Self-Organizing Multi-Agent Systems

Cornelis Jan van Leeuwen

Self-Organizing Multi-Agent Systems

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 8 februari 2021 om 15:00 uur

door

Cornelis Jan VAN LEEUWEN

Master of Science in Artificial Intelligence, University of Groningen, The Netherlands,
born in Delfzijl, The Netherlands.

Dit proefschrift is goedgekeurd door de

promotor: Prof. dr. K.G. Langendoen
copromotor: Dr. P. Pawełczak

Samenstelling promotiecommissie:

Rector Magnificus,	voorzitter
Prof. dr. K.G. Langendoen,	Technische Universiteit Delft
Dr. P. Pawełczak,	Technische Universiteit Delft
Onafhankelijke leden:	
Prof. dr. ir. D.H.J. Epema,	Technische Universiteit Delft
Dr. M.M. de Weerdt,	Technische Universiteit Delft
Prof. dr. J.L. Hurink,	Universiteit Twente
Dr. R. Zivan,	Ben-Gurion University of the Negev, Israel
Prof. dr. G. Picard,	École de Mines de Saint-Étienne, France
Reserve leden:	
Prof. dr. E. Visser,	Technische Universiteit Delft

Printed by: Ridderprint, The Netherlands

Cover design: Elizabeth Busey (Bloomington, IN, USA)
Emanation. Monoprint Collage. 2019
https://elizabethbusey.com/a-study-in-blues/

ISBN 978-94-6366-362-5

An electronic version of this dissertation is available at
http://repository.tudelft.nl/.

Those who can imagine anything, can create the impossible.

Alan Turing

Contents

Summary

Self-organizing multi-agent systems are systems, comprised of a group of independent agents, that are able to adapt and improve their collective behavior through local interactions, without external intervention. Self-organizing multi-agent systems depend less on human interaction in order stay operational and perform well under varying conditions. This is becoming more important as the scale of networks grows, and the complexity increases. Human operators will no longer be able to keep up with growing networks, or oversee the behavior that has emerged from many local interactions between the system's components. There are many applications where self-organization is becoming essential in order to continue to develop, for instance smart grids, autonomous vehicles, and computer and communication networks.

Self-organization is a property that can be hard to control; it is a property of the group, but it is defined by the behavior of the individual. Existing solutions often use ad hoc approaches by adding some additional mechanism that implements some (limited) form of self-organization, allowing it to adapt to changing environment. However, even if a system is suitable for changing its configuration at runtime, there is still the problem of coordinating the adaptations between agents. To solve that, cooperation between agents is required.

This thesis addresses the problem of providing self-organization to multi-agent systems, or improve the existing self-organizing capabilities. In order to do this, a framework for self-organization is proposed to guide and ease the implementation of self-organization, making sure that adaptive deployment is possible. Then, different mechanisms are provided that define *how* to cooperatively make decisions in a network.

A formalization for this distributed decision-making is called Distributed Constraint Optimization Problems (DCOP). In this formalization every decision variable is controlled by an agent, and agents have to communicate with one another in order to find variable assignments that minimize a set of constraints. A new DCOP-solving algorithm called CoCoA is introduced that differs from existing methods, by not relying on iteratively updating an existing solution. At its core, CoCoA shares information locally, so that agents can make decisions taking into account their effects on neighboring agents. This algorithm is extended in CoCoA_CA, which performs an extra step to avoid simultaneous decisions, this avoids potential conflicts in the outcome. Finally, a hybrid mechanism is proposed which combines the advantages of CoCoA and that of existing iterative algorithms.

CoCoA is evaluated in experiments using synthetic benchmark problems such as graph-coloring and semi-randomized problems. More realistic use cases for self-organization are provided in a greenhouse sensor network and wireless power transfer networks. Despite the generic approach for self-organization, there are problem classes that cannot be solved efficiently using a DCOP-based approach. An example is the power dispatch problem faced in smart grids, where there are constraints that involve *all* agents. For this problem instance, a custom decision-making algorithm called Local Pricing Receding Horizon is introduced based on a market mechanism, where agents negotiate prices to settle a power program.

Samenvatting

Zelforganiserende multi-agent systemen zijn systemen, gebouwd van een groep van onafhankelijk acterende agenten, welke hun collectieve gedrag kunnen aanpassen en verbeteren door lokaal met elkaar te interacteren, zonder invloed van buitenaf. Dit soort zelforganiserende multi-agent systemen zijn minder afhankelijk van menselijke interventie om operationeel te blijven en goed te blijven presteren onder variërende omstandigheden. Dit is nog belangrijker wanneer de schaal van het netwerk en de complexiteit groter worden. Voor een mens wordt het steeds lastiger om de grootte van het netwerk bij te benen, laat staan het complexe gedrag van lokale interacties tussen de componenten te overzien. Er bestaan reeds vele toepassingen waarbij zelforganisatie een belangrijke rol speelt voor de voortgaande ontwikkeling. Voorbeelden hiervan zijn intelligente elektriciteitsnetwerken, zelfrijdende auto's en uiteraard computer- en communicatienetwerken.

Zelforganisatie is een eigenschap die niet gemakkelijk te beheersen is; het is een eigenschap van de groep, maar wordt bepaald door het gedrag van de individu. Bestaande oplossingen gebruiken dan ook meestal *ad hoc* benadering om een gelimiteerde vorm van zelforganisatie te implementeren, waardoor het systeem in zich in zekere mate kan aanpassen aan de veranderende omstandigheden. Echter, ook al is een systeem technisch in staat om zichzelf aan te passen, dan resteert nog steeds het probleem om de aanpassingen te coördineren tussen de verschillende agenten. Om ook dit probleem op te lossen is er samenwerking nodig tussen de agenten.

In dit proefschrift wordt getracht zelforganisatie toe te voegen aan multi-agent systemen, of bestaande vormen van zelforganisatie te verbeteren. Om dit te bewerkstelligen wordt een framework geïntroduceerd wat het implementatieproces begeleid en vergemakkelijkt; hierdoor wordt een adaptief systeem mogelijk. Tevens worden enkele mechanismes voorgesteld om te redeneren over *hoe* een coöperatieve agent als onderdeel van een netwerk beslissingen zou moeten maken.

Een formele beschrijving van het gedistribueerde beslismodel is het zogenaamde *Distributed Constraint Optimization Problem* (DCOP). In deze formalisatie wordt elke variabele beheerd door een agent, en agenten moeten met elkaar communiceren om een toewijzing te bepalen waarmee de kosten van set van constraints wordt geminimaliseerd. Een algoritme voor DCOPs wordt geïntroduceerd genaamd CoCoA; het verschilt van bestaande algoritmes door niet iteratief een bestaande oplossing te verbeteren. In plaats daarvan wordt in CoCoA lokaal wat informatie gedeeld waardoor een agent een beslissing kan maken, rekening houdende met de

gevolgen daarvan op de agenten in zijn buurt. Het CoCoA algoritme wordt uitgebreid in het CoCoA_CA algoritme, waar een stap wordt toegevoegd om gelijktijdige beslissingen te voorkomen, teneinde een conflicterende situatie te vermijden. Tot slot wordt een hybride mechanisme voorgesteld wat de voordelen van CoCoA en bestaande iteratieve methoden met elkaar weet te combineren.

CoCoA wordt geëvalueerd in kunstmatige experimenten, zoals een *graph-coloring* probleem, evenals semi-willekeurige problemen. Meer realistische toepassingen van zelforganisatie worden behandeld in een sensornetwerk voor glastuinbouw en draadloze oplaadnetwerken. Ondanks de generieke aanpak van zelforganisatie, zijn er typen problemen die niet efficiënt kunnen worden opgelost middels de DCOP aanpak. Als voorbeeld hiervan wordt het *power dispatch* probleem beschreven, waarbij een planning moet worden gemaakt een elektriciteitsnetwerk. In dit probleem zijn er constraints tussen alle agenten, waardoor een DCOP niet schaalt. Specifiek voor dit probleem wordt een algoritme voorgesteld genaamd *Local Pricing Receding Horizon*. Dit algoritme gebruikt lokale prijsstrategieën en een gedecentraliseerd marktmechanisme, waarin agenten met elkaar onderhandelen over een prijs om tot een planning te komen.

1

Introduction

Networks of connected computer systems are becoming more and more present in our world. First, networks were relatively small, and every component of the network was carefully designed, engineered, tested, deployed and maintained [24, 127]. However, networks nowadays are vast complex systems and components are being added, updated and removed continuously. The dynamic nature of today's networks make it impossible for a human operator to control every aspect of a network, let alone a large network of hundreds, if not thousands, of systems.

There are many examples in the real world where networks consist of countless devices that are impossible to maintain by hand; sensor networks [154], smart grids [77], autonomous vehicles, IoT (Internet of Things) [8], cellular networks [68], and of course, the mother of all: *the Internet*. Mostly, nodes in these networks are controlled and maintained by other computer systems, but to a lesser extent, there are also systems that rely on *self-organization*.

Before we elaborate on self-organization, let us propose a definition, since this term is overloaded and may have different meanings in different contexts.

Definition 1.1 (Self-organization). Self-organization is a property of a system composed of subsystems, that are able to improve their collective behavior, either by emergent behavior, or explicit cooperation.

This definition means that a network system is self-organizing if it has the following characteristics [32]:

1. **Dynamic operation**. The system is capable of changing its mode of operation when appropriate.
2. **A lack of explicit external control**. The system receives no external trigger indicating either *when* or *how* to change its behavior.

3. **Decentralized control**. There is no central authority, instead the system components interact locally with one another, and either explicitly or implicitly affect each other's behavior.

Note that the last characteristic does not necessarily mean that all components are peers; instead there might be different roles amongst the nodes that define a hierarchy. This could be compared to a bee colony in which there are workers, drones, a queen and other types of bees, which all have their own roles and tasks.

Self-organization typically yields some highly desirable system properties such as increased robustness and reliability, less effort for (human) maintenance, and hence an overall reduce in costs. A related concept to self-organization is *autonomic computing*, which describes some of the same desirable properties. Instead of networked systems, this concept focuses on the self-managing capabilities of computer systems in general. By that line of logic, self-organization is to *networks* what autonomic computing is to *computers*. The concept was first described by the IBM paper on autonomic computing [75] where it was stated that:

> *Autonomic Computing helps to address complexity by using technology to manage technology.* [...] *self-managing autonomic capabilities anticipate IT system requirements and resolve problems with minimal human intervention.*

The same paper also provides a blueprint for self-managing systems: the MAPE-K framework as depicted in Figure 1.1. This is a generic framework for adaptive systems, in which different components *Monitor* the operation of the system at hand, *Analyze* the behavior and determine if any changes are required, *Plan* a new configuration and finally *Execute* the changes needed to come to a new state. These four components all have access to a common *Knowledge* source, which contains information of the system and how to operate it. The MAPE-K framework for self-adaptivity is often used for systems that have to operate in a dynamic environment, and is frequently referenced by other studies for instance in the domains of IoT [145], Wireless Sensor Networks [7] and Cyber Physical Systems [31].

In our definition of self-organization, the first method of attaining self-organizing behavior is through emergent behavior. In the field of artificial intelligence, the term *emergent behavior* refers to (seemingly) complex intelligent behavior of a system comprised of multiple simple subsystems. Often this means that a group of agents behaves in a collective intelligent manner, whereas each individual follows a simple set of rules [103]. A well-known example involves a flock of birds where the behavior of the group is very complex, yet if one simulates each bird following three simple rules to (i) avoid collisions, (ii) stay close together and (iii) follow the movement of neighbors, the complex behavior "emerges".

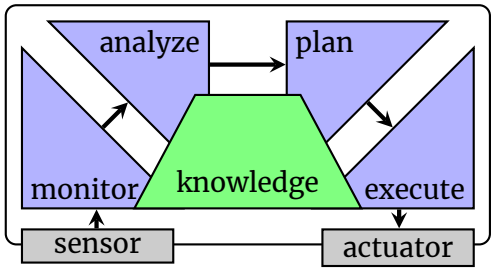

Figure 1.1: Diagram of the MAPE-K adaptation loop. Figure *redrawn* from [75].

The second method to attain self-organization is through explicit cooperation between components; this is the approach in *multi-agent systems*.

Definition 1.2 (Multi-agent Systems). Multi-agent Systems are systems in which multiple independent agents act in some shared environment, and solve problems while interacting with each other.

Multi-agent systems is a field of distributed Artificial Intelligence, which exists at least since the late 1980s [40, 50, 83]. Here, an agent is defined as a distinct entity, capable of making some decisions, with some interfaces for input and output. This definition includes software-based agents, agents in a simulated environment with a virtual presence, and agents with physical hardware connected (robots). Agents may either compete with each other in order to achieve their own individual goals, or they may cooperate to achieve a common goal. The philosophy behind cooperative multi-agent systems is to divide a large problem into many subproblems, which are subsequently solved by independent agents [67].

In the scope of network systems, it is common that every agent runs on, and controls one node. This means that the terms agents and nodes are used interchangeably when multi-agent systems are used to monitor and control network systems.

A formal method for describing cooperative multi-agent problems and the algorithms to solve these problems, is called *Distributed Constraint Optimization Problem* (DCOP). DCOPs are a type of distributed optimization problems in which variables are controlled by agents that have to find assignments to minimize a cost function over the complete set of variables [59]. This means that the agents share a common goal, and in order to perform well on the global task, have to pass messages between one another in order to coordinate their value assignments, and thus *self-organize*. A formal definition will be presented in Section 3.2.

DCOP originates from the Constraint Satisfaction Problem (CSP) [43]. CSP is a class of well-studies problems [59, 105], in which variables have to be assigned values from a domain, such that a set of constraints are satis-

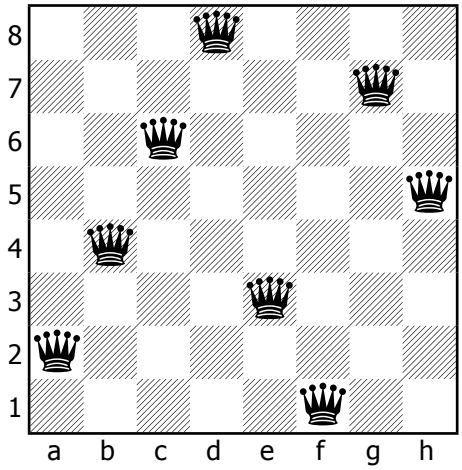

Figure 1.2: An example CSP is the 8-queens problem; here an example solution is shown in which no queen on the chessboard can take one of the others.

fied. These constraints are *logical* in that their result is either true or false. A famous example of a CSP is the eight queens problem; in this problem eight queens have to be places on a standard eight by eight squared chess board, such that no two queens are able to take one another. One example of a solution to this problem is shown in Figure 1.2.

Distributed Constraint Satisfaction (DCSP, or in other literature also: DisCSP), is an extension of CSP in which the variables are controlled by different agents [152]. The involved agents have to cooperate in order to find an allocation of their corresponding variables such that all constraints are satisfied. At the same time a generalization of CSP is the Constraint Optimization Problem (COP), in which the constraints are no longer logical, but instead can result in any non-negative real number. Then, the problem is no longer about finding a solution that satisfies all constraints, but instead to minimize the sum of all constraints. Similar to how DCSP is an extension of CSP, DCOP is an extension of COP.

By extending the eight-queens problem from a CSP to the class of DCOPs, would mean that instead of placing eight queens so that *not one* can take another; the problem would require placing, for example, ten queens in such a way that the fewest number of queens can take another queen. Moreover, for every queen there is a separate instance (agent) deciding where to place its assigned queen. This is just an example problem, but DCOPs are often used to formally describe and solve problems for instance for Wi-Fi channel allocation [34], power dispatch and balancing in Smart Grids [44] or even coordination of mobile (robot) search and rescue teams [148, 156].

There exists an extension of the DCOP framework called Asymmetric

DCOP (ADCOP) [56, 57], in which agents may value a set of constrained assignments differently. What this means is that constraints have different costs for the involved agents. This would allow to describe a relation between two agents in which one agent benefits from a certain assignment, but another agent is greatly hindered. For example, in a group of Wi-Fi routers, the transmission powers can be controlled in order to optimize the respective network throughputs. When one router sets it power to the maximum possible, it will increase the throughput of its *own* network, but will probably cause interference to neighboring networks. Hence, a more coordinated strategy is required.

In the context of dynamic systems, DCOPs have been applied as *superstabilizing* [117, 118] DCOP algorithms, that continuously adapt to changes in the environment. The max-sum algorithm can also run continuously [38], allowing for a best-effort optimum at any time. This strategy could pertain to changes in constraint costs, but it is unclear how added or removed nodes could be taken into account. Another more formal way of defining dynamic problems, lead to a family of algorithms dealing with Dynamic DCOPs (DDCOPs) [60, 61, 86, 149]. These algorithms proactively take into account potential future changes in the problem, and minimize the *expected* future costs, considering a model of foreseen future changes.

According to recent research, problems in self-organization in multi-agent systems can be categorized in either one of the following four main categories [147]:

1. **Task/Resource allocation** [39, 74, 153] is the problem where either tasks or resources have to be allocated to a group of agents in order to deal with them as efficiently as possible. Typically, in task allocation an agent cannot finish all tasks by itself (in time) and tasks have to be allocated to different agents to carry out. Moreover, there is often a dependency of tasks from one to another, which has to be taken into consideration when assigning tasks. Allocation of resources is a similar problem, where agent have to share (a set of) scarce resources such that all agents can perform their tasks.

2. **Relation adaptation and coordination** [53, 55, 82], also referred to as relation modification, studies how relations can be adapted in order to achieve efficient cooperation in a multi-agent system. A group of agents has to rearrange their internal structure in order to adapt to dynamic operating conditions and environments. This category includes leader election, group formation, and other problems in which agents change the relations (or type thereof) with other agents.

3. **Coordinated reasoning and learning** [1, 36] is the problem that agents face when they have to learn about the effect of their actions on the environment. Either by trial-and-error, or by an underlying model, agents must decide on how to act in the environment in order

to achieve a *better* state, according to some utility function. In the multi-agent domain, groups of agents can coordinate strategies in order to determine the most effective course of actions, especially when the strategies have to be in line with one another.

4. **Collective decision-making and planning** [14, 150] relates to economics and social sciences, and studies the issue of how to coordinate decisions made by agents. In many contexts of multi-agent systems, the choice of one agent influences the performance of those around it. Hence, decisions have to take into account actions of others, and should be coordinated to achieve the optimal group result. In a multi-agent context this is especially challenging since only local perception and agent-to-agent communication can be taken into account, since no agent has global knowledge. Planning can be considered a special case of decision-making, when a series of sequential choices have to be made.

These issues are overlapping, and in many real applications, solutions from multiple categories have to be applied.

1.1. Problem Statement

Although some existing networks already depend on self-organization to keep up with changing operating conditions, this property is hard to master, and there are often still many places to improve. The goal could be the reliability, efficiency, speed or cost of the system, but also that of the coordination strategy itself. In this thesis the self-organizing behavior in large networked systems is studied, and we formulate the main research question as follows:

> How can we achieve self-organization, or improve the self-organization capabilities in a network of cooperative agents?

In order to answer this question, we can divide it into the following two main challenges:

Challenge 1: Create a framework that allows self-organizing systems to redefine their deployment. In some situations the desired behavior of a network cannot entirely be specified *a priori* to its deployment. Some design decisions must be made when the system is already operational. Delaying these decisions can be used to gain robustness of the system when facing operational changes (internal) or environmental changes (external). Traditionally, changing the system deployment implies that some human intervention is required in order to implement the new configuration. This can be very expensive, slow, tedious and perhaps even dangerous in hostile environments or hard-to-reach areas. Therefore, a self-organization framework is needed that can redefine a system's deployment at runtime.

