    end
    send($\mathbf{P}_i$, $\mathfrak{U}_i^j$);
when received final $\hat{\mathfrak{S}}_j$ from parent $\mathbf{P}_i$
    processFinal($\hat{\mathfrak{S}}_j$);
```

```
Function optimizeLocalVariables($\mathfrak{S}_j$)
    $\rho_{\mathbf{X}_i}$ := computeOptimalSample($\kappa$);
    $\mathfrak{S}_i := \mathfrak{S}_j \cup \{\rho_{\mathbf{X}_i}\}$;
    $\mathfrak{U}_i^j$ := calculateUtility($\mathfrak{S}_i$);
    return $\mathfrak{U}_i^j$;
```

```
Function calculateUtility($\mathfrak{S}_i$)
    $\mathfrak{U}_i := \min_{\rho \in \Sigma_{\mathbf{X}_i}} \eta_{f_n \in \mathbf{F}_i} \left( f_n(\rho_{\mathbf{V}_n} \mid \mathfrak{S}_j) \right)$;
    if $\mathbf{C}_i \neq \emptyset$ then
        $\hat{\mathfrak{U}}_i :=$ getChildUtility($\mathfrak{S}_i$);
        $\mathfrak{U}_i^j := \eta \left( \mathfrak{U}_i, \hat{\mathfrak{U}}_i \right)$;
    else
        $\mathfrak{U}_i^j := \mathfrak{U}_i$;
    storeUtility($\mathfrak{U}_i^j$, $\mathfrak{S}_i$);
    return $\mathfrak{U}_i^j$;
```

```
Function getChildUtility($\mathfrak{S}_i$)
    foreach $a_k \in \mathbf{C}_i$ do send($a_k$, $\mathfrak{S}_i$);
    when received $\mathfrak{U}_k^i$ from all $a_k \in \mathbf{C}_i$
        $\hat{\mathfrak{U}}_i := \eta_{a_k \in \mathbf{C}_i} \left( \mathfrak{U}_k^i \right)$;
    return $\hat{\mathfrak{U}}_i$;
```