Challenge 2: Find or improve strategies to coordinate collective behavior. Changing the operation of a single node in a network is rarely the solution to improve the overall behavior of the complete network. Under most circumstances changes across the network need to be coordinated and orchestrated in an "intelligent" manner. Moreover, a reconfiguration might increase the local performance of a single node, but decrease the global performance of the complete network. Cooperation between nodes is absolutely required in order to achieve top-level goals in the complete network. Therefore, a strategy must be found (or existing strategies improved) to coordinate behavior between nodes.

These two challenges are not independent of one another; both will need to be addressed in order to add self-organizing capabilities to a system. Furthermore, the self-organization process of the system should not be taking too much communication or computation overhead, especially when designing a low-powered embedded systems with limited resources. The problem underlying self-organization is a very challenging one by itself. The multi-agent variable assignment problem can be formulated as a DCOP, which is known to be NP-hard [107]. For this reason we should not design a system that searches the optimal solution, but we will have to make sure that the strategy used for achieving self-organization is properly balanced in terms of required resources and benefit for the "primary" system.

1.2. Contributions and Thesis Outline

In this thesis the aforementioned challenges will be addressed in the following manner.

Firstly, Challenge 1 is addressed in Chapter 2. We elaborate on the notion of self-organization, and introduce a framework for self-organizing systems. This framework describes a two-layered approach where all self-organization tasks are executed in a "secondary" layer, which runs independent of the primary system. This secondary layer gathers performance indicators and acts on the deployment of the system, optimizing its behavior. The framework will be put to use in a first use case of self-organizing state estimation in a greenhouse climate control system.

Secondly, Challenge 2 is addressed using two different strategies throughout this thesis:

1. **CoCoA**, a newly proposed (A)DCOP solver is developed in Chapter 3 (which is subsequently improved in Chapters 4 and 5),
2. a decentralized market based mechanism called **LP-RH** is used in Chapter 6.

CoCoA is an algorithm to solve ADCOPs, which coordinates collective behavior. Moreover, it is designed to do so using fewer system resources

than comparative algorithms. Through several experiments, the performance of the CoCoA algorithm is compared to state-of-the-art DCOP solvers. Finally, it is shown how CoCoA would solve the same greenhouse problem from Chapter 2.

In Chapter 4 a variant of the CoCoA algorithm is described that is specifically suited to deal with hard constraints. The new variant is applied to the Wireless Power Transfer (WPT) problem [97]. WPT is a technology where power is sent from a set of transmitters to receivers, to allow for remote wireless charging. Safety regulations apply such that electromagnetic radiation never exceeds safe human operation levels. In this particular problem hard constraints are particularly crucial to the safe operation of the network, and hence the extension of the CoCoA algorithm guarantees avoiding violating such hard constraints.

Chapter 5 describes a general extension of DCOP-solving algorithms to allow running of multiple existing algorithms, in order to cooperate and find a better solution together, than either one would on its own. The initialization of a DCOP algorithm turns out to be particularly important, and by using CoCoA for initialization, we see that the eventual solution cost decreases, and requires less time to converge.

Finally, not all types of problems can be cast (efficiently) as a DCOP, hence a different method for addressing Challenge 2 is proposed in Chapter 6. A coordination algorithm called Local Pricing Receding Horizon, is introduced in order to facilitate self-organization of the power dispatch problem in smart grids. This approach uses a more hierarchical approach than a DCOP-based solution, and is more fitting to the topology of the energy distribution network. The proposed solution uses a multi-agent based strategy to solve the problem using a combination of economic steering signals to incentivize the agents and local optimization to find an economic and sustainable power dispatch.

The thesis concludes in Chapter 7, where a brief overview of the contributions and results is provided, as well as an answer to the main research question and the two above-mentioned challenges.

2

Self Organizing State Estimation

2.1. Introduction

Self-organization and self-optimization are approaches to address the practical challenges of large-scale sensor and actuator (control) networks such as easy deployment, robustness and energy efficiency. In this chapter the first challenge of implementing self-organization is addressed: *to create a framework that allows a system to redefine its deployment*. The framework proposed in this chapter, aims to add self-organizing properties to a multi-agent system; specifically for self-organization of a sensor network. This framework will enable the system to cope with *changing system configurations* (i.e. adding and removing subsystem components) without re-programming the existing set-up (ease-of-deployment), to support a *mobile group* of subsystems observing particular areas (dynamic sensor management), to *adjust on environmental changes* affecting the communication resource (changing network capacities) and to *adapt to a variety of system goals* during operation depending on current needs and the monitored situation (multi-purpose).

In this chapter we propose a framework for self-organization, and apply it to distributed state estimation. Distributed signal processing components can be interchanged or reconfigured, depending on the requirements and system context. This system was earlier presented in [132], and is demonstrated in a greenhouse scenario, where the temperature distribution within the greenhouse is to be estimated. The main reason for selecting distributed state estimation is that state

This chapter is based on the article by C. J. van Leeuwen, J. Sijs, and Z. Papp†, *A reconfiguration framework for self-organizing distributed state estimators*, in Proc. FUSION (2013) [92].

estimation is used in a wide variety of applications, such as object tracking, traffic management and indoor climate control, to name a few. Yet, the proposed reconfiguration framework is applicable for any system in which state estimation is the key component for processing sensor measurements, as well for many more multi-agent systems in general. In what follows a generalized framework is presented for reconfigurable state estimating systems.

Notation and Preliminaries

Throughout this chapter, the following notation will be used: $\mathbb{R}$, $\mathbb{R}_{>0}$, $\mathbb{Z}$ and $\mathbb{Z}_{>0}$ define the set of real numbers, non-negative real numbers, integer numbers and non-negative integer numbers, respectively. For any $\mathcal{C} \subset \mathbb{R}$, let $\mathbb{Z}_{\mathcal{C}} := \mathbb{Z} \cap \mathcal{C}$. The notation 0 is used to denote either zero, the null-vector or the null-matrix of appropriate dimensions, while I_n denotes the $n \times n$ identity matrix. The transpose, inverse and determinant of a matrix $A \in \mathbb{R}^{n \times n}$ are denoted as A^T, A^{-1} and $|A|$, respectively. Further, $A^{\frac{1}{2}}$ denotes the Cholesky decomposition of a matrix $A^{n \times n}$ (if it exists). Given that a random vector $x \in \mathbb{R}^n$ is Gaussian distributed, denoted as $x \sim G(\mu, \Sigma)$, then $\mu \in \mathbb{R}^n$ and $\Sigma \in \mathbb{R}^{n \times n}$ are the *mean* and *covariance* of x.

2.2. Distributed State Estimation

Since this chapter focuses on reconfigurable distributed state estimation, this section starts off by considering a linear process model describing the state dynamics. In this context, several distributed solutions of the Kalman filter have been explored—see for example, solutions proposed in [35, 78, 122, 130] and the references therein. This section aims to derive a general framework, so that most of the currently available distributed Kalman filtering (DKF) solutions can be employed in the proposed reconfiguration scheme. To that extent, let us start with the state estimation problem, after which the generalized framework is introduced along with some illustrative estimation algorithms.

2.2.1. Problem Formulation

Let us consider a linear process that is observed by a sensor network with the following description. The networked system consists of N sensor nodes, in which a node $i \in \mathcal{N}$ is identified by a unique number within $\mathcal{N} := \mathbb{Z}_{[1,N]}$. The set $\mathcal{N}_i \subseteq \mathcal{N}$ is defined as the collection of *neighboring* nodes $j \in \mathcal{N}$ that exchange data with node i.

The dynamical process measured by each node $i \in \mathcal{N}$ is described with discrete-time process model, for some local sampling time $\tau_i \in \mathbb{R}_{>0}$ and

some k-th sample instant, i.e.,

$$x[\mathrm{k}_i] = A_{\tau_i} x[\mathrm{k}_i - 1] + w[\mathrm{k}_i - 1], \tag{2.1}$$

$$y_i[\mathrm{k}_i] = C_i x[\mathrm{k}_i] + v_i[\mathrm{k}_i], \tag{2.2}$$

where the state and local measurement are denoted as $x \in \mathbb{R}^n$ and $y_i \in \mathbb{R}^{m_i}$, respectively. Here, n and m_i correspond to the number of elements in the state and the measurements of i, respectively. The process noise $w \in \mathbb{R}^n$ and measurement noise $v_i \in \mathbb{R}^{m_i}$ follow the Gaussian distributions $w[\mathrm{k}_i] \sim G(0, Q_{\tau_i})$ and $v_i[\mathrm{k}_i] \sim G(0, V_i)$, for some $Q_{\tau_i} \in \mathbb{R}^{n \times n}$ and $V_i \in \mathbb{R}^{m_i \times m_i}$. A_{τ_i} and Q_{τ_i} are the variance parameters of the physical process model. Similarly, C_i and V_i are parameters of the sensor model of node i, defining how a measurement is obtained from the actual state. A method to compute the model parameters A_{τ_i} and Q_{τ_i} from a corresponding continuous-time process model $\dot{x} = Fx + w$ with model parameter F, yields

$$A_{\tau_i} := e^{F\tau_i} \quad \text{and} \quad Q_{\tau_i} := B_{\tau_i} \mathrm{cov}\left(w(t - \tau_i)\right) B_{\tau_i}^{\mathsf{T}}, \tag{2.3}$$

$$\text{with} \quad B_{\tau_i} := \int_0^{\tau_i} e^{F\eta} d\eta. \tag{2.4}$$

In distributed state estimation, the goal of a sensor network is to compute a local estimate $x_i \in \mathbb{R}^n$ of the global state x in each node i. Since the process model is linear and both noises are Gaussian distributed, it is appropriate to assume that the random variable $x_i[\mathrm{k}]$ is Gaussian distributed as well, i.e., $x_i[\mathrm{k}_i] \sim G(\hat{x}_i[\mathrm{k}_i], P_i[\mathrm{k}_i])$ for some *mean* $\hat{x}_i[\mathrm{k}_i] \in \mathbb{R}^n$ and *error covariance* $P_i[\mathrm{k}_i] \in \mathbb{R}^{n \times n}$. To that extent, each node i performs a local estimation algorithm for computing x_i based on its local measurement y_i and on the data shared by its neighboring nodes $j \in \mathcal{N}_i$.

Existing methods on DKF present solutions for computing x_i and determine what variables should be exchanged, at what time and with which nodes, e.g. [35, 78, 122, 130]. The goal of the reconfiguration framework is to reason about the design decisions made by these existing solutions and to select the most appropriate one for the current situation (depending on available communication and computational resources and on the estimation performance). This reasoning process will be addressed in a "management layer" encapsulating the variants for local Kalman filtering, which will be discussed in Section 2.3.3.

In order to be able to reason about different configurations, the alternatives for state estimation solutions should be described within a generalized framework, otherwise it is infeasible to compare the different alternatives objectively. Let us introduce such a general framework for distributed state estimation.

2.2.2. A General Framework for Distributed Kalman Filtering

The functional framework for computing the local estimate x_i is derived from existing DKF solutions. Typically, these solutions propose that each

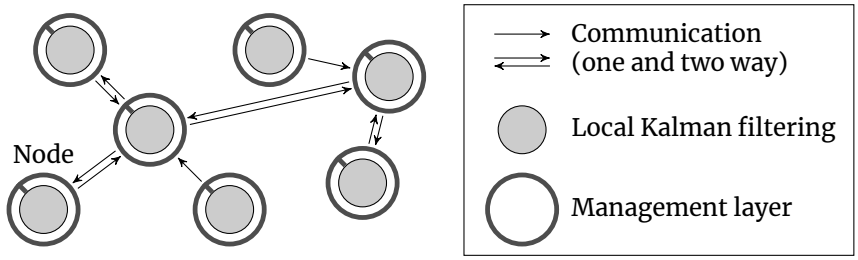

Figure 2.1: A network of Kalman filters supported by a management layer.

node i performs a Kalman filter (locally) based on its local measurement y_i and thereby, establishes an initial estimate $x_i \sim G(\hat{x}_i, P_i)$, e.g., in [35, 78, 122, 130] and some overview articles in [18, 129]. After that, different DKF solutions propose different types of variables exchanged between two neighboring nodes i and j, being:

- Local measurements: node i receives y_j for all $j \in \mathcal{N}_i$, which can be exploited for updating x_i via a Kalman filter.
- Local estimates: node i receives $x_j \sim G(\hat{x}_j, P_j)$ for all $j \in \mathcal{N}_i$, which can be exploited for updating x_i via various merging solutions, e.g. consensus or state fusion.

Based on the currently available DKF methods a generalized local estimation function is designed relying on the node's local measurement and on data received from neighboring nodes. The corresponding framework consisting of so called "functional primitives" is depicted in Figure 2.2. Each functional primitive of this framework, i.e., the Kalman filtering function f_{KF} and the merging function f_{ME}, is characterized by a specific algorithm, though it is not necessary to specify them prior to deployment. Instead, nodes are deployed with a number of suitable implementations for each functional primitive, from which a selection can be made during operation. Alternative implementations related to computing local estimation results of a single node $x_i \sim G(\hat{x}_i, P_i)$ and that of multiple nodes $x_{i^+} \sim G(\hat{x}_{i^+}, P_{i^+})$ are presented, next. Note that the subscript $i+$ indicates that the estimate or error covariance is based on a set of nodes including i and at least one other node.

Remark. Note that the measurement model $y_j[\mathsf{k}_i] = C_j x[\mathsf{k}_i] + v_j[\mathsf{k}_i]$ should be available to node i before $y_j[\mathsf{k}_i]$ can be exploited. Therefore, node j shares the local measurement (y_j, C_j, V_j), while if node j shares local estimates, it exchanges $(\hat{x}_j, P_j)$. This implies that the received data of Figure 2.2, yields:

- $\mathbb{Y}_j \subset \mathbb{R}^{m_j}, \mathbb{R}^{m_j \times n}, \mathbb{R}^{m_j \times m_j}$ is the collection of (y_j, C_j, V_j) received from neighboring nodes $j \in \mathcal{N}_i$. Note that $\mathbb{Y}_j$ could be empty, for example, when none of the nodes $j \in \mathcal{N}_i$ shares its local measurement;

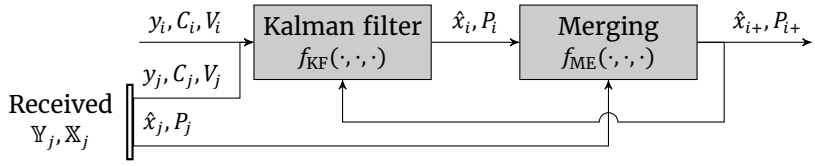

Figure 2.2: A framework of the generalized local estimation function, which consists of functional primitives, performed by each node i in the network, supported by measurements of another node j The inputs of the functions differ slightly per implemented function.

- $\mathbb{X}_j \subset \mathbb{R}^n, \mathbb{R}^{n\times n}$ is the collection of $(\hat{x}_j, P_j)$ received from neighboring nodes $j \in \mathcal{N}_i$. Similar as to $\mathbb{Y}_j$, also $\mathbb{X}_j$ can be an empty collection.

Let us continue with a detailed description of the Kalman filtering function in Figure 2.2. This functional primitive combines the local $y_i[\mathrm{k}_i]$ with the received measurements $y_j[\mathrm{k}_i]$ to update its cooperative local estimate $x_{i+}[\mathrm{k}_i - 1] \sim G(\hat{x}_{i+}[\mathrm{k}_i - 1], P_{i+}[\mathrm{k}_i - 1])$. Measurements are combined via the original Kalman filter or by the alternative Information filter, which was proposed in [35]. For this latter approach, measurements are rewritten into their *information form*, for some $z_j \in \mathbb{R}^n$ and $Z_j \in \mathbb{R}^{n\times n}$, i.e.,

$$z_j[\mathrm{k}_i] := C_j^\top V_j^{-1} y_j[\mathrm{k}_i] \quad \text{and} \quad Z_j[\mathrm{k}_i] := C_j^\top V_j^{-1} C_j. \tag{2.5}$$

The Information filter has similar results as the original Kalman filter but differs in computational demand. Furthermore, the information filter is more convenient when the amount of received measurements $(y_j, C_j, V_j) \in \mathbb{Y}_j$ varies at sample instants. Therefore, the Kalman filtering function in the framework of Figure 2.2 employs the Information filter. More precisely, $f_{\mathrm{KF}}(\cdot,\cdot,\cdot)$ is a function with three inputs and two outputs according to the following characterization:

$$(\hat{x}_i[\mathrm{k}_i], P_i[\mathrm{k}_i]) := f_{\mathrm{KF}}(\hat{x}_{i+}[\mathrm{k}_i - 1], P_{i+}[\mathrm{k}_i - 1], \mathbb{Y}_j[\mathrm{k}_i])$$

$$P_i[\mathrm{k}_i] = \Big(M_i^{-1} + Z_i[\mathrm{k}_i] + \sum_{(y_j, C_j, V_j)\in\mathbb{Y}_j[\mathrm{k}_i]} Z_j[\mathrm{k}_i]\Big)^{-1},$$

$$\hat{x}_i[\mathrm{k}_i] = P_i[\mathrm{k}_i]\Big(M_i^{-1} A_{\tau_i} \hat{x}_{i+}[\mathrm{k}_i - 1] + z_i[\mathrm{k}_i] + \sum_{(y_j, C_j, V_j)\in\mathbb{Y}_j[\mathrm{k}_i]} z_j[\mathrm{k}_i]\Big),$$

$$M_i = A_{\tau_i} P_{i+}[\mathrm{k}_i - 1] A_{\tau_i}^\top + Q_{\tau_i}.$$

The merging function of Figure 2.2, introduced as $f_{\mathrm{ME}}(\cdot,\cdot,\cdot)$, merges the local estimate $x_i[\mathrm{k}_i]$ with the received estimation variables $x_j \sim (\hat{x}_j, P_j) \in \mathbb{X}_j[\mathrm{k}_i]$ into a new estimate $x_{i+}[\mathrm{k}_i] \sim G(\hat{x}_{i+}[\mathrm{k}_i], P_{i+}[\mathrm{k}_i])$. Typically, the functional primitive $f_{\mathrm{ME}}(\cdot,\cdot,\cdot)$ is based on solutions for merging two state estimation results x_i and x_j, yielding a recursive behavior to merge

all received estimation results. This means that the merging function $f_{\mathrm{ME}}(\cdot,\cdot,\cdot)$ performed by a node i has the following definition:

$$
\begin{aligned}
&(\hat{x}_{i^+}[\mathrm{k}_i], P_{i^+}[\mathrm{k}_i]) := f_{\mathrm{ME}}\left(\hat{x}_i[\mathrm{k}_i], P_i[\mathrm{k}_i], \mathbb{X}_j[\mathrm{k}_i]\right)\\
&\qquad \text{for each estimate } (\hat{x}_j[\mathrm{k}_i], P_j[\mathrm{k}_i]) \in \mathbb{X}[\mathrm{k}_i] \text{ do}\\
&\qquad\quad (\hat{x}_i[\mathrm{k}_i], P_i[\mathrm{k}_i]) = \Omega\left(\hat{x}_i[\mathrm{k}_i], P_i[\mathrm{k}_i], \hat{x}_j[\mathrm{k}_i], P_j \mathrm{k}_i]\right),\\
&\qquad \text{end}\\
&\qquad \hat{x}_{i^+}[\mathrm{k}_i] = \hat{x}_i[\mathrm{k}_i], \quad P_{i^+}[\mathrm{k}_i] = P_i[\mathrm{k}_i],
\end{aligned}
$$

where $\Omega(\cdot,\cdot,\cdot,\cdot)$ is the inner merging function for merging the local estimation results x_i with received estimation results x_j. Let us present three examples from the literature of such inner merging functions $\Omega(\cdot,\cdot,\cdot,\cdot)$ one synchronization and two fusion approaches.

Synchronization Merging

Recent DKF solutions, e.g., [78, 122], adopt a synchronization approach to characterize $f_{\mathrm{ME}}(\cdot,\cdot,\cdot)$. Such an approach stems from the idea of synchronizing different internal clocks in the network. Typically, synchronization is employed on the estimated means, for some scalar weight $\omega_{ij} \in \mathbb{R}_{>0}$, yielding the following inner-merging function:

$$
\begin{aligned}
&\text{SY: } (\hat{x}_i[\mathrm{k}_i], P_i[\mathrm{k}_i]) = \Omega\left(\hat{x}_i[\mathrm{k}_i], P_i[\mathrm{k}_i], \hat{x}_j[\mathrm{k}_i], P_j[\mathrm{k}_i]\right)\\
&\qquad \hat{x}_i[\mathrm{k}_i] = (1-\omega_{ij})\hat{x}_i[\mathrm{k}_i] + \omega_{ij}\hat{x}_j^{-1}[\mathrm{k}_i],\\
&\qquad P_i[\mathrm{k}_i] = P_i[\mathrm{k}_i].
\end{aligned}
$$

Solutions for establishing the weights ω_{ij} have been presented in literature extensively, for example in [137, 144].

Merging solutions that take the combination of error covariances into account in addition to a combined estimated mean are known as fusion solutions. An optimal fusion method was presented in [9], though it requires that correlation of the two prior estimates is available. In the considered sensor networks one cannot impose such a requirement, as it amounts to keeping track of shared data across the entire network. Alternative fusion methods that can cope with an unknown correlation are covariance intersection (CI) and ellipsoidal intersection (EI), as proposed in [70] and [131], respectively. We will discuss these solutions below.

Covariance Intersection

In CI the fusion function is characterized as a convex combination of the two prior estimates x_i and x_j, for some scalar weight defined as $\omega_{ij} = \mathrm{tr}(P_i)\left(\mathrm{tr}(P_i) + \mathrm{tr}(P_j)\right)^{-1}$, i.e.,

CI: $(\hat{x}_i[\mathsf{k}_i], P_i[\mathsf{k}_i]) = \Omega\left(\hat{x}_i[\mathsf{k}_i], P_i[\mathsf{k}_i], \hat{x}_j[\mathsf{k}_i], P_j[\mathsf{k}_i]\right)$

$$\begin{aligned}\Sigma_i &= \left((1-\omega_{ij})P_i^{-1}[\mathsf{k}_i] + \omega_{ij}P_j^{-1}[\mathsf{k}_i]\right)^{-1},\\ \hat{x}_i[\mathsf{k}_i] &= \Sigma_i\left((1-\omega_{ij})P_i^{-1}[\mathsf{k}_i]\hat{x}_i[\mathsf{k}_i] + \omega_{ij}P_j^{-1}[\mathsf{k}_i]\hat{x}_j^{-1}[\mathsf{k}_i]\right),\\ P_i[\mathsf{k}_i] &= \Sigma_i.\end{aligned}$$

Ellipsoidal Intersection

The fusion method EI results in a "smaller" error covariance compared to CI, as the fusion result is not a convex combination of prior estimate. Instead, EI finds an explicit expression of the (unknown) correlation before merging independent parts of x_i and x_j via algebraic fusion formulas. To that extent, the (unknown) correlation is characterized by a *mutual covariance* $\Gamma_{ij} \in \mathbb{R}^{n\times n}$ and a *mutual mean* $\gamma_{ij} \in \mathbb{R}^n$, yielding the following inner function $\Omega(\cdot,\cdot,\cdot,\cdot)$:

EI: $(\hat{x}_i[\mathsf{k}_i], P_i[\mathsf{k}_i]) = \Omega\left(\hat{x}_i[\mathsf{k}_i], P_i[\mathsf{k}_i], \hat{x}_j[\mathsf{k}_i], P_j[\mathsf{k}_i]\right)$

$$\begin{aligned}\Sigma_i &= (P_i^{-1}[\mathsf{k}_i] + P_j^{-1}[\mathsf{k}_i] - \Gamma_{ij}^{-1})^{-1},\\ \hat{x}_i[\mathsf{k}_i] &= \Sigma_i\left(P_i^{-1}[\mathsf{k}_i]\hat{x}_i[\mathsf{k}_i] + P_j^{-1}[\mathsf{k}_i]\hat{x}_j^{-1}[\mathsf{k}_i] - \Gamma_{ij}^{-1}\gamma_{ij}\right),\\ P_i[\mathsf{k}_i] &= \Sigma_i.\end{aligned}$$

The mutual mean γ_{ij} and mutual covariance Γ_{ij} are found by a singular value decomposition, which is denoted as $[S, D, S^{-1}] = \mathrm{svd}(\Sigma)$ for a positive definite $\Sigma \in \mathbb{R}^{n\times n}$, a diagonal $D \in \mathbb{R}^{n\times n}$ and a rotation matrix $S \in \mathbb{R}^{n\times n}$. As such, let us introduce the matrices $D_i, D_j, S_i, S_j \in \mathbb{R}^{n\times n}$ via the singular value decompositions

$$[S_i, D_i, S_i^{-1}] = \mathrm{svd}\left(P_i[\mathsf{k}_i]\right), \tag{2.6}$$

$$[S_j, D_j, S_j^{-1}] = \mathrm{svd}\left(D_i^{-\frac{1}{2}} S_i^{-1} P_j[\mathsf{k}_i] S_i D_i^{-\frac{1}{2}}\right). \tag{2.7}$$

Then, an expression of γ_{ij} and Γ_{ij}, for some $\varsigma \in \mathbb{R}_{>0}$ and $\{A\}_{qr} \in \mathbb{R}$ denoting the element of a matrix A on the q-th row and r-th column, yields

$$D_{\Gamma_{ij}} = \mathrm{diag}\left(\max[1, \{D_j\}_{11}], \cdots, \max[1, \{D_j\}_{nn}]\right), \tag{2.8}$$

$$\Gamma_{ij} = S_i D_i^{\frac{1}{2}} S_j D_{\Gamma_{ij}} S_j^{-1} D_i^{\frac{1}{2}} S_i^{-1}, \tag{2.9}$$

$$\begin{aligned}\gamma_{ij} = &\left(P_i^{-1} + P_j^{-1} - 2\Gamma_{ij}^{-1} + 2\varsigma I_n\right)^{-1} \times\\ &\left(\left(P_j^{-1} - \Gamma_{ij}^{-1} + \varsigma I_n\right)\hat{x}_i + \left(P_i^{-1} - \Gamma_{ij}^{-1} + \varsigma I_n\right)\hat{x}_j\right).\end{aligned} \tag{2.10}$$

A suitable value of ς follows: $\varsigma = 0$ if $|1 - \{D_j\}_{qq}| > 10\epsilon$, for all $q \in \mathbb{Z}_{[1,n]}$ and some $\epsilon \in \mathbb{R}_{>0}$, while $\varsigma = \epsilon$ otherwise. The value 10 is used in this work to define that the value is much larger, and combined with a design parameter ϵ support a numerically stable result.

This completes the description of the estimation function depicted in Figure 2.2. Let us continue by explaining the reconfiguration parameters that can be tuned for the considered estimation function.

2.2.3. Tuning the Functional Primitives

The estimation function illustrated in Figure 2.2 consists of two functional primitives. A node i combines received measurements $\mathbb{Y}_j$ via the Kalman filtering functional primitive f_{KF}, while received estimates $\mathbb{X}_j$ are merged in the functional primitive f_{ME} via either SY (synchronization), CI (covariance intersection) or EI (ellipsoidal intersection). There are several parameters and structural changes that the management layer can select, so that a suitable estimation function is constructed in line with the current situation and available resources. The different options for tuning a functional primitive are addressed in this section.

- Sampling time τ_i: The sampling time of both primitives f_{KF} and f_{ME} can be tuned accordingly. At every sample time the Kalman filter executes and provides a new local estimate. Then, the merging function is triggered, merging any received estimates into a regional estimation. Lowering the sampling time τ_i implies that estimation accuracy, computational demand and data exchange will be decreased, i.e., saving communication and computational resources and thereby saving energy.
- Shared data $\mathbb{Y}_j$ and $\mathbb{X}_j$: The selection of which variables are shared, i.e., (y_i, C_i, V_i) and/or $(\hat{x}_i, P_i)$, can change in time. Exchanging both improves estimation results throughout the network but requires communication resources. Decreasing the usage of this resource can be done by exchanging either the local measurement or the local estimation result, which would yield a decrease in the estimation accuracy. Additionally, the frequency of exchanging data can be influenced by changing the communication frequency parameter f_i.
- Implemented algorithm: There is only one implementation of the functional primitive f_{KF}, which is the Information filter. Yet, for the merging primitive f_{ME} one has the option between three alternative implementations—only one of which can be active in a node at any given time:
  1. SY: The inner-function $\Omega(\cdot,\cdot,\cdot,\cdot)$ is characterized by the synchronization approach. The required computational power for this alternative is low, though it would result in a poor estimation accuracy as well;

2. CI: The inner-function $\Omega(\cdot,\cdot,\cdot,\cdot)$ is characterized by the covariance intersection approach [70]. The required computational power for this alternative is moderate and would result in a moderate estimation accuracy;
3. EI: The inner-function $\Omega(\cdot,\cdot,\cdot,\cdot)$ is characterized by the ellipsoidal intersection approach [131]. The required computational power for this alternative is high and would result in a high estimation accuracy;

The above options on changeable parameters and alternative functional primitive are exploited by the management layer for finding a match between the desired estimation quality (accuracy) and the required resources.

2.3. Reconfiguration Framework

The following reconfiguration framework provides the tools needed for a self-organizing state estimation sensor node. We use concepts from model-based design in runtime to aid the reconfiguration process by providing information about potential configuration states. A reasoner can then compare different states, and implement the most beneficial one.

2.3.1. Model-Based Design Concepts

The design challenge for any embedded system is to realize the required functionalities (in this case state estimation) on a given hardware platform while satisfying a set of nonfunctional requirements, such as response times, dependability, power efficiency, etc. Model-based system design has been proven to be a successful methodology for supporting the system design process [73]. Model-based methodologies use multiple models to capture the relevant properties of the design. These models can then be used for various purposes, such as automatic code generation, design optimization, system evolution, etc. [72] Crucial for the design process are the interactions between the different models.

Two fundamental models of the design are (i) the task model, capturing the required functionalities of the employed signal processing method, and (ii) the physical model capturing the hardware configuration of the implementation. In Figure 2.3 the task model is represented as directed a graph: the signal processing components are represented by the vertices of the graph, while their data exchange or precedence relations are represented by the edges. Both the tasks and the interactions are characterized by a set of properties, which typically reflect non-functional requirements/properties. The tasks run on a connected set of processors, represented by the physical model of the system. The components in the physical model are the computing nodes and the communication links.

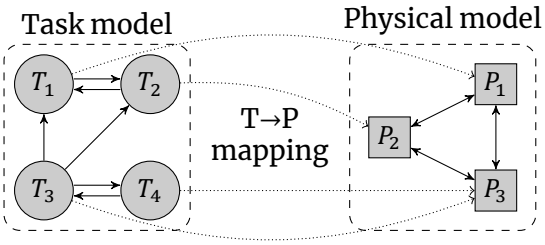

Figure 2.3: The task and physical model represent the different functional components and the processing components where they can run, respectively. A mapping between them defines which functions run where.

It should be mentioned that in the signal processing context the task graph is designed in two phases as shown in Figure 2.4: first the functional primitives are connected to form the signal flow graph satisfying the functional requirements and design constraints. Then, the task network should be created via clustering the elements of the network of functional primitives to tasks considering computation, communication and temporal requirements. This is not a linear process and there is a strong dependency on the physical configuration. Moreover, the search and optimization in the design space make this an iterative process, which requires interactions between hardware, software and signal processing architecture designs.

The design process involves finding a particular mapping that defines the assignment of a task T_q to a processor P_i, i.e., it determines which task runs on which node. Obviously the memory and execution time requirements define constraints when assigning the tasks to nodes. Further, data exchange between tasks makes the assignment problem more challenging in distributed configurations, as a task assignment also defines the use of communication links c_{ij} - and the communication links have limited capacities. The design process results in a sequence of decisions, which lead to a feasible system design. Traditionally this design process is "off-line", i.e., it is completed before the implementation and deployment of the system itself. The task model, the hardware configuration and their characteristics are assumed to be known during this design time and the design uncertainties are assumed to be low.

2.3.2. Runtime Reconfiguration

These are overly optimistic assumptions for large-scale sensor and actuator (control) networks: in many cases they are deployed in "hostile" environments, where component failures and dynamically changing configurations manifest themselves as common operational events. Expressed in the concepts of Figure 2.3, conceptually the runtime reconfiguration is carried out via changing the task graph (i.e. selecting a

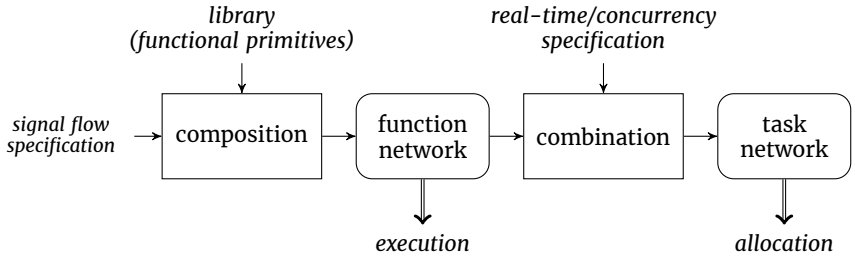

Figure 2.4: The design flow of the signal processing system; the function network influences the design choices of the task network.

different signal processing scheme, changing certain parameters of the functional primitives, etc.) or re-mapping the task graph to the physical model (i.e. changing the task assignment with the consequential change in the communication topology).

As Figure 2.1 already indicates, our goal is to realize the reconfiguration functionality for distributed state estimation to improve robustness and scalability. The functional scheme of the reconfiguration is shown in Figure 2.5. The primary functionality (in our case the state estimation) is realized by the task network (via the invocation of associated function primitives). The task network is built during initialization time according to (off-line) design specification. The reconfiguration runs parallel to the primary data stream: based on the execution status (e.g. quality of the results generated, the conditions of the hardware resources, the availability of the communication links, etc.) the reconfiguration functionality makes decisions about the configuration, its parameterization and resource usage in order to satisfy the given requirements and constraints. The reconfiguration is event driven, triggered by changes in the execution context or the changing (user) requirements and constraints. The reconfiguration may act on the software side (e.g. selecting a different algorithm to implement a particular functional primitive, changing task allocation, etc.) or on the hardware side (e.g. adjusting transmission power, suspending/awaking components, etc.).

The following characteristics of the proposed scheme should be emphasized:

- Every time instant the function/task network is a snapshot of the possible variants and mappings. Alternatives may not be explicitly enumerated but can be the result of a reasoning process.
- The scheme explicitly supports the separation of concerns principle; reconfiguration mechanisms can be designed and implemented relatively independently.
- The reconfiguration can be a resource demanding activity. Therefore, the scheme allows for tuning the "intelligence level" of

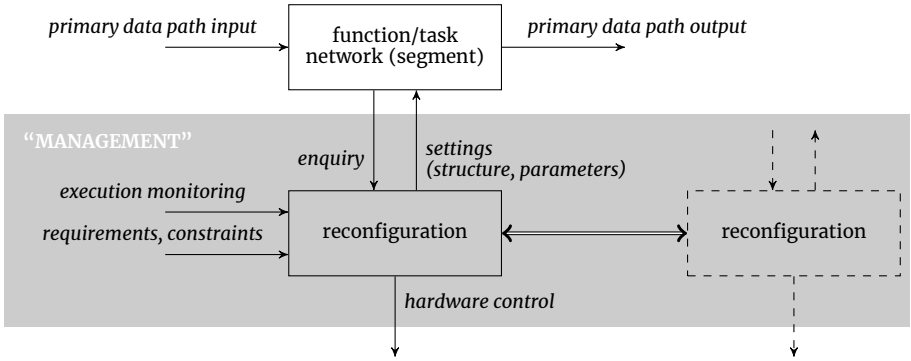

Figure 2.5: A schematic representation of the reconfiguration process. The primary data path is updated by a secondary "management" layer by updating the structure or parameters. The nodes' reconfiguration components can coordinate with one another to find configuration that are mutually beneficial.

the reconfiguration depending on the performance of the hardware configuration—virtually leaving the signal processing aspect uninfluenced.

- There are low-overhead implementations available for dynamic data-flow graph based signal processing (e.g. [112]). Consequently, the influence of the reconfiguration on the signal processing performance can be kept low. The interfacing between the data-flow graph and the "management" side is usually implemented by a simple API or message passing mechanism.

- The scheme is not specific to the distributed state estimation problem, but other signal processing tasks including control or other multi-agent based tasks can be mapped into this framework as well.

- The scheme is applicable to reconfiguration on "various levels of granularity": task, node and system levels, i.e. the reconfiguration strategy scales from fine grade distributed to a centralized one. Needless to say distributed reconfiguration may need cooperation among the reconfiguration functionalities.

- From execution point of view the reconfiguration functionalities should be included in the task graph (as one or more cooperating tasks) and their resource demand should be accounted for.

In the following the "management" side will be further detailed. The representation of the configuration and the design knowledge as well as the associated reasoning mechanisms are the key elements on the management layer, thus in the following these aspects will be emphasized.

2.3.3. Knowledge Representation and Reasoning

For the management layer as seen in Figure 2.1, which reconfigures the core tasks' functionality, a three-step strategy is implemented. The first step in reconfiguration is the *monitoring* step, in which the current status quo is observed, in order to reflect the system's own health/performance and the state of the embedding environment. The second step is the *reasoning* step in which the observed characteristics are analyzed, decided whether reconfiguration is even required, and if it is required how to do so. The final step is the *actuation* step which performs the decisions made in step two, thereby completing the reconfiguration process.

The reasoning step is one of special interest, because it is mainly here where the intelligence about the configuration is represented in the system. Because the reasoning is only a single step within the reconfiguration process and it always operates with a predefined interface, it can be separated from the rest of the system which makes it relatively easy to implement any kind of reasoning module.

In order to reconfigure the tasks' core functionality, the reconfigurator requires knowledge and a method for reasoning about the various configurations. These two are intrinsically connected since the form of the knowledge representation depends on the method of reasoning. A case-based reasoner would require knowledge in terms of different complete configurations, whereas a utility reasoner would require a utility function.

From the many forms of reasoning and knowledge representations, for this article a first order logic (FOL) reasoner will be used. The corresponding knowledge representation exists of a rule base existing of atoms representing the configurable parameters, and constraints and conditions in the form of logical compound statements to determine what parameter choices are valid and which are not. The problem statement of reconfiguration is now reduced to finding the values for the atoms that satisfy the statements (effectively a search). Now, problem solving methods can be used for solving FOL problems such backtracking and the Selective Linear Definite clause resolution which is proven to be sound and complete [6, 10, 135].

FOL has been used to describe reconfiguration knowledge in the past as well [37, 133]. An important reason for this is that FOL reasoners have proven to be very expressive in that they can describe both conditions and constraints, perform boolean, numerical and symbolical operations and therefore can reconfigure either by optimizing parameters or starting/stopping components [15].

Using a FOL reasoner however, also means that *a priori* to deploying the system, all constraints and conditions must be determined because a rule base is required by the reasoner. This requires some design time knowledge which can be expert or empirical knowledge from experiments. For scenarios in which multiple methods are applicable, a preferential order

must be determined. More complex bias-free reasoners, which are capable of taking into account the effect of one's choice on neighboring nodes, will be looked into in the next chapter.

2.4. Case Study

In the case study, the proposed framework is applied to a scenario in which the temperature distribution of a greenhouse has to be monitored. In the scenario there are N nodes all equipped with a temperature sensor, and a communication interface for communicating wirelessly. Each node had to compute an estimate the global greenhouse temperature distribution based on a local measurement and communication with the other nodes. The implementation of the experiment was in the form of a discrete event simulation [29].

Using off-line measurements from a real physical small scale greenhouse setup, the system had access to realistic data, but over multiple runs, the sensors would obtain the same input. In the experiment, the six simulated nodes are equipped with simple temperature sensors, which would base their measurements on the values from this database.

The nodes were constrained in the bandwidth of the communication with other nodes, the amount of integer and floating-point operations per second (IOPS/FLOPS) and a limited energy supply. In the ideal situation these constraints should imply that the node should initially use the most "expensive" method that would be computationally feasible, and under decreasing battery capacity would decrease its effort, and in the end use the "cheapest" method. At the same time the system would always employ methods that satisfy the current bandwidth conditions.

In the scenario implemented for this experiment, there are six nodes and in one of the nodes the monitor finds the battery level to drop below a certain threshold, such that reconfiguration is desired. In the experimental setup the system can change the following variables:

1. the algorithm used for the functional primitive f_{ME}, which can be either EI, CI or SY,
2. the sampling rate at which to sample the sensor τ_i,
3. the communication rate to send out information to neighboring nodes f_i,
4. the *sleep* time of the radio, which controls the rate of incoming messages from neighboring nodes f_i'.

In the initial configuration the system uses EI in the merging functional primitive, samples its sensor once every 60 seconds and broadcasts its state estimation results every 90 seconds. Furthermore, it fires the monitoring cycle every 150 seconds and shuts down the radio for 5 seconds every other 30 seconds (thereby using a pattern of 5 seconds

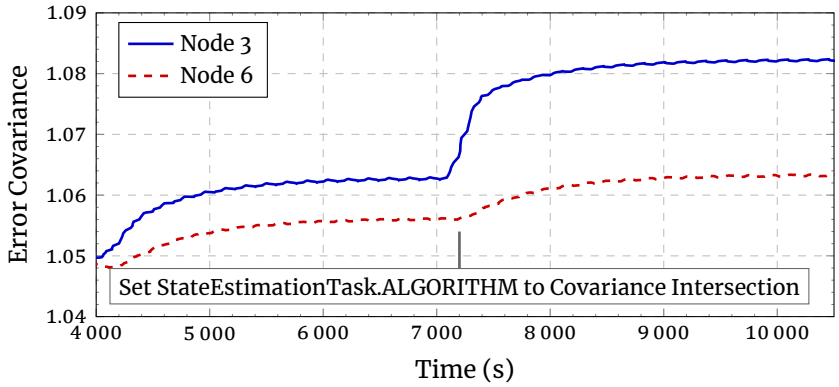

Figure 2.6: The error covariance trace of nodes 3 and 6, when changing the merging algorithm in node 3 from Ellipsoidal Intersection to Covariance Intersection.

sleep, followed by 25 seconds receiving.) Exchanged variables are automatically chosen, so that if a node uses EI or CI it broadcasts $(\hat{x}_i, P_i, \mathsf{k}_i, i)$ whereas if it uses SY it broadcasts $(\hat{x}_i, \mathsf{k}_i, i)$.

The reasoner implemented relies on a Prolog based FOL interpreter[1]. Choosing this implementation has a couple of additional advantages. First and foremost, it has been used for a long time by experts for formally defining expert knowledge [125], and is very expressive, but also flexible in the type of rules that could be used to describe the knowledge for the application. Secondly, Prolog is a well-known logical programming language and therefore the framework can easily be programmed using existing syntax and methods. Finally, the Prolog implementation can use an external rule base, separating the reasoning rule base from the implementation itself. This results in having the reconfiguration behavior defined separately and independently of the rest of the system.

The knowledge of the reconfiguration reasoner must be coded *a priori* to the deployment of the system. The constraints and conditions of the rules, and the order in which the different reconfiguration actions will be performed must be determined. This does not mean that the system will reconfigure in any specific order, because the operating conditions are unknown beforehand, and therefore the constraints and conditions determine the configuration at runtime.

The FOL rule base allows for a wide range of rule types. The rule base created for the case study contained rules of varying complexity, from very simple rules considering the choice of parameters to more complex rules combining different conditions of system properties into a desired reconfiguration action. Example statements would formally be implemented in Prolog as following:

[1] http://www.gnu.org/software/gnuprologjava/

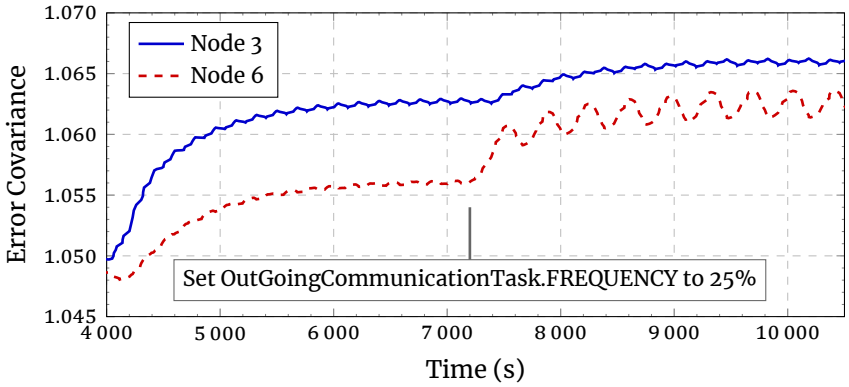

Figure 2.7: The error covariance trace of nodes 3 and 6, when decreasing the outgoing communicating frequency in node 3 to 25%.

Example Prolog reconfiguration statements

```
idealSamplingFrequency(0.2).

batteryCritical(State) :-
  getBatteryLife(State, ~BatteryLife),
  minBattery(Threshold),
  BatteryLife < Threshold.

action(lowerCommunicationFrequency, ~State, ~Reason) :-
  batteryLow(State),
  \+ communicationFrequencyLow(State),
  Reason = "The battery is low and the communication
  frequency is high enough.".

suggest(NewSetting, lowerSamplingFrequency, State) :-
  samplingFrequencyLow(State),
  getSamplingFrequency(State, SamplingFrequency),
  NewSetting is SamplingFrequency * 0.75.
```

Whenever a re-parametrization is in order, the reasoner also has to provide the system with the new parameters. The last statement in the above example shows an example of how the choice of parameters might be implemented. Note that this way of choosing the parameters means that the system will act like a feedback loop. In order to find an optimal new value, another type of reasoner will have to be implemented such as a utility based reasoner.

In order to determine the ordering of the different type of configurations and to create the related rule base, decision were made upfront by the system designer by experimentally trying out different configurations. In these experiments the six nodes would operate, and after two hours

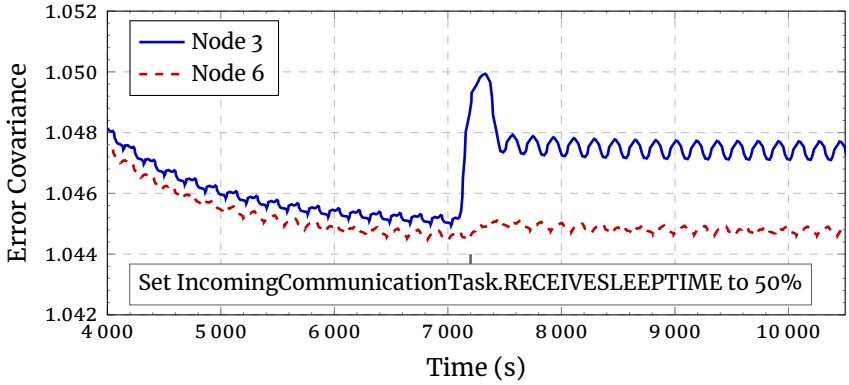

Figure 2.8: The error covariance trace of nodes 3 and 6, when changing the radio sleep time in of node 3 to 50%, effectively decreasing the incoming communication frequency.

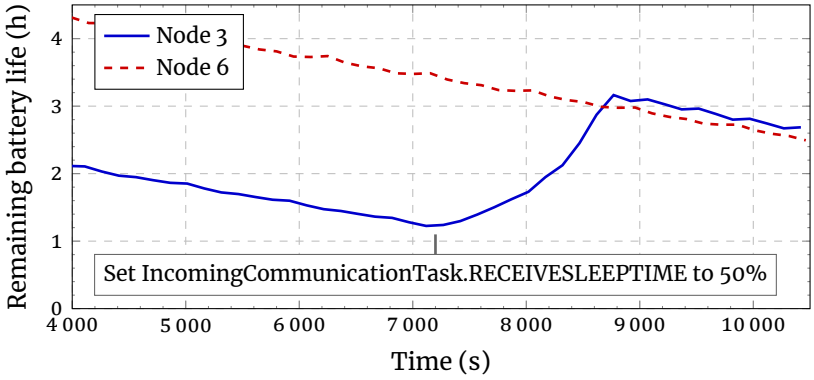

Figure 2.9: The estimated remaining battery life of nodes 3 and 6, when changing the radio sleep time. Here, the reduction of power consumption of the radio is very noticeable.

of simulation time, the configuration of Node 3 would change. For the results of these experiments, see Figures 2.6 through 2.8. In Figure 2.6 it can be seen that changing the fusion method has a significant impact on the error covariance of the state estimation in reconfigured node, but also in the node that stays the same. The same holds for a change in the outgoing communication frequency f_i, of which the results can be seen in Figure 2.7. The sleep time of the radio has relatively the smallest impact, and seems the best option for the first reconfiguration action. This means the node will receive fewer messages from the other nodes and therefore the error covariance will go up, but much less than with any of the other options. Using this information, a rule base can be created.

When running the resulting system with the created rule base in a simulated experiment, the reconfiguration reasoner will deduce that a differ-

ent sleep time is most desirable. In Figure 2.9 it can be seen that the corresponding remaining battery lifetime goes up significantly.

2.5. Conclusions

In this chapter a framework was proposed to implement self organization in a sensor network. We defined a framework for distributed state estimation using a collection of functional primitives that can be redeployed, reconnected and parameterized by a management layer. This introduces the possibility to reason about the configuration of how to do state estimation under a variety of circumstances and user requirements.

In the greenhouse case study we have shown that the proposed framework is capable of improving the battery lifetime of a sensor by runtime reconfiguring the state estimation method, while maintaining a good performance level. This is done by modifying the outgoing frequency, and the sleep period parameters for the communication strategy, and thereby having minimal impact on the state estimation error. However, in this chapter we only considered the state estimation problem, whereas the framework allows for reconfiguring all kinds of distributed reasoning tasks. In the later chapters we will look at different problem domains, e.g. wireless power transfer in Chapter 4 and smart grids in Chapter 6.

The modular framework allows for easy implementation of a different reasoning engine, but also for portability for different state estimation tasks. In the current reasoner, a first-order logic core uses an *a priori* determined set of ordered rules from the rule base. This is a drawback of the logic-based reasoner, and in a real implementation it would be recommendable to use an unbiased context-free configuration space exploring reasoner. An example of this could be a genetic algorithm adapted to the reconfiguration scenario, or a learning algorithm. Another drawback of the current reasoner is the lack of coordination among nodes; although this could be implemented in a logic based reasoner, this has not been done in this chapter. To address this shortcoming, in the next chapter a DCOP-based method for orchestrating configurations among different nodes is proposed, for coordinated decision-making such that all nodes benefit.

3

CoCoA: an (A)DCOP Solver

3.1. Introduction

Self-organization is a property of a group of systems that can adapt their behavior to changing environments, or user requirements. In Chapter 1, we introduced the notion of self-organization either through emergent behavior, or through explicit cooperation. In Chapter 2, we introduced a framework for reconfiguring a system's deployment, but when we reason about *how* the system should be configured, we can either make decisions locally (thus using the emergent behavior approach), or use a distributed decision-making method. Local reasoning about an agent's behavior is definitely feasible, as we have shown in the previous chapter. It is however, much more challenging to adapt while keeping in mind the effect of a single agent's decisions on the group as a whole. In order to make sure the agents act as a group, decisions need to be coordinated or orchestrated. However, the traditional approach using a central coordinator is not desirable, since it leads to a *single point of failure* and does not scale well. Without a central coordinator or hierarchy, the agents need to collaborate as peers in order to make sure the group objective is maintained.

Distributed Constraint Optimization Problems (DCOP) is a class of optimization problems in which discrete variables are controlled by distributed agents and the optimization function itself operates over the complete set of variables [59]. This class of problems allows to formalize the type of decisions a group of agents needs to make in a coordinated fashion. By definition of DCOP, the involved agents are part of a team and need to cooperate in order to perform well on the global task. Usually in DCOPs, cooperation between agents is achieved by passing messages

This chapter is based on the article by C. J. van Leeuwen and P. Pawełczak, *CoCoA: A non-iterative approach to a local search (A)DCOP solver*, in Proc. AAAI (2017) [89].

from one agent to another. DCOPs are encountered in many fields such as in wireless LAN channel allocation [150], coordination of mobile sensing teams [148] or coordination of tasks [38].

A number of complete algorithms have been proposed to find the optimal solution of a DCOP, amongst which are ADOPT [108], DPOP [116], NCBB [19] and Asynchronous Forward Bounding [52]. However, DCOP problems are NP-hard [107], so the effort to find the optimal solution becomes intractable for increasingly large-scale problems. Therefore, incomplete DCOP algorithms trade a distance from the optimal solution for convergence speed and are thus more suited for large-scale problems. Examples of incomplete DCOP solvers are DBA [152], DSA [45], max-sum [38], MGM and MGM-2 [100]. Also, a combination of an incomplete solver, with a complete solver is proposed in a hybrid strategy called HS-CAI [20].

Recently, an extension on the DCOP framework has been described in which agents may value a set of constrained assignments differently. In these Asymmetric DCOPs (ADCOP) constraints have different costs for their agents. The ACLS and GCA-MGM algorithms [56, 57] have been proposed to enable solving this class of problems. More recently, it has also been shown that the max-sum_ADVP algorithm can solve ADCOPs [158]. In solving ADCOPs there is usually a trade-off in privacy versus optimality [30].

Most existing DCOP algorithms use an iterative approach, which requires many rounds of message passing during the optimization process, and thus require a relatively large communication overhead. In this chapter we present a case study in which this communication overhead is especially important: to find an optimal configuration of a sensor network, tasked with monitoring the cargo of a shipping container for an extensive period of time. Since in sensor networks communication between nodes has the largest influence on the battery lifetime, we need to minimize the communication between nodes during the optimization process, and require an algorithm that is guaranteed to converge to a solution as quickly as possible.

We will introduce a new (A)DCOP algorithm, denoted as Cooperative Constraint Approximation (CoCoA), which uses a non-iterative, semi-greedy approach with a single step lookahead (SSLA). We show that it finds high quality solutions much faster than other (A)DCOP solvers, using far fewer system resources. Experimentally, we show that in some cases this leads to a reduction of up to two orders of magnitude in the number of transmitted messages and cost function evaluations, thus leading to superior running times.

3.2. DCOP: Problem Statement and Challenges

DCOPs are defined as a tuple $\mathcal{T} = \langle \mathcal{A}, \mathcal{X}, \mathcal{D}, \mathcal{R} \rangle$ in which $\mathcal{A}$ is a finite set of agents $\{A_1, A_2, \dots, A_n\}$ and $\mathcal{X}$ is a finite set of variables $\{X_1, X_2, \dots, X_n\}$ with finite discrete domains $\{D_1, D_2, \dots, D_n\}$ from $\mathcal{D}$ such that $X_i \in D_i$. Each agent A_i is assigned one variable X_i, and therefore $|\mathcal{A}| = |\mathcal{X}| = |\mathcal{D}|$. Then, $\mathcal{R}$ is a set of relations (constraints) in which each constraint $C \in \mathcal{R}$ defines a non-negative cost. For DCOPs such costs are defined for every possible value assignment of a set of variables $C\colon D_{i_1} \times D_{i_2} \times \dots \times D_{i_k} \to \mathbb{R}_{\geq 0}$, while for ADCOPs each constraint defines a set of costs for each involved variable, i.e. $C\colon D_{i_1} \times D_{i_2} \times \dots \times D_{i_k} \to \mathbb{R}^k_{\geq 0}$. Having all definitions of $\mathcal{T}$, in (A)DCOPs the goal of the agents is to minimize the global cost function, i.e.

$$\arg\min_{\mathcal{X}} \sum \mathcal{R}. \tag{3.1}$$

In the rest of this chapter we shall only take into account binary constraints, in which exactly two variables are considered, of the form $C_{i,j}\colon D_i \times D_j \to \mathbb{R}^2_{\geq 0}$. Note that the constraints between variables can be shown as an undirected graph, for example as shown in Figure 3.1.

Definition 3.1 (Neighbors). We refer to agents as *neighbors* if there is a constraint between their corresponding variables. This follows the real-life situation of limited range between agents, e.g. communication range in wireless networks. The set of all neighbors of an agent $\mathcal{M}_i \subseteq \mathcal{A}$ is called the *neighborhood*.

Definition 3.2 (Current Partial Assignment). The set $\hat{\mathcal{X}}_i$ denotes the set of known assigned values of the neighbors of A_i and is also referred to as the *current partial assignment* (CPA).

Definition 3.3 (Non-Iterative Algorithm). A non-iterative algorithm is one that does not rely on multiple rounds (iterations) of value assignments, or information sharing between agents, until some stopping criterion is met. A non-iterative algorithm is by definition asynchronous; it does not rely on synchronous activation certain phases of the algorithm, instead, the activation of agents happens sequentially triggered by local interactions.

3.2.1. DCOP: Existing Solvers

DCOP solvers can be categorized into *complete* and *incomplete*. Complete solvers search the entire solution space and are guaranteed to find the optimal solution, while incomplete solvers try to find a "good" solution in a reasonable time.

Incomplete solvers from literature, e.g. DSA [45], MGM-2 [100], DALO [76] or GCA-MGM [57] are local search algorithms, trying to approximate the global function by solving a local problem. DSA is known

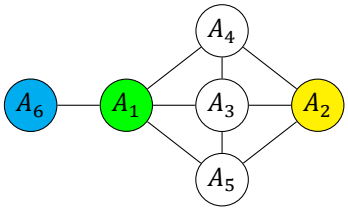

Figure 3.1: Example of a constraint graph in a graph coloring problem with six agents (vertices), variables (colors), and nine constraints (edges) between them.

for its low communication overhead and its ability to find high quality solutions for symmetric DCOPs [115], whereas ACLS and GCA-MGM can also solve ADCOPs.

The max-sum algorithm [38] is another incomplete solver, that works in a completely different manner. It operates on a bipartite graph, separating variable from constraint nodes, and spreads information through the graph to estimate the effect of value assignments. It is capable of finding optimal solutions, but only when the graphs contain no cycles [25]. The max-sum algorithm will converge to the optimal solution on acyclic graphs [25].In order to deal with cyclic graphs and asymmetric costs, variations of the algorithm have been proposed to deal with cyclic graphs such as max-sum_ADVP [159] or with asymmetric problems [158], but is then no longer guaranteed to find the optimal solution. Max-sum and the local search algorithms apply an *iterative approach*; they evaluate their performance, share information, update their variable, and repeat until a stopping criterion is met—usually a predetermined number of iterations.

3.2.2. Challenges

In this chapter we propose an algorithm capable of solving (A)DCOPs with a minimal communication and computation overhead. We hypothesize we can achieve this by *not* iteratively sending messages and updating the variable, but instead using a greedy, SSLA approach. This means that each agent takes a decision based on information only from its direct neighbors and will activate agents sequentially. Under these circumstances we need to address the following challenges:

Challenge 1: Premature Assignment A non-iterative DCOP solver assigns a value to a variable only once. In its most simple form an agent would only look at its local constraint costs and the known values of its neighbors. It would select a value that minimizes its local cost and update its variable. Greedy algorithms have the advantage of converging very fast, but early choices may turn out to be suboptimal when neighbors have assigned their value.

Challenge 2: Synchronization When two neighbor agents are both deciding for a new value a race condition may occur, i.e. the outcome of one agent arrives too late for the other agent's decision. For iterative algorithms this is not an issue since in a later iteration one agent can correct for any incorrect assumptions; for non-iterative solver this is not possible.

Challenge 3: Asymmetric Costs An incomplete solver may end up in a local minimum if a local beneficial assignment leads to poor global results. An agent may assign a variable to decrease its local cost, but potentially increases the cost of its neighbors. This *over-greediness* may cause the global cost to increase or lead to unstable solutions. In strongly asymmetric constraints for every combination of value assignments at least one agent can improve its local cost by shifting to another assignment without decreasing the global cost. Under these circumstances iterative solvers may not converge to the minimum where both agents are assigned the same value, but agents on either side of the constraint will maintain a cycle of assigning different values to improve their local cost.

3.3. CoCoA: A New DCOP Solver

To address the challenges introduced in the previous section we propose a new incomplete ADCOP algorithm based on a semi-greedy strategy, denoted as *Cooperative Constraint Approximation* (CoCoA), employing three key ideas:

1. *A single step lookahead (SSLA)* to consider the effect of an assignment on the cost of neighbors. This is especially effective when a neighbor is constrained in its choices;

2. *A unique-first* approach, such that an agent will only assign a value if it is a unique local optimum for its variable. If it cannot find a unique solution, the decision will be delayed until more information is available;

3. *A state machine* to spread and keep track of the algorithm's activity, prevent dead-locks or endless loops.

By minimizing not only the local cost, but also the cost of its single-hop neighbors, we hypothesize that CoCoA can find a solution with a low global cost with relatively little overhead. When CoCoA is triggered it first inquires its neighbors what the effect of different assignments would be for their local cost. The neighboring agents decide the resulting cost effect asynchronously and return to the inquirer what the minimum effect would be. Upon receiving the estimated costs the active agent will assign the value that minimizes the sum of all incurred costs, including its own.

During the variable assignment process it is possible that multiple values suit equally well, especially in an early stage of the algorithm when multiple neighbors have no assigned values. In such cases the assignment will be postponed until a neighbor has changed its value. We hypothesize that this *unique-first* approach will help in avoiding premature convergence to a sub-optimal solution. However, this approach may lead to a deadlock if all the neighbors are waiting for each other. Therefore, we partition the algorithm in four *states* and have agents inform their neighbors about their internal state. When all neighbors are in the HOLD state, a bound denoting the "allowed uniqueness" is increased until a decision can be made. The proposed algorithm is given in pseudocode in Algorithm 1, with accompanying set of messages and agent states in Table 3.1 and Table 3.2, respectively and discussed in detail in the subsequent section.

Remark. Due to its non-iterative approach, it is impossible to recover from early choices. This may lead to situations in which a variable *must* be set to a value that leads to a very high cost, i.e. it has to break a hard constraint.

3.3.1. CoCoA: Algorithm Description

CoCoA assumes that all neighbors A_i can communicate with, $\mathcal{M}_i$, are known. Also, each $A_j \in \mathcal{M}_i$ must know its neighbor's domain D_i. Finally, we assume that all nodes are reachable from any other node, i.e. for every pair of nodes there must be a set of edges that connects them.

Initially all agents start in the IDLE state and activation occurs at any random node. As soon as one agent finishes the algorithm it will trigger the algorithm for its neighbors. When any A_i has to find an assignment, it will first send an **InqMsg**$(i, \hat{\mathcal{X}}_i)$ message to its neighbors (line 3 of Algorithm 1). This will trigger agents $A_j \in \mathcal{M}_i$ to calculate for every possible assignment for X_i what the lowest cost would be for A_j taking into account the CPA and that assignment for X_i. That is, each $A_j \in \mathcal{M}_i$ calculates for every $X_{i,k} \in D_i$

$$\Theta_{j,k} = \min_{X_{j,l} \in D_j} \sum_{C \in \mathcal{R}_j} C\left(\hat{\mathcal{X}}_j \cap X_{i,k} \cap X_{j,l}\right), \tag{3.2}$$

where $X_{i,k}$ denotes that X_i is assigned the kth value of D_i.

If variables are not yet assigned in $\hat{\mathcal{X}}_j$, the cost of their constraints can not be determined and the mean cost is used, i.e. a *single* step lookahead is performed. The resulting cost map $\Theta_j = \{\Theta_{j,1}, \Theta_{j,2}, \ldots, \Theta_{j,|D_i|}\}$ is sent via a **CostMsg**(Θ_j) back to the inquiring A_i. Then, A_i finds X_i by

$$\delta = \arg\min_k \sum_{j=1}^{|\mathcal{M}_i|} \Theta_{j,k}, \tag{3.3}$$

and assigning $X_{i,\delta}$. The minimizing value may achieve a minimum for more than one value of X_i since the arg min operator can return a set of

Algorithm 1 CoCoA Algorithm

When started **or** upon receiving **UpdState**$(j, \texttt{DONE})$ on A_i:
Require: `state:=IDLE` **or** `state:=HOLD`
1: `state`$\leftarrow$ `ACTIVE`
2: send $\forall A_j \in \mathcal{M}_i$ **UpdState**$(i, \texttt{ACTIVE})$
3: send $\forall A_j \in \mathcal{M}_i$ **InqMsg**$(i, \hat{\mathcal{X}}_i)$
4: wait for all **CostMsg**(Θ_j)
5: find δ using (3.3)
6: **if** $\mathcal{U}(X_i) \leq \beta$ **or** number of idle/active neighbors is 0 **then**
7: $X_i \leftarrow$ random from $X_{i,\delta}$
8: `state`$\leftarrow$ `DONE`
9: send $\forall A_j \in \mathcal{M}_i$ **UpdState**$(i, \texttt{DONE})$
10: send $\forall A_j \in \mathcal{M}_i$ **SetVal**$(i, \hat{\mathcal{X}}_i)$
11: **else**
12: `state`$\leftarrow$ `HOLD`
13: send $\forall A_j \in \mathcal{M}_i$ **UpdState**$(i, \texttt{HOLD})$
14: **end if**

Upon receiving **InqMsg**$(i, \hat{\mathcal{X}}_i)$ at A_j:
15: incorporate $\hat{\mathcal{X}}_i$ in $\hat{\mathcal{X}}_j$*
16: **for all** $X_{i,k} \in D_i$ **do**
17: find $\Theta_{j,k}$ using (3.2)
18: **end for**
19: Send A_i **CostMsg**(Θ_j)

Upon receiving **UpdState**(i, S):
20: Store state S of neighbor A_i
21: **if** S is `HOLD` **and** my state is `HOLD` **and** number of idle/active neighbors is 0 **then**
22: β++
23: Repeat algorithm
24: **else if** S is `DONE` **and** my state is `HOLD` **then**
25: Repeat algorithm.
26: **end if**

*This line is added since the original publication for clarity; the implementation did not change.

minimizers. The *uniqueness* of this minimal cost is the number of distinct values that achieve this minimum, defined as $\mathcal{U}(X_i) = |\delta|$. For CoCoA this means that a value can now be assigned to the variable, and for that any minimizer will do. CoCoA_UF however, differs from CoCoA in that it also employs the unique-first approach.

In CoCoA_UF the uniqueness will be compared with a bound β to determine if this solution is accepted (line 6 of Algorithm 1). Initially $\beta = 1$, so that only unique solutions are accepted. If $\beta < \mathcal{U}(X_i)$ and at

Table 3.1: List of messages sent by CoCoA

Message	Description
InqMsg$(i, \hat{\mathcal{X}}_i)$	Sent by A_i at start of algorithm
CostMsg(Θ_j)	Neighbors' reply; contains Θ_j
SetVal(i, X_i)	Indicator: A_i assigned a value to X_i
UpdState(i, S)*	Sent when A_i updates its state to S

*$\texttt{UpdState}(i, \mathrm{S})$ is only sent in the unique-first, CoCoA_UF variant.

Table 3.2: Agents' potential states in CoCoA

State	Description
IDLE	Agent's default/initial state; indicates agent is active, but not yet started.
ACTIVE	State after agent's activation; finding an assignment for its variable.
HOLD	At an impasse; delay variable assignment and await information from neighbors.
DONE	Final CoCoA state; indicates that agent has assigned a final value to its variable.

least one neighbor is ACTIVE or IDLE, the algorithm switches to the HOLD state and waits until another node has updated its state to DONE before the algorithm is run again. If an **UpdState**$(j, \texttt{HOLD})$ message is received, indicating that the last neighbor is in the HOLD state, then β is increased by one and the algorithm is repeated (line 22 of Algorithm 1). This mechanism makes sure that premature choices are avoided until more information is available. Initially, when no agents have a value assigned this may occur frequently, but as more variables are set the chances of such *impasses* decrease.

If $\mathcal{U}(X_i) \leq \beta$ then X_i is chosen randomly from all minimizers and is communicated to the agent's neighbors $A_j \in \mathcal{M}_i$ in a **SetVal**(i, X_i) message. This makes the neighbors A_j update their CPA (as they now know the value of X_i) and triggers the algorithm for them.

3.3.2. CoCoA: Example Run

Let CoCoA solve a graph coloring problem, where each variable must be assigned value, which can be either blue (b), green (g), or yellow (y) such that $\forall i D_i = \{\mathrm{b}, \mathrm{g}, \mathrm{y}\}$. The constraints in the graph coloring problem are that neighboring variables should not have the same color—a cost of one is induced for every pair of neighbors that are assigned the same color. In the example in Figure 3.1 the variables of A_1, A_2, and A_6 are already assigned a color. Let us assume that this is the starting condition, and

we have to find the best assignment in this situation starting at A_3. As we shall see under these premises the best solution must violate some constraints—there is no perfect solution with zero cost.

Agent A_3 starts by sending **UpdState**$(3, \texttt{ACTIVE})$ and **InqMsg**$(3, \hat{\mathcal{X}}_3)$, informing that it has started and wants to know the effect its assignment may have, to all of its single-hop neighbors $A_j \in \mathcal{M}_3 = \{A_1, A_2, A_4, A_5\}$, where $\hat{\mathcal{X}}_3 = \{X_1 = \texttt{g}, X_2 = \texttt{y}\}$. As these messages arrive at all the neighbors, they will save the information that is in the CPA and the state of A_3.

Each neighbor will then calculate a cost map Θ_j, which contains for every assignment $X_{i,k} \in D_i$, what the lowest possible local cost is, given the CPA (3.2). Agent A_1 will return a **CostMsg**(Θ_1) message with the mapping $\Theta_1 = \{\texttt{g} \to 1, \texttt{y} \to 0, \texttt{b} \to 0\}$ and for agent A_2 this mapping will be $\Theta_2 = \{\texttt{g} \to 0, \texttt{y} \to 1, \texttt{b} \to 0\}$. For both A_4 and A_5 it will be $\Theta_4 = \Theta_5 = \{\texttt{g} \to 0, \texttt{y} \to 0, \texttt{b} \to 1\}$, and agent A_3 its own costs are $\Theta_3 = \{\texttt{g} \to 1, \texttt{y} \to 1, \texttt{b} \to 0\}$. All cost maps will be received by A_3, which sums over the possible assignments and finds $\{\texttt{g} \to 2, \texttt{y} \to 2, \texttt{b} \to 2\}$. There are now three potential assignments leading to the same minimal cost. Since initially $\beta = 1$, the choice is delayed until more information is available (because $\beta < 3$ and other neighbors are still either ACTIVE or IDLE). Agent A_3 sends an **UpdState**$(3, \texttt{HOLD})$ to its neighbors.

Since A_1 and A_2 are already DONE, they will not react to the new information. Agents A_4 and A_5 are now activated and after inquiring their neighbors, they gather a combined mapping $\Theta_j = \{\texttt{g} \to 2, \texttt{y} \to 2, \texttt{b} \to 1\}$. They can both find a unique minimal solution, so they assign their value to $X_j \leftarrow \texttt{b}$. Agents A_4 and A_5 send a **SetVal**$(j, \texttt{b})$ to spread the algorithm's activity and their new value. Upon receiving this information the neighbors update their CPA accordingly. Immediately after, an **UpdState**$(j, \texttt{DONE})$ message is sent, notifying all neighbors that they are now done.

Activity returns to A_3, who receives two **UpdState**$(j, \texttt{DONE})$ messages. After the *first* message it will assume that another neighbor is still active and will run the algorithm again without increasing β. This time, A_3 will find assignment costs $\{\texttt{g} \to 2, \texttt{y} \to 2, \texttt{b} \to 3\}$ (assuming that one of the neighbors is done, otherwise it would contain $\texttt{b} \to 4$; this is a race condition). As there are two minimizers and the uniqueness bound $\beta = 1$, A_3 will go to the HOLD state. After the second **UpdState**$(j, \texttt{DONE})$ message arrives, A_3 knows that there are no more active neighbors, so it will increase its bound $\beta \leftarrow 2$. Now it will find the cost of assignments to be $\{\texttt{g} \to 2, \texttt{y} \to 2, \texttt{b} \to 4\}$ with two distinct minima; since the uniqueness bound is now 2, it will select a random minimizer out of the two.

3.3.3. CoCoA: Termination Guarantees

With the state-mechanism in place, one could run the risk of entering an endless loop. We have the following Proposition:

Proposition 3.1. *The CoCoA algorithm will converge after a finite number of messages and function evaluations.*

Proof. Assume a situation in which an agent A_i and all of its neighbors are in the HOLD or DONE state. At some point A_i receives an **UpdState**(j, S) message and it will find that there are no more active neighbors, thus increases its β by one. CoCoA will run again and either there will be a unique solution or not. If no solution is found, A_i will set its state to HOLD, we are again at an *impasse*, and the process will repeat. At some point however $\beta = |D_i|$ since the domain is finite. At this point any assignment must satisfy $\mathcal{U}(X_i) \leq \beta$, so a value will be picked, and an endless loop is avoided. □

3.3.4. CoCoA: Privacy

When solving ADCOPs, there is always the possibility of transmitting the full local constraint cost matrix to one agent's neighbors. Sharing all constraint information between neighbors, and adding the received costs to the local costs, effectively converts any ADCOP into an equivalent symmetric DCOP. This strategy is also referred to as Private Events as Variables [101], and the main motivation not to use this strategy is the loss of privacy. In literature there are four different types of privacy defined [87]: agent privacy, topology privacy, constraint privacy, and decision privacy. The max-sum algorithm is known to be very private [138], especially because of its unique bipartite approach, separating utility and function nodes. However, when we consider constraint privacy, we argue that CoCoA is at least as privacy-conserving as max-sum.

Proposition 3.2. *CoCoA preserves at least as much privacy as max-sum.*

Proof. The max-sum algorithm is known to be more privacy preserving than the local search algorithms ACLS and GCA-MGM [158]. Only in two iterations entries from the cost matrices are exchanged between agents. In other iterations the shared values are derived from multiple entries as information spreads through the graph. CoCoA, on the other hand, shares cost matrix entries only once, i.e. before any agent has assigned a value and after that it is always derived from multiple entries. In CoCoA, for each assignment, the lowest total cost is sent taking into account the complete CPA (line 17 of Algorithm 1). As with max-sum, in every message to A_i, $|D_i|$ values are transmitted, however since CoCoA does not iterate, the number of exchanged messages is lower. □

3.4. CoCoA Performance: Experimental Results

CoCoA is tested and compared against state of the art DCOP solvers, being: DSA [45] (variant C, with update probability, $p = 0.5$), MGM-2 [100] (with offer probability $p = 0.5$), max-sum_ADVP [158] from hereon also referred to as simply max-sum, (switching graph direction after 100 iterations, value propagation after two switches, and using the constraint standard inner order), ACLS (with update probability $p = 0.5$) and GCA-MGM [57] (non-parametric). Also, we show the individual effects of the SSLA and

the unique-first approach by showing the results for CoCoA with and without the unique-first (UF) strategy.

For all experiments 100 problems are generated (the type of problems will be described subsequently) and the presented results are the average over all problems. To compare the performance of CoCoA we look at the following performance metrics: (i) the cost of the final solution (S), (ii) the number of transmitted messages (M), (iii) the number of cost function evaluations (E), and (iv) running time of the algorithm (T). A cost function evaluation is defined as computing or looking up the local cost of one constraint, similar to Non-Concurrent Constraint Checks (NCCCs) [104]. These are not necessarily non-concurrent, but they do indicate a non-implementation specific measurement of computational effort. We keep track of the global cost function and when no better solutions are found for more than 100 iterations the solver is stopped. Afterwards we report the performance metrics at the moment where a solver was first within 1% of the best solution. This approach is similar to an anytime framework as described in [102, 157], but instead of keeping track of the best state at every agent, we maintain this information in the experiment script; this information is only used for evaluation.

The solvers are implemented in Java 1.7, and the experiments are set up in MATLAB 2015b, which is also used to post-process and present the result figures[1]. The experiments are carried out on a laptop with an Intel Core i7-3720 CPU 2.6 GHz and 8 GB RAM.

3.4.1. Graph Coloring

A common problem for benchmarking DCOP solvers is the graph coloring problem e.g. [100, 108, 123]. As in the example run, the values of x represent the colors of nodes, and the solvers need to assign colors such that nodes on the ends of edges have different colors. In the first experiment every constraint violation will induce a cost of 1. In the experiment the number of colors $|D| = 3$, so the cost matrix for every constraint is $\mathbf{C} = \mathbf{I}_3$. The graphs are generated by selecting $n = 500$ random points in two-dimensional space using a Poisson point process, representing the variables, and the constraints are chosen as the edges of a Delaunay triangulation between those points—the average density of the graphs is 0.01.

Experiment Results

In Figure 3.2 the solution cost is plotted against the running time for several algorithms, while Table 3.3 shows the full set of metrics. CoCoA_UF finds a solution of near optimal cost in a single iteration, requiring less function evaluations than any other algorithm, and second least

[1]For a replicability of results and figures, the source code of the implemented algorithms is available at https://github.com/coenvl/jSAM/tree/AAAI17, and of the experiment setups and visualization at https://github.com/coenvl/mSAM/tree/AAAI17.

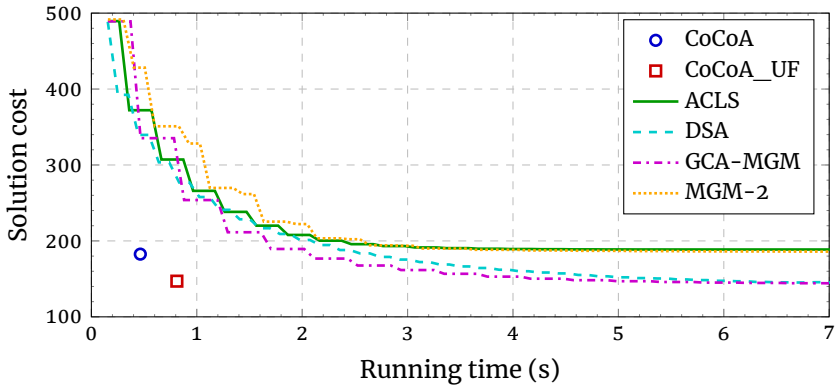

Figure 3.2: Graph coloring experiment: DSA finds the best solution, but CoCoA_UF finds a similar solution in less time.

Table 3.3: Graph coloring experiment results

Algorithm	I	S	M*	E*	T
CoCoA	N/A	183	**8.8**	**138**	**0.5**
CoCoA_UF	N/A	147	15.4	152	0.8
ACLS	35	189	101.0	194	3.5
DSA	200	**129**	28.3	1 177	18.9
GCA-MGM	58	144	128.0	213	6.0
MGM-2	80	184	98.2	452	8.1

* = $\times 10^3$

number of messages; therefore it is also the fastest algorithm. The result demonstrates that the unique-first approach definitely provides a benefit in terms of eventual solution cost, as CoCoA_UF finds a solution that is 20% better than CoCoA at the cost of some additional messages and cost function evaluations, almost as good as DSA, which finds the best.

In the experiment max-sum is left out since it is unable to converge to a solution. This is because there are $|D|$ "mirrored" solutions that perform equally well, and there is no local preference of one coloring over the other.

3.4.2. Semi-Randomized Asymmetric Problems

In the second experiment we generate semi-random asymmetric problems by creating scale-free graphs according to [5] with an initial graph of ten randomly connected nodes, and iteratively adding up to four nodes until $n = 200$, resulting in graphs with an average density of 0.04. The variable domain size $|D_i| = 10$, and for every constraint an integer semi-random cost is generated for both sides of the constraints. A cost of zero is selected with a probability of $p = 0.35$ and uniformly randomly

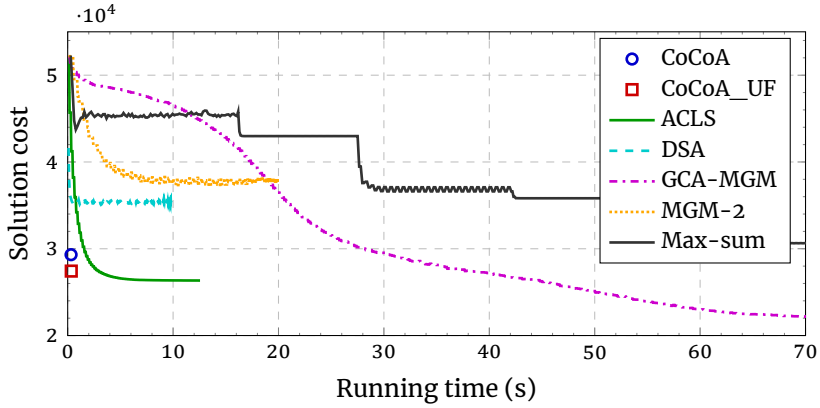

Figure 3.3: Semi-randomized asymmetric experiment: CoCoA finds a Pareto-optimal solution, but GCA-MGM eventually finds the best solution after more than 60 seconds.

Table 3.4: Semi-randomized asymmetric experiment results

Algorithm	S*	M*	E*	T
CoCoA	29.3	**4.8**	1 595	**0.3**
CoCoA_UF	27.4	6.9	1 334	**0.3**
ACLS	26.6	153.1	1 296	5.0
DSA	32.3	40.8	**1 250**	3.8
GCA-MGM	**22.3**	1 349.2	5 397	59.8
MGM-2	35.1	91.8	3 287	11.5
Max-sum	27.1	1 709.4	51 221	133.9

* = $\times 10^3$

chosen in the domain $[1, 100]$ for the remainder. This setup recreates the experiment as described in [56, Section 5.2]. In this experiment the DSA and MGM-2 algorithms are taken into account, even though it is already known that they cannot solve asymmetric problems; they are used to demonstrate the fact that this problem is indeed an ADCOP.

Experiment Results

Figure 3.3 presents the solution costs of different algorithms as they converge to a solution. CoCoA and CoCoA_UF quickly converge to a good solution, more than 10 times faster than any other algorithm. The symmetric DCOP solvers DSA and MGM-2 fail to find a solution. The max-sum algorithm also converges to a reasonable solution, but only after more than two minutes (not visible in Figure 3.3). The GCA-MGM algorithm shows a "discovery" phase in the first 25 seconds, after which it finds a global optimum, which is better than any other algorithm finds. The added overhead of the GCA-MGM algorithm can clearly be seen in Table 3.4. The better so-

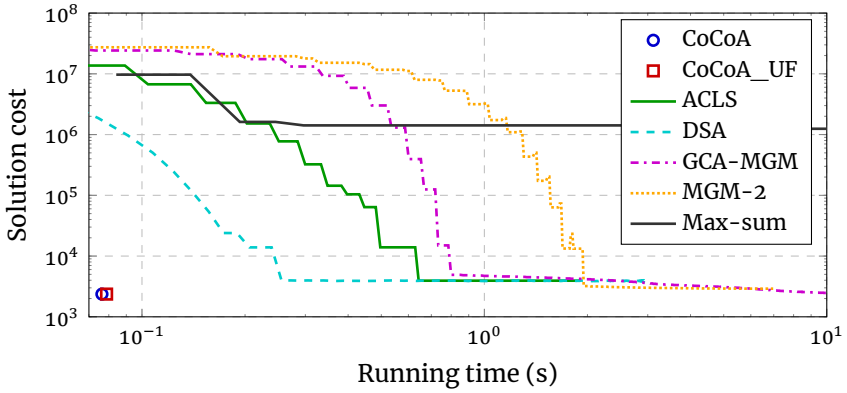

Figure 3.4: Sensor planning experiment: CoCoA finds the optimal solution and converges the fastest out of all evaluated algorithms.

Table 3.5: Sensor planning experiment results

Algorithm	S	M*	E*	T
CoCoA	**2 365**	**1.2**	389	**0.1**
CoCoA_UF	**2 365**	1.8	332	**0.1**
ACLS	3 934	7.0	**44**	0.3
DSA	3 332	13.1	427	1.3
GCA-MGM	2 379	197.2	952	10.7
MGM-2	2 917	24.5	1 161	3.9
Max-sum	1 246 306	12.0	1 300	0.9

* = $\times 10^3$

lution is found by sending nearly 200 times the amount of messages, evaluating 4 times the number of cost functions and running 200 times longer than CoCoA_UF.

3.4.3. Sensor Planning

The final experiment is motivated by an example in which a sensor network is used to monitor the cargo state of a shipping container. The sensors have to maintain a good quality estimation of the cargo such that they can either warn the cargo owner in case the shipping circumstances unexpectedly change, or provide a trace of the cargo state upon arrival. In this scenario the cargo estimation has to be optimized, but is constrained by limited battery life. The scenario is the adapted from Chapter 2, from which we use the outcome to model the effect of the communication frequency on the estimation quality and the battery lifetime. In this problem x are communication rates between sensors, and hard constraints make sure that the agents (nodes) will meet the

minimum required battery lifetime. Asymmetric constraints between agents are used to model the effect that more *shared* information does not reduce the local estimation error, but it does improve the performance of neighboring nodes. The domains $\mathcal{D}$ are integer communication frequencies of $[1, 11]$ Hz, and the networks are generated by connecting $n = 50$ nodes, in randomly generated graphs with an average density of 0.17.

Experiment Results

The results are show in Figure 3.4 and Table 3.5. They show that CoCoA is not only capable of finding the best solution, but also does so in the least amount of time and using the fewest messages from all considered algorithms—only in terms of computational effort, ACLS is more efficient. However, ACLS is unable to find a better solution than the symmetric solvers DSA and MGM-2. This may contribute to its low evaluation count—it is simply considered as "converged" after a few evaluations.

3.5. Conclusions

In this chapter we studied the problem of coordinated decision-making in multi-agent systems. In order to reason about the configuration of an agent, taking into account the neighboring agents, we have proposed and investigated a new ADCOP solver: CoCoA. CoCoA is a non-iterative, semi-greedy strategy using SSLA to incorporate the effect a decision has on neighboring nodes. We compared its performance with state-of-the-art solvers by using (i) three-color graph coloring, (ii) randomized asymmetric problems and (iii) a sensor network use case problem.

We showed that CoCoA finds high quality solutions, with a similar overhead for symmetric problems, and a much smaller overhead for asymmetric problems, both in terms of communication and running time. The GCA-MGM algorithm finds better solutions for randomized asymmetric problems, but requires up to 200 times more messages, and over 4 times more cost function evaluations. In the sensor planning problems CoCoA finds better solutions, and does so faster than the benchmarks. When we look at performance of the algorithm with or without the unique-first approach, we can conclude that preferring unique solutions yields a clear advantage in terms of solution cost.

Because CoCoA requires no iterative approach, it can be used in applications where fast convergence is required, or when communication is a limiting factor. We have shown that for this reason CoCoA is very well suited to find assignments in sensor planning problems. CoCoA seems a promising method for collaborative decision-making, to find communication strategies that benefit everyone.

In recent studies, another extension of DCOPs was proposed: Functional DCOPs or F-DCOPs. In F-DCOPs the variable domains are continuous, and consequently the constraints must be expressed as functions,

and cannot be written as tables [23, 46, 62]. C-CoCoA [126] is an extension of the CoCoA algorithm, which is capable of solving F-DCOPs, using the same strategies underlying CoCoA i.e. it uses a non-iterative one-step lookahead to asses the effects of local decision on neighboring nodes.

A downside of CoCoA is that it cannot recover from early choices, and there is a serious chance of simultaneous assignments in neighboring nodes which in the presence of hard constraints may lead to infeasible solutions. In the next chapters, these shortcomings will be addressed. First, in Chapter 4 an extension is presented that avoids simultaneous assignments, which will be utilized in a use case with hard constraints in wireless power transfer. Then, in Chapter 5 an extension is considered, where the best of both worlds is exploited; the fast convergence of CoCoA is complemented with high quality solutions, and possibility to improve on early choices, of other solvers.

4

Self-organizing Wireless Power Transfer

4.1. Introduction

In this chapter, an extension to the CoCoA algorithm (from the previous chapter) is introduced called *CoCoA Controlled Activation* (CoCoA_CA). As we concluded in the previous chapter, CoCoA is a good starting point for an algorithm driving self-organization, but might have difficulty dealing with hard constraints. The underlying problem is that the activation of the algorithm is likely to occur simultaneously in neighboring agents. When this happens, both agents make a decision based on the same information, but cannot take into account the assignment that other agent is making. In order to address this, an additional check is added to make sure simultaneous decisions cannot occur, and experiments are carried out to see the effect of this additional cycle.

A use case is presented in which a self-organizing system controls a network of *Wireless Power Transfer* (WPT) transmitters. These transmitters form a system which has a goal to provide electrical power to a group of receivers, by means of radio frequency (RF) waves. Such radio waves are safe for operation in an environment where humans are also present, provided that transmission levels are maintained. In this chapter we will propose a wireless charging protocol, built on top of the CoCoA_CA algorithm. Since for the WPT use case, it is crucial that safe transmission levels are maintained, hard constraints must be implied on the underlying problem (when represented as a DCOP). Therefore, we will need to make sure that hard constraints are never violated, justifying the use of the CoCoA_CA algorithm.

This chapter is based on the article by C. J. van Leeuwen, K. S. Yıldırım, and P. Pawełczak, *Self adaptive safe provisioning of wireless power using DCOPs*, in Proc. SASO (2017) [93].

4.1.1. A Primer on Wireless Power Transfer Networks

Today's ecosystem of the IoT is composed of millions of embedded devices that can monitor and control the physical world [4]. These devices are equipped with several hardware components such as sensors, actuators, microcontrollers and transceivers, that still consume a considerable amount of power. Fortunately, the research efforts on electronic circuits have already decreased the power consumption of these hardware components to a few microwatts [54]. This allows the provision of power wirelessly by means of harvesting the energy of RF waves [106, 141]. As the efficiency of harvesting circuits will improve, in the future more devices can be powered wirelessly, using only RF energy without any external power source such as batteries.

The electromagnetic radiation (EMR) at a particular point can be modeled as a linear function of the received power [27, 58]. Since, several energy transmitters can be active simultaneously to charge nearby receivers, it might be the case that the total EMR value at some particular points in the charging area—which have a contribution from all active transmitters—can exceed the limit defined by the RF exposure regulations [27, 97]. Therefore, a *safe-charging* WPTN must ensure that it does not create harmful electromagnetic radiation [143], as it is trying to minimize the charging time by increasing the power levels of the transmitters [97].

For safe, sustainable and continuous operation, a wirelessly-powered IoT system requires several dedicated RF energy transmitters that can control their power level to charge nearby receivers *collaboratively*, forming a *Wireless Power Transfer Network (WPTN)* [48, 65, 96]. The deployment of this conceptual network is an important issue for the fast provisioning of wireless energy. Since in such a network the amount and locations of receivers is continuously changing, it is especially important to make use of a power optimization strategy, that is fast and reliable. Therefore, using local information for decentralized decision-making seems an appropriate strategy, and hence self-organizing multi-agent systems, is a natural fit for this problem.

4.1.2. Problem Statement

The main challenge is the maximization of the *total transmitted power* by finding individual transmission power levels of the transmitters, while satisfying the *safety constraints*, which is even more difficult due to the dynamic charging environment. There are two requirements that need to be addressed:

1. **Dealing with the dynamic environment** New energy transmitters, as well as receivers, can be introduced to the charging system. This means that EMR values at some particular locations can exceed the threshold if these new transmitters start provisioning power. The

optimization strategy needs to be able to include addition, removal or changing of its variables and constraints.

2. **Dealing with model uncertainty** Based on a transmitter output power, it is difficult to model and estimate exactly the received power, and in turn the EMR value, due to the environmental dynamics of the RF wave propagation. Therefore, the latest (local) information should *always* be taken into account, and an offline centralized charging algorithm is not feasible.

Apparently, *self-adaptation* or *self-organization* shall become a necessary property of a WPTN such that the charging algorithm should transfer the network into a safe state, when the safety constraints are violated. Unfortunately, we are unaware of a *self-organizing and safe* charging algorithm in the current literature that (i) allows energy transmitters to interact with the energy receivers and sensors locally in a dynamic environment; (ii) does not use a centralized entity to find sub-optimal power levels subject to safety regulations.

4.1.3. Contributions

In this chapter we propose a method based on DCOP solvers to find near-optimal solutions for the optimization problems without any centralized entity and by only using local interactions among the nodes. Accordingly, in this chapter, we introduce a new wireless charging system called TESSA (Transferring Energy Safely by Self-Adaptation). TESSA is based on a variation of the DCOP solver CoCoA proposed in Chapter 3, and is self-adaptive, in the sense that it runs an algorithm on the transmitter nodes that will find an optimal transmission power level. This not only keeps a WPTN near-optimal in terms of power transfer, but also safe with respect to EMR regulations. Within this context, we provide the following contributions to the state of the art:

1. We formulate a distributed constraint optimization problem where the energy transmitter devices maximize the total transmitted power to the receiver devices, i.e. minimize the *charging time*, while keeping the network *safe* in the EMR sense using measurements from the locally deployed sensors;

2. A variation on the CoCoA algorithm is proposed, called CoCoA_CA, that enables solving the distributed optimization problem, while making sure the EMR thresholds are never violated. This algorithm drives a self adaptive charging system, called TESSA. TESSA optimized the charging network to a safe state, even when perturbed by environmental dynamics such as new energy transmitters joining in the network;

3. We compare the CoCoA_CA algorithm with the existing DCOP solvers via simulations. Our results show that the existing algorithms are unable to find feasible solutions, whereas CoCoA_CA finds solutions near the theoretical optimal, reaching on average 85% of the maximum (optimal) solution, in terms of power transmission;
4. We show that our charging system based on DCOP solvers maintains safe EMR levels, without relying on a model for the RF propagation. Hence, it is capable of dealing with unexpected RF measurements, which a model-based centralized solution cannot.

4.2. Related Work

We provide a brief overview of the related work on wireless power transfer and distributed constraint optimization, which our work builds upon.

4.2.1. Wireless Power Transfer

The number of IoT nodes continues to grow exponentially [11, 128]. This exposes a problem of sustainable energy provision to such a mass of (battery-powered) IoT devices. Fortunately, the recent advancements in RF energy harvesting and low-power electronics, e.g. [49], make it feasible to power ultra low-power microcontroller-based IoT devices wirelessly using electromagnetic energy [134]. Wireless power transfer revealed several optimization problems that gained considerable attention from the research community, e.g. the optimization of the harvested power [84], energy outage [64] and charging delay [47].

However, the electromagnetic safety issues provide additional constraints on the wirelessly supplied power. Specifically, since an EMR value that is above the limits of well-defined regulations [71] is considered as a threat to human health [111], it introduces an important constraint to the objective of aforementioned optimization problems.

To the best of our knowledge, there are only two recent studies (from the same authors) that target the optimal wireless power transfer incorporating the human health effects. A one-shot *centralized* solution that maximizes the total transmitted power subject to the safety constraints at each point on a predefined deployment area is presented in [27]—in [28], the same problem is handled in a *distributed* fashion. Unfortunately, these two studies have a fundamental drawback: they use deterministic models to estimate EMR. However, RF propagation is non-deterministic and modeling errors might lead to violate the safety constraints. Moreover, the distributed algorithm in [28] is quite complex and composed of several phases. The algorithm employs a distributed redundant constraint reduction algorithm, splits the deployment area into small squares and employs linear programming (LP) by considering the local constraints inside each square. Therefore, it cannot be considered as a self-organizing solution since optimization is not performed by local interactions solely,

rather than using the global information within each square.

The charging algorithm that we will introduce is not dependent on any RF propagation model since it uses direct measurements from the sensor nodes. Therefore, it always satisfies the safety constraints. Moreover, our algorithm is fully-distributed and self-organizing, based on the local interactions among the energy transmitters, receivers and sensors.

4.2.2. Distributed Constraint Optimization

DCOPs are a type of problems from the field of multi-agent systems where agents need to cooperatively assign a set of variables in order to optimize some cost function. We introduced the formalization and notation of DCOPs in Section 3.2. DCOP solvers can be divided into complete and incomplete solvers, which provide either the global optimal solution, or a near-optimal solution near the overall best, respectively. However, since the optimization problems are NP-hard [107], optimal solvers are by definition exponentially slow for increasing problem sizes. Therefore, for the application for WPT, we are more interested in incomplete solvers that are not guaranteed to find the optimal solution, and find a good solution in a feasible time.

Iterative solvers

Most incomplete DCOP methods use an iterative approach and belong to the class of local search algorithms. This means that these solvers start with an initial variable assignment and iteratively search the local problem space for a solution that incrementally improves, until it converges to a local optimum. Some examples of solvers belonging to this class are the DSA [155], MGM or MGM-2 [100] algorithms. Solvers for problems with asymmetric constraints such as ACLS and GCA-MGM [56] or max-sum [158] are also considered iterative.

Non-iterative solvers

There exist some *complete* non-iterative solvers in the literature such as DPOP [116], ADOPT [108] or AFB [52]. However, since these complete solvers are very time-consuming, for the Wireless Power Transfer problem an incomplete solver is preferable. To the best of our knowledge there is only one non-iterative incomplete approach, which is offered by the CoCoA algorithm as introduced in Chapter 3.

The advantage of the non-iterative solvers is that they will immediately present their final solution. This means that *if* the solver is capable of finding a solution that satisfies the EMR constraints, it will immediately find this solution, thus *never* violating the EMR thresholds. This is in contrast to iterative solvers, that might initially violate the constraints, even if it may eventually find a solution that yields more transmitted power.

4.3. System Model: A Network of Energy Provision

We abstract a WPTN as a graph in the 2D plane, that has different types of nodes representing either energy transmitters (ETs), energy receivers (ERs) or sensor nodes. It is assumed that each ET node is equipped with an antenna that can emit RF waves to charge ER nodes inside its *power transmission range* wirelessly. Besides, each ER node is assumed to be equipped with an RF-harvester circuit that accumulates the harvested energy into the storage component. Moreover, each sensor nodes is assumed to be able to measure the EMR value at its specific location.

We further assume that each ET, ER and sensor node is equipped with a transceiver that allows communication with the other nodes inside their *communication range*. For the sake of simplicity, it is assumed that the power transmission range of the ET nodes are identical to their communication range, and ET, ER and sensor nodes are *stationary* during the optimization process.

4.3.1. ET Model

The set of ET nodes is denoted by $T = \{T_1, T_2, \dots, T_t\}$, where t represents the number of ETs. We denote the transmission power of T_i such that $i \in \{1, \dots, t\}$ by P_i and assume that each ET node can modify it by assigning values in the interval $[P_{\min}, P_{\max}]$. The set of ER and sensor nodes with which T_i can communicate and transfer power, i.e. the set of neighbors of T_i, is denoted by $\mathcal{M}_i$.

4.3.2. ER Model

We denote the set of ER nodes by $R = \{R_1, R_2, \dots, R_r\}$, where r represents the number of ERs. Each $R_j \in \mathcal{M}_i$ such that $j \in \{1, \dots, r\}$ receives power from the ET node T_i. Following the model in [58, Eq. (4)] which is based on Friis' free space equation, the harvested power at R_j from T_i can be expressed as

$$P_{i \to j} = \eta \frac{G_i G_j}{L_p} \left(\frac{\lambda}{4\pi (d_{ij} + \beta)} \right)^2 P_i, \tag{4.1}$$

where G_i and G_j are antenna gains of T_i and R_j, λ is the wavelength, L_p is the polarization loss, β is a parameter to adjust the Friis' free space equation for short distances, d_{ij} denotes the distance between T_i and R_j and η represents the efficiency coefficient of the RF harvester. By introducing $\gamma = \frac{G_i G_j}{L_p} \left(\frac{\lambda}{4\pi} \right)^2$, we can write $P_{i \to j}$ in compact form as

$$P_{i \to j} = \eta \frac{\gamma}{(d_{ij} + \beta)^2} P_i. \tag{4.2}$$

In reality, the emitted energy waves from transmitters can combine at receivers either constructively or destructively, that might led to energy

cancellation [109]. For the sake of simplicity, we consider only the case of constructive combination. In turn, we model the total harvested power at receiver R_j as the sum of individual received energy from the transmitters, which we denote by θ_j as

$$\theta_j = \sum_{i:R_j \in \mathcal{M}_i} P_{i \to j}. \tag{4.3}$$

4.3.3. Sensor Model

The set of sensor nodes is denoted by $S = \{S_1, S_2, \dots, S_s\}$, where s represents the number of sensors. Each sensor S_k such that $k \in \{1, \dots, s\}$ is able to measure the EMR value at its specific location. We model the EMR value at S_k as a *linear* function of the total transmitted power as

$$E_k = \rho \sum_{i:S_k \in \mathcal{M}_i} \frac{\gamma}{(d_{ik} + \beta)^2} P_i, \tag{4.4}$$

where ρ is a constant that captures the linear relationship between the EMR and received power, and d_{ik} denotes the distance between T_i and S_k.

4.4. Problem Description

We define the centralized linear programming problem. Define EMR threshold as α and let $\mathcal{P} = [P_1, \dots, P_t]^T$. We formulate the optimization problem as

$$\begin{aligned} \max_{\mathcal{P}} \quad & \sum_j \theta_j, \\ \text{s.t.} \quad & \forall k : E_k \leq \alpha, \\ & P_{\min}\mathbf{1} \leq \mathcal{P} \leq P_{\max}\mathbf{1}, \end{aligned} \tag{4.5}$$

where $\mathbf{1}$ denotes a vector with all components equal to 1.

4.4.1. Translation into a DCOP

In DCOPs problems are described as a tuple $\mathcal{T} = \langle \mathcal{A}, \mathcal{X}, \mathcal{D}, \mathcal{R} \rangle$. We can formulate the wireless power transfer problem by stating that every transmitter in T is represented by an agent $\mathcal{A} = \{A_1, A_2, \dots, A_n\}$ and their corresponding transmission powers in P are the variables that are being optimized $\mathcal{X} = \{X_1, X_2, \dots, X_n\}$. One of the main assumptions in DCOP theory is that all variables must have finite discrete domains $\mathcal{D}$, which should contain every possible power level. This means that the interval $[P_{\min}, P_{\max}]$ has to be discretized to contain a finite set of possible values, yielding n variable domains $\mathcal{D} = \{D_1, D_2, \dots, D_n\}$. This discretization can follow a specific set of possible power values that the energy transmitter allows.

Finally, in a DCOP, $\mathcal{R}$ is the set of constraints, which map the assignment of variables to a non-negative cost: $C\colon D_{i_1} \times D_{i_2} \times \ldots \times D_{i_k} \to \mathbb{R}_{\geq 0}$. In the WPT problem statement (4.5) there are two types of constraints. Firstly, we have the constraints that will maximize the amount of energy transmitted. These constraints represent the ER nodes R with a constraint cost function such that

$$C_{R_j} = -\theta_j. \tag{4.6}$$

Note that we model *maximization* of transmitted energy, as the minimization of the negative energy. Secondly, there is the set of sensors S that are modeled using a threshold constraint. This constraint aims to make sure that the EMR safety threshold should not be exceeded. This is modelled as

$$C_{S_k} = \begin{cases} 0, & \text{if } E_k < \alpha, \\ \tau, & \text{otherwise,} \end{cases} \tag{4.7}$$

where a hard constraint can be simulated by choosing $\tau = \infty$ as described in [13], or simply a very high value such that $\theta_j \ll \tau$. Hence, the full set of constraints $\mathcal{R}$ can be defined as the combined set of both receiver and sensor constraints

$$\mathcal{R} = \{C_{R_1}, C_{R_2}, \ldots, C_{R_r}, C_{S_1}, C_{S_2}, \ldots, C_{S_s}\} \tag{4.8}$$

such that the minimization function will maximize the transmitted power, while minimizing the number of sensors where the EMR threshold is violated. The DCOP minimization function is defined simply as

$$\arg\min_{\mathcal{X}} \sum \mathcal{R}. \tag{4.9}$$

4.5. TESSA: A Safe Wireless Charging System

Having presented the system model of wireless power transfer we consider in this chapter, we are ready to present our charging system. We first present the high-level charging protocol that governs the charging requests of the energy receiver devices. Then, we present the extension of CoCoA denoted as CoCoA_CA, which is triggered by TESSA.

4.5.1. The Main Charging Protocol

The main charging protocol, executed by each transmitter T_i in the wireless power transfer network, is presented in Algorithm 2. The objective of this protocol is to trigger the CoCoA_CA algorithm, presented in Algorithm 3 and discussed in Section 4.5.3, that will determine the sub-optimal power levels for the transmitters. Initially, the power transmitter can be turned off and waiting for a receiver inside its neighborhood $\mathcal{M}_i$ to send a `Charge` message. The transmitter will add the corresponding receiver to its requests list $\mathtt{R}_i$ and reset the CoCoA_CA algorithm to recalculate the optimal power levels (Lines 1–3). Similarly, when a

Algorithm 2 Charging Protocol executed by Transmitter T_i

1: $R_i \leftarrow \emptyset$ {Vector of charge requests for transmitter i}

Upon receive **`Charge`** from $R_j \in \mathcal{M}_i$
2: $R_i \leftarrow R_i \cup \{R_j\}$ {Add new request from receiver j}
3: **`Reset`** CoCoA_CA {Execute CoCoA_CA optimizer}

Upon receive **`EndCharge`** from $R_j \in \mathcal{M}_i$
4: $R_i \leftarrow R_i \setminus \{R_j\}$ {Remove request from receiver j}
5: **if** $R_i = \emptyset$ **then**
6: Turn off charger T_i
7: **end if**
8: **`Reset`** CoCoA_CA {Execute CoCoA_CA optimizer}

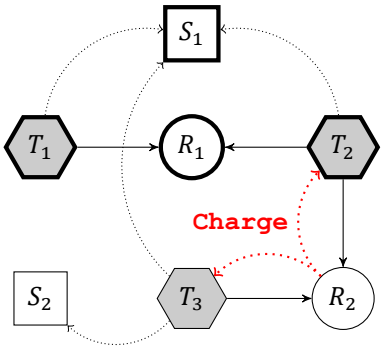

Figure 4.1: Illustrative example of TESSA execution. Transmitters T_1 and T_2 are currently charging the receiver R_1 subject to the constraint on the sensor S_1. The receiver R_2 sends a charging request to transmitters T_2 and T_3 that forces T_3, T_2, and in turn T_1, to run the CoCoA_CA optimizer again with also considering the constraints on the sensor S_2.

`EndCharge` message is received, the corresponding receiver is removed from the requests list, the transmitter is turned off if the requests list is empty and CoCoA_CA algorithm is restarted (Lines 5–10).

The rationale behind resetting and in turn restarting the actual DCOP solver CoCoA_CA is to force the network to adapt to the new state. As an example, consider Figure 4.1: transmitters T_1 and T_2 are currently charging receiver R_1, and receiver R_2 sends a charging request to transmitters T_2 and T_3. Since the sensor S_1 is already in the neighborhood of T_3 and there is a new sensor S_2 that will be solely affected by the power transmission of T_3, the EMR constraints on these sensors will affect not only the power level of T_3 but the current power level of T_2 and in turn T_1. Therefore, if T_3 start-s/resets CoCoA_CA, the neighboring transmitter T_2 should be informed. In turn, that will lead T_2 to inform T_1 so that all transmitters re-calculate the optimal power levels.

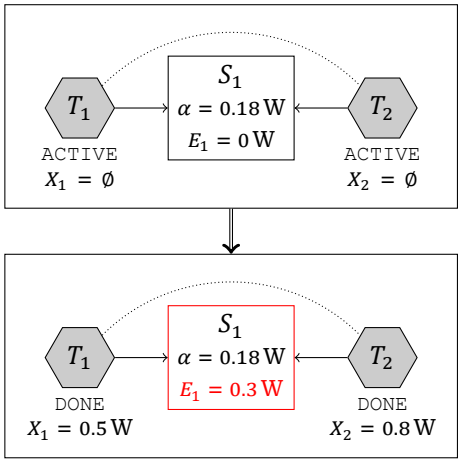

Figure 4.2: Race condition problem of CoCoA in the wireless power transfer context. In the initial state (top), since S_1 is measuring 0 W, both transmitters decide on a high value eventually exceeding the example EMR threshold of 0.18 W (bottom). E_1 is computed according to (4.4) where $S_1 \in \mathcal{M}_1$ and $S_1 \in \mathcal{M}_2$.

4.5.2. CoCoA and Race Conditions

The CoCoA algorithm is introduced in Chapter 3. By itself, it already has some properties to avoid race conditions, i.e. situations in which two agents simultaneously make a decision on a variable assignment. However, they are not fully effective, since two agents may become simultaneously activated from different neighbors. Also, CoCoA is designed under the assumption that there are only binary constraints in the problem graph; when applied to higher order constraints, race conditions are more likely to occur and may involve more than two agents.

If race conditions would occur in the WPT scenario, this means that two (or more) transmitters simultaneously decide on a power level. The situation may occur as shown in Figure 4.2, where initially two transmitters (T_1 and T_2) are simultaneously actively running the CoCoA algorithm. As they both know that a shared sensor (S_1) is not exceeding the EMR threshold, they may decide both to increase their power level. Not taking into account the assignment of the neighbor, it is possible for the two agents to assign two values that actually would violate the EMR constraint.

It is very difficult to avoid race conditions from occurring in a multi-agent system. However, since CoCoA already provides a mechanism to disseminate the current state of the algorithm, we can detect if one has occurred. In the following section we introduce the extension of CoCoA: CoCoA_CA, where race conditions are recognized, and concurrent assignments avoided.

Algorithm 3 CoCoA_CA Algorithm

Algorithm start on A_i:
1: Assign $\phi_i \leftarrow$ `ACTIVE` and inform neighbors
2: Send request to neighbors for cost maps
3: Wait for all responses
4: **if** number of `ACTIVE` *higher ranked* neighbors > 0 **then**
5: Go to line 2
6: **end if**
7: Find minimizing assignments for X_i
8: **if** number of minimizer $\leq \beta$ **or** number of idle/active neighbors is 0 **then**
9: Assign X_i and $\phi_i \leftarrow$ `DONE`; send to all neighbors
10: **else**
11: Assign $\phi_i \leftarrow$ `HOLD` and send to all neighbors
12: **end if**

Upon receiving cost inquiry message at A_j:
13: **for all** $X_{i,k} \in D_i$ **do**
14: Calculate costs for $X_{i,k} \cap \theta_r \cap E_s$
15: **end for**
16: Reply A_i with all costs

Upon receiving new state ϕ_i from $A_i \in \mathcal{M}_j$ on A_j:
17: Store neighbor's state ϕ_i
18: **if** ϕ_i is `HOLD` **and** ϕ_j = `HOLD` **and** number of idle/active neighbors is 0 **then**
19: Increment uniqueness bound β and repeat algorithm
20: **else if** ϕ_i is `DONE` **and** ϕ_j is `HOLD` **then**
21: Repeat algorithm
22: **end if**

Upon receiving **`Reset`** on A_i:
23: **if** $\phi_j \neq$ `IDLE` **then**
24: Assign $\phi_j \leftarrow$ `IDLE`
25: Forward **`Reset`** to all neighbors
26: **end if**

4.5.3. Solving CoCoA Race Condition Issue: CoCoA_CA

In Algorithm 3 we present the pseudocode of the CoCoA_CA algorithm. Note that in the algorithm we use ϕ_i to denote the state of A_i, which initially is `IDLE`, but can be set to `ACTIVE`, `HOLD` or `DONE`. Also, the uniqueness bound β is initially set to 1. CoCoA_CA starts out the same way as CoCoA does by inquiring the neighbors the costs of local assignments. Then in lines 4–6 the algorithm checks whether any neighboring agents are also currently running, and if there are—it will go back in the algorithm to the point where it will gather information anew. However, since this current agent

itself is also running, we would introduce a deadlock here, where two simultaneously activated neighbors would stay in this cycle ad infinitum. In order to break this potential deadlock we introduce this notion of ranking.

At line 4, we check the number of *higher ranked* neighbors. In principle any ranking could be used, as long as all involved agents agree on the ranking. In our implementation we use the alphabetical ranking of the identifier of the variable, but any other rankings, such as the MAC addresses of the agents may be used as well. Even a random number selected at the time of this impasse would serve, as long as there is always *one* highest ranked agent. Only the highest ranked agent may finish the variable assignment, and the other agent(s) will have to restart the algorithm. By doing so, we make sure that no two agents are deciding on an assignment simultaneously.

Between line 7 and 12, where the algorithm assigns a variable based on the neighbors' cost messages, the logic is the same as for CoCoA. All received costs are added, and the minimizing value is selected if the *uniqueness* of the minimizer is less or equal than β.

When a message is received inquiring about the assignment costs (lines 12–16), nodes update their state based on the received value assignments and gather information from the receivers based on either their actual measurements or estimations based on the theoretical energy harvesting model as per (4.3), and from the sensors by requesting their measured power level. By using the actual measured power, not the model predicted amount, we can make sure that the EMR thresholds are always satisfied. Using this information we can compute the local cost with (4.6) and (4.7) in line 14.

For handling the state update message from neighbors (lines 17—22) the same logic is applied as in the original CoCoA algorithm. Whenever a `HOLD` state is received, the algorithm checks if the uniqueness bound β needs to be updated, or it will repeat the algorithm if the agent itself is in the `HOLD` state and a neighbor informs us that it is finished.

Finally, an additional message was added to reset the algorithm. At line 23 we specify that if a **`Reset`** message is received, it will update the local state to `IDLE` and forward the message to the neighboring agents, if the state was not already reset.

4.6. Experiments

We performed four experiments to evaluate the performance of the different methods in the WPT scenario. For all experiments we generate 200 problem instances and let all methods solve the same problem instance. For the DCOP solvers we had to discretize the potential power levels into $|D| = 20$ levels linearly spaced in $[P_{\min}, P_{\max}]$ where $P_{\min} = 0\,\mathrm{W}$ and $P_{\max} = 10\,\mathrm{W}$. We set the power transmission variables arbitrarily to the

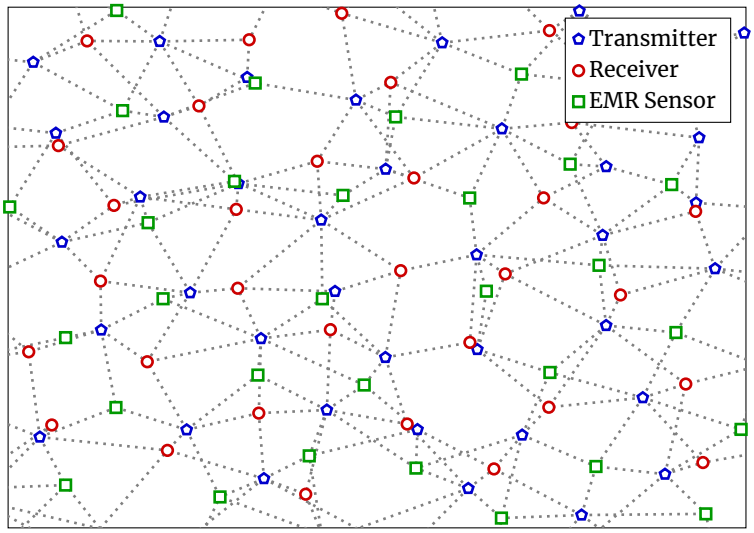

Figure 4.3: An example randomized graph as generated using the proposed methods for the simulation experiments with $t = 70$, $r = 60$ and $s = 50$. There are receivers and sensors with 1 up to 4 transmitters visible.

values $\gamma = \beta = 100$ and $\eta = \rho = 1$. The EMR threshold was set to $\alpha = 0.018\,\text{W}$, and the threshold violation cost $\tau = 10^9\,\text{W}$. For all experiments we define the total solution cost as the sum of all constraint costs as per (4.6) and (4.7), i.e. negative the amount of total power consumption plus τ times the number of sensors where the EMR threshold is violated.

To generate the problem graph we selected the position of the t transmitters using a Poisson point process in 2D space. Subsequently, the r receiver and s sensor locations are selected using the same method, and their locations are then scaled such that they span the same area. We determine the average distance of the third-nearest neighbor in Euclidean space and define that any sensor or receiver within *this* distance, is a neighbor of the transmitter, and hence will receive energy. Using this method we can generate a seemingly natural distribution of nodes with some variation in the number of neighbors that a transmitter has. An example of such a generated graph is shown in Figure 4.3, with $t = 70$, $r = 60$ and $s = 50$.

All experiments were performed in simulation on a laptop with an Intel Core i7-6600U at 2.6 GHz and 16 GB of RAM. [1]

4.6.1. Comparing Solvers

In the first experiment we compare the performance of various DCOP solvers with the centralized solver. For this experiment we generated

[1]For reproducibility of the results all code for the algorithms (Java) is available from https://github.com/coenvl/jSAM/tree/SASO2017, and experimental setups and visualization (MATLAB) from https://github.com/coenvl/mSAM/tree/SASO2017.

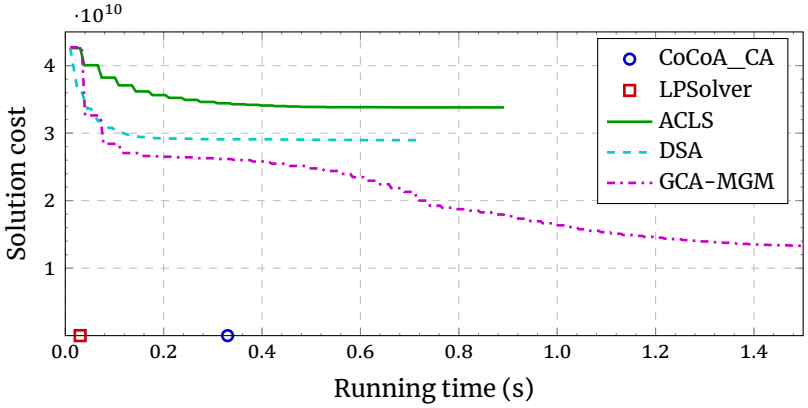

Figure 4.4: Various iterative algorithms fail to find a solution that satisfy the receiver constraints. CoCoA_CA however is capable of finding a valid solution, similar to the centrally computed optimum.

Table 4.1: Numerical results of the DCOP solvers. Colored costs indicate hard constraints being violated—red numbers are consistent constraint violations, and orange numbers indicate incidental violations.

Algorithm	I	S	M*	E*	T
CoCoA_CA	N/A	**−1.932**	1.7	299	0.3
ACLS	27	33×10^9	**1.0**	74	0.3
DSA	16	28×10^9	3.1	**54**	**0.2**
GCA-MGM	117	13×10^9	34.6	210	1.1
Max-sum	49	4×10^7	30.0	312302	321.7

* = $\times 10^3$

problems with $t = 100$, $r = 75$ and $s = 50$. We compare the results of the DCOP solvers CoCoA_CA, ACLS, DSA, max-sum and GCA-MGM introduced in Section 3.2.1 with a centralized LP solver. As a stopping criterion for the iterative solvers we keep track of the reported solution cost. When it has not dropped for 100 iterations we stop the simulation, and report the metrics at the moment at which the solution was first within 1% of best found solutions.

From the results in Figure 4.4 we can see that from all DCOP solvers, only CoCoA_CA is consistently capable of finding solutions that satisfy the EMR threshold constraints. Averaged over 200 experiments, it found a solution that transmits 1.93 W in 0.33 s compared to 2.15 W in 0.03 s for the centralized LP solver. Of course because it is constrained in the exact values of the power levels, its final solution cost is somewhat worse. The max-sum algorithm did perform relatively well, but took very long to solve the problem, hence was left out from Figure 4.4.

In Table 4.1 the numeric results of the experiment are shown. These

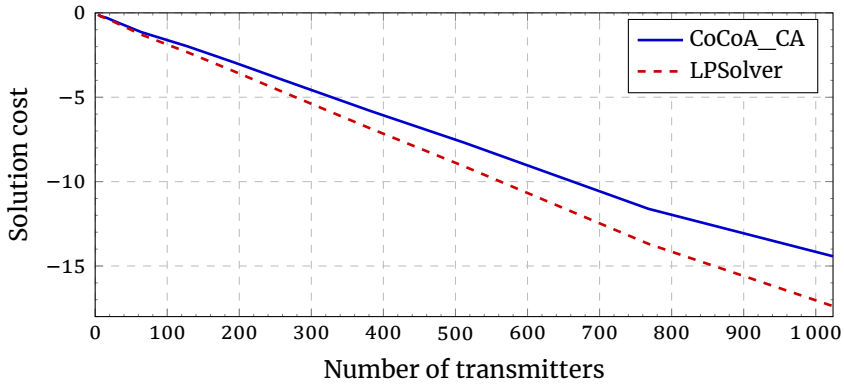

Figure 4.5: CoCoA_CA has the capability of solving even larger graphs, and the distance to the global optimum is linearly dependent on the graph size.

correspond to the results in Figure 4.4, but also show the communication overhead and the number of iterations for the iterative solvers. For every algorithm the amount of iterations (I), the final solution cost (S), the number of transmitted message (M) and the time to solve (T) in seconds. The max-sum algorithm violated constraints 3.5% of the time (probably because in those situations max-sum failed to converge, as it may not converge on cyclic graphs), but if it does not, the mean cost is −2.067 W, which is slightly better than CoCoA's mean result. Only in one instance out of the 200 experiments did GCA-MGM find a solution that did not violate the EMR constraints, whereas DSA and ACLS never did.

4.6.2. Scalability

In the second experiment we investigate the performance of the CoCoA_CA algorithm under varying problem sizes. Fixing all other parameters of the problem we generate instances with increasingly more transmitters (varying between 4 and 1024), and 0.8 times as many receivers and 0.6 times as many sensors.

In Figure 4.5 the solution cost is shown for CoCoA_CA and the LP solver for increasingly large problems. As can be seen, CoCoA_CA performance is only linearly worse than the optimal solution for the problem size. On average the solution by CoCoA_CA yields 85% the amount of power compared with the optimal solution found by the LP solver.

4.6.3. Performance Under Model Error

In the third experiment we validate our hypothesis that using a DCOP approach will keep performing well, even under unpredictable amounts of energy transmitted. In the TESSA charging system, the sensor nodes communicate to the ETs, their actual measurements of the EMR values

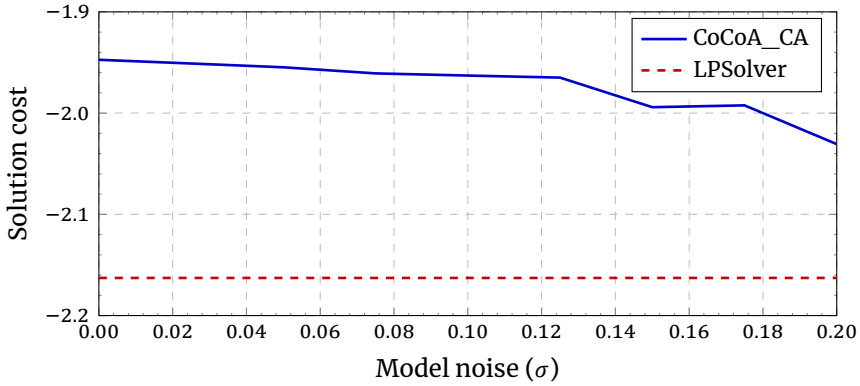

Figure 4.6: The total transmitted power of all receivers, shows that noise in the model is not of large influence to the solution quality of the DCOP solver. The line in the graph shows the theoretical optimum *without* model noise.

which do not always perfectly follow the model as proposed in Section 4.3; for example because of quantization effects. Similarly, the ET nodes can either use (i) the amount of measured harvested power based on the measurements by the ERs or (ii) the predicted total harvested power based on the theoretical model represented by in (4.3).

In all previous experiments we assumed that (i) our theoretical model in Section 4.3 perfectly represents the amount of transmitted energy and (ii) both sensors and receivers perform measurements reflected by (4.3) and (4.4). In order to explore the effect of the measurements on the performance of our system, we introduce an error in the model on how much energy is received by the receivers and the sensors. i.e. for any combination of transmitter i and receiver j the amount of harvested energy is

$$P'_{i\to j} = \epsilon\eta \frac{\gamma}{(d_{ij}+\beta)^2} P_i = \epsilon P_{i\to j}, \tag{4.10}$$

and similarly the amount of EMR measured by a sensor k is

$$E'_k = \epsilon\rho \sum_{i:S_k\in\mathcal{M}_i} \frac{\gamma}{(d_{ik}+\beta)^2} P_i = \epsilon E_k, \tag{4.11}$$

where ϵ is a random noise multiplier from the normal distribution $\epsilon \sim \mathcal{N}\left(1,\sigma^2\right)$: white noise added to the amount of transmitted power from the original model.

In Figure 4.6 the results are shown for the CoCoA_CA algorithm as it solves different problems with an increasing amount of noise, compared to the centralized LP solver. We observe that our solver performs well by continuously satisfying the EMR constraints. The centralized LP solver makes its assignments based on the predetermined

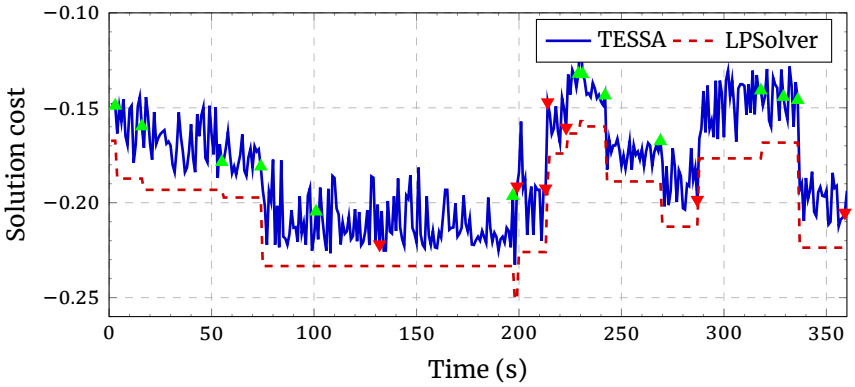

Figure 4.7: We can see that TESSA finds safe solutions, no EMR threshold violations of τ are seen, that are near the optimal solutions found by the LP solver. Upward arrows (green) indicate a random transmitter was added to the environment, and downward triangles (red) indicate the removal of a transmitter.

energy harvesting and EMR model and cannot take into account the actual measurements. Obviously, it is possible to extend the LP solution where sensor measurements are sent to the central solver, removing any model noise. However, that can not longer be considered a *centralized* solution.

In a purely centralized solution, because of modeling and measurement errors, in practical scenarios it is impossible to estimate the actual harvested power and in turn the EMR values [97]. If we apply the solution found by the LP solver in the noisy model, we find that in all instances except where $\sigma = 0$, some sensor constraints were violated, leading to invalid solutions. The solutions of the LP solver without model noise are shown in Figure 4.6 as a lower bound.

4.6.4. Dynamic Environment

In the final experiment we investigate the performance of the TESSA charging system under network dynamics. In this experiment we generate a network with 10 transmitters, 8 receivers and 6 sensors. We run the TESSA charging system, and randomly add or remove agents. Specifically in every second there is a 5% chance that the network will change, and if it does, then half of the times a transmitter is added, and half the time a randomly selected transmitter is removed from the WPTN. The CoCoA_CA algorithm is reset every second.

In Figure 4.7 the total amount of transmitter power is presented for both TESSA, and for the centralized LP solver that calculates the optimal power levels. Observe that when a transmitter is removed from or added to the charging system which is currently charging receivers, TESSA disseminates a `Reset` message (see Algorithm 2) to start the optimization process again. Therefore, the charging network reacts to this disturbance

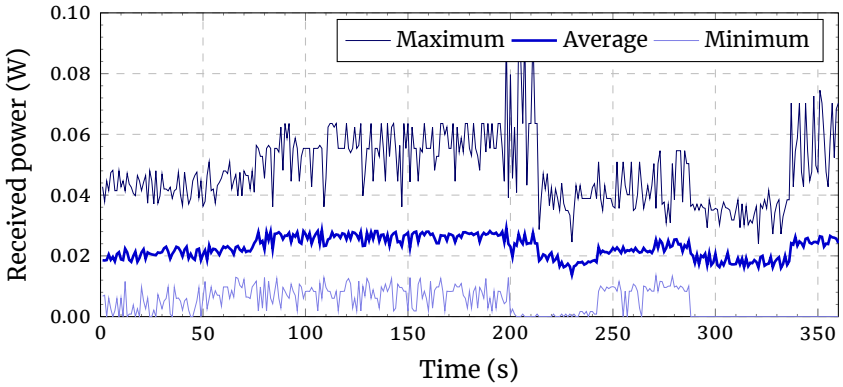

Figure 4.8: Example of tracking the amount of individual received powers for all ER in the dynamic experiment, showing the minimum, average and maximum amount of received power for all ERs.

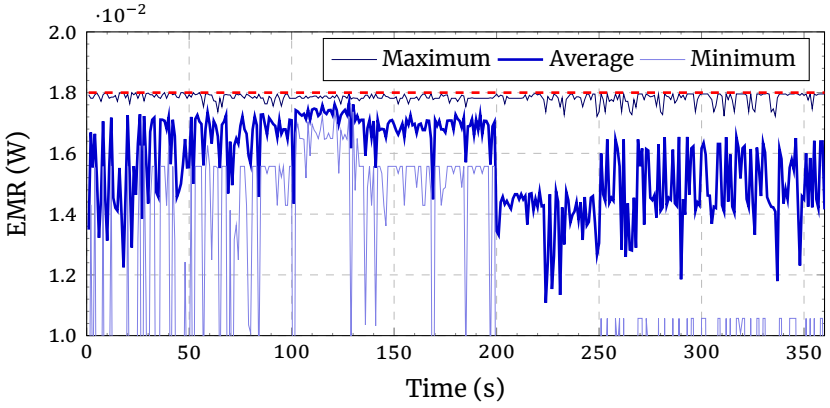

Figure 4.9: The EMR is logged for every sensor in the dynamic environment experiment. Here we show the EMR threshold at 0.018 W (red dashed line), together with the minimum, average and maximum measured EMR of all sensors.

by re-calculating the optimal power levels of the whole transmitters with respect to the EMR safety constraints. Eventually, all transmitters will start transmitting energy according to the new power levels that comply with the new safety constraints in accordance with the new structure of the network.

In Figure 4.8 the amount of received power is shown over time. The minimum, average and maximum amount of power is shown, for the full set of ERs. It is obvious that there is a some spread between ERs, due to the random placement of both receivers and sensors. Receivers that are near one or more sensors, obviously cannot be charged, due to the exposure in the sensors. The EMR levels of the sensors in the dynamic placement

experiment are shown in Figure 4.9; again here we show the minimum, average and maximum exposure of all sensors. From this Figure we can determine that even for the *most* exposed sensor, the maximum EMR threshold is never violated. However, due to the formalization of the optimization problem (4.5), the amount of received radiation of the sensors is maximal, but always less than the EMR threshold.

This network adaptivity is almost impossible to achieve with the centralized LP solver. The reason is that, for each transmitter removal or addition, the whole state of the network—including transmitter power levels, the positions of the receivers and the sensors—should be collected and sent to the central entity that calculates the optimal LP solution. What is more, the results of these calculations should be distributed back to the corresponding transmitters so that they update their power levels. Thanks to TESSA and the CoCoA_CA solver, adaptivity is achieved by only the local interactions among the agents in our system.

4.7. Conclusions

In this chapter we introduced an extension to the CoCoA algorithm that limits the activation of neighboring agents, to avoid simultaneous decisions. This enables the algorithm (called CoCoA_CA) to deal with hard constraints, and also find assignments with even lower solution costs. This new algorithm is the driving force of a new self-organizing wireless power charging system TESSA, for safe wireless power transfer, to ensure that electromagnetic radiation levels never exceed safety guidelines. We formalized the safe wireless charging problem as a DCOP so that any DCOP solver can be used to solve this problem. We compared CoCoA_CA with existing solvers and justified that it is capable of consistently finding solutions that maintain safe levels of EMR. Then, we presented experiments that showed the TESSA charging system is self-adaptive in the sense that it reacts to the network dynamics and always transfers the network to an EMR-safe state, while ensuring a high total transmitted power.

In our experiments we show that CoCoA_CA consistently finds solutions that maintain the EMR thresholds, whereas other DCOP solvers can not. The amount of transferred power was on average 85% of the amount of power transferred in the theoretical optimal conditions, independent of the problem size. The losses can be partially explained by the fact that the solver is incomplete, so better solutions may be possible. Also, because the DCOP solver can only choose a power level from a finite set, whereas in the optimal setting any power level between the minimum and the maximum can be selected. Unfortunately we were unable to find performance guarantees with respect to the optimal solution, other than our empirical results. We also showed that our method was able to find a solution that satisfies the EMR threshold, by communicating

measurements from sensors to transmitters. This makes the method more robust, by not relying on a (potentially unreliable) RF model.

We did not perform any experiments involving hardware implementations, but this would be the reasonable next step in providing safe wireless charging solutions. This will yield valuable information about how well the methods perform with realistic disturbances and practical problems.

In this chapter CoCoA_CA definitely outperformed DCOP solvers from the current literature, but in earlier experiments (e.g. Chapter 3) we have already seen that in other classes of problems this is not always the case. In the next chapter, we will introduce hybrid DCOPs in which we can combine the benefits of different DCOP solvers. This could benefit TESSA, for instance, by initially finding a proper solution using CoCoA_CA, and *then* improving it with other solvers—something that CoCoA by itself cannot.

5 Hybrid DCOPs

5.1. Introduction

In this chapter we continue our work for finding an algorithm that is capable of driving self-organization through DCOPs. In Chapter 3 we introduced CoCoA, which differs from existing state-of-the-art DCOP solvers, in the sense that it does not require multiple iterations, or cycles of negotiations and assignments. Instead, it uses a single step lookahead (SSLA) approach to find a reasonable assignment, and then finishes its execution. We have already shown that this offers very good performance in many use cases, and already provided an extension CoCoA_CA in Chapter 4 that enables it to deal with hard constraints. But CoCoA is non-iterative, which (by definition) makes it impossible to recover from early mistakes. Therefore, the issue of recovery from early mistakes is addressed in this chapter. We will generalize the SSLA approach, as well as study the effect of continuing local search with other algorithms, after a SSLA solver has found an initial assignment.

As we have already shown in earlier chapters, Distributed Constraint Optimization Problems (DCOP) are a method for formalizing and solving problems that have a distributed nature, and in which multiple cooperating agents control discrete variables in order to optimize a common problem. Moreover, a special kind of DCOP can be formulated where agents having a shared constraint may assign different costs for a value assignment; these problems are called Asymmetric DCOP (ADCOP) [56]. In ADCOPs the aspect of cooperation and distributed decision-making is even more important, because (in contrast to DCOP) in ADCOP an assignment leading to a local improvement may deteriorate the global

This chapter is based on the article by C. J. van Leeuwen and P. Pawełczak, *Hybrid DCOP solvers