```
Function processFinal($\hat{\mathfrak{S}}_j$)
    $\hat{\rho}_{\mathbf{X}_i} :=$ retrieveOptimalLocalSample($\hat{\mathfrak{S}}_j$);
    $\hat{\mathfrak{S}}_i := \hat{\mathfrak{S}}_j \cup \{\hat{\rho}_{\mathbf{X}_i}\}$;
    foreach $a_k \in \mathbf{C}_i$ do send($a_k$, $\hat{\mathfrak{S}}_i$);
```

B DIRICHLET KERNEL INTERVAL FUNCTIONS

In this appendix, the derivation of the mean and variance function corresponding to the Dirichlet kernel is presented as introduced in Chapter 4.

As shown in the work of Ding *et al.* [1, Theorem 2], a kernel κ of the Markovian class reduces the mean function $\mu_s(\cdot)$ and the variance function $\sigma_s^2(\cdot|\mathcal{O})$ of the posterior on the interval between observations as given in Equations (4.8) and (4.9), respectively. For the Dirichlet kernel as defined by Equation (4.10), for a normalized domain $x_i, x_j \in [0,1]$ and the kernel scale parameter λ, the non-zero elements of the $\boldsymbol{K}_s^{-1}(\mathcal{O})$ matrix are given by

$$(\boldsymbol{K}_s^{-1}(\mathcal{O}))_{s,s} = \begin{cases} \lambda^{-2}\frac{x_1}{x_1\left(x_2-x_1\right)}, & \text{if } s=1, \\ \lambda^{-2}\frac{(x_{s+1}-x_{s-1})}{(x_s-x_{s-1})(x_{s+1}-x_s)}, & \text{if } s\in\{2,\dots,S-1\}, \\ \lambda^{-2}\frac{(1-x_{S-1})}{(1-x_S)(x_S-x_{S-1})}, & \text{if } s=S, \end{cases}$$

and

$$(\boldsymbol{K}_s^{-1}(\mathcal{O}))_{s-1,s} = (\boldsymbol{K}_s^{-1}(\mathcal{O}))_{s,s-1} = \frac{-\lambda^{-2}}{(x_s-x_{s-1})},\quad s=2,\dots,S.$$

The mean function $\mu_s(\cdot)$ and the variance function $\sigma_s^2(\cdot|\mathscr{O})$ for the Dirichlet kernel can be rewritten accordingly as

$$\begin{aligned}
\mu_s(x|\mathscr{O}) &= \boldsymbol{\kappa}_s^{\mathrm{T}}(x,\mathscr{O})\boldsymbol{K}_s^{-1}(\mathscr{O})\boldsymbol{y}_s(\mathscr{O}) \\
&= \begin{bmatrix} 0 & \dots & 0 & \frac{x_s-x}{x_s-x_{s-1}} & \frac{x-x_{s-1}}{x_s-x_{s-1}} & 0 & \dots & 0 \end{bmatrix} \begin{bmatrix} y_1 \\ \vdots \\ y_{s-2} \\ y_{s-1} \\ y_s \\ y_{s+1} \\ \vdots \\ y_S \end{bmatrix} \\
&= \begin{bmatrix} \frac{x_s-x}{x_s-x_{s-1}} & \frac{x-x_{s-1}}{x_s-x_{s-1}} \end{bmatrix} \begin{bmatrix} y_{s-1} \\ y_s \end{bmatrix} \\
&= \frac{y_{s-1}(x_s-x)+y_s(x-x_{s-1})}{x_s-x_{s-1}}
\end{aligned} \tag{B.1}$$

and

$$\begin{aligned}
\sigma_s^2(x|\mathscr{O}) &= \kappa\,(x,x) - \boldsymbol{\kappa}_s^{\mathrm{T}}(x,\mathscr{O})\boldsymbol{K}_s^{-1}(\mathscr{O})\boldsymbol{\kappa}_s(x,\mathscr{O}) \\
&= \lambda^2 x(1-x) - \begin{bmatrix} 0 & \dots & 0 & \frac{x_s-x}{x_s-x_{s-1}} & \frac{x-x_{s-1}}{x_s-x_{s-1}} & 0 & \dots & 0 \end{bmatrix} \begin{bmatrix} \lambda^2 x_1(1-x) \\ \vdots \\ \lambda^2 x_{s-2}(1-x) \\ \lambda^2 x_{s-1}(1-x) \\ \lambda^2 x(1-x_s) \\ \lambda^2 x(1-x_{s+1}) \\ \vdots \\ \lambda^2 x(1-x_S) \end{bmatrix} \\
&= \lambda^2 \left(x(1-x) - \left(\frac{x_{s-1}(1-x)(x_s-x)}{x_s-x_{s-1}} + \frac{x(1-x_s)(x-x_{s-1})}{x_s-x_{s-1}} \right) \right) \\
&= \lambda^2 \frac{-(x_s-x)(x_{s-1}-x)}{x_s-x_{s-1}}.
\end{aligned} \tag{B.2}$$

Curriculum Vitæ

Jeroen Edwin FRANSMAN

27–04–1988	Born in Rotterdam, The Netherlands

Education

2009–2012	**Master Systems & Control** Delft University of Technology *Thesis: Distributed Model Predictive Control for Greenhouse Climate* *Extracurricular: Fulltime team manager, TU Delft RoboCup (2010–2011)*
2006–2009	**Bachelor Mechanical Engineering** Delft University of Technology *Minor: Electrical Engineering for Autonomous Exploration Vehicles*
2000-2006	**Secondary education** (preparatory scientific education) Emmauscollege, Rotterdam (2000-2001) Maerlant college, Brielle (2001-2006)

Experience

2020–2022	**Research Scientist** *Intelligent Autonomous Systems (IAS) department* TNO, The Hague
2016–2022	**Ph.D. Student** *Delft Center for Systems and Control (DCSC)* Delft University of Technology
2013–2016	**Scientist Innovator** *Distributed Sensor Systems (DSS) department* TNO, The Hague

List of Publications

Peer-reviewed

1. J. Fransman, J. Sijs, H. S. Dol, E. Theunissen, and B. De Schutter. “Distributed constraint optimization for continuous mobile sensor coordination”. In: *European Control Conference (ECC)*. Limassol, Cyprus, 2018. DOI: 10.23919/ECC.2018.8550486
2. J. Fransman, J. Sijs, H. Dol, E. Theunissen, and B. De Schutter. “Distributed constraint optimization for autonomous multi AUV mine counter-measures”. In: *OCEANS MTS/IEEE*. Charleston, SC, USA, 2018. DOI: 10.1109/OCEANS.2018.8604924
3. J. E. Fransman, J. Sijs, H. Dol, E. Theunissen, and B. De Schutter. “The Distributed Bayesian algorithm: simulation and experimental results for a cooperative multi UAV search use-case”. In: *AAMAS workshop on Optimization and Learning in Multiagent Systems (OptLearnMAS)*. 2020

Under review

4. J. Fransman, J. Sijs, H. Dol, E. Theunissen, and B. De Schutter. “Distributed Bayesian: a continuous distributed constraint optimization problem solver”. In: *Submitted to JAIR* (2021)

Extended abstracts

5. J. Fransman, J. Sijs, H. Dol, E. Theunissen, and B. De Schutter. “Bayesian-DPOP for continuous distributed constraint optimization problems”. In: *International Conference on Autonomous Agents and Multiagent Systems (AAMAS)*. Montreal, Canada, 2019. URL: https://www.ifaamas.org/Proceedings/aamas2019/pdfs/p1961.pdf

Non-peer-reviewed

6. J. Fransman, J. Sijs, and B. De Schutter. “Distributed constraint optimization for mobile sensor coordination”. In: *Benelux meeting on Systems and Control*. 2018
7. J. E. Fransman and B. De Schutter. “Experimental results of distributed multi UAV search optimization”. In: *Benelux meeting on Systems and Control*. 2020
8. J. Fransman, J. Sijs, H. Dol, E. Theunissen, and B. De Schutter. “Distributed Bayesian: a continuous Distributed Constraint Optimization Problem solver”. In: *arXiv* (2020). arXiv: 2002.03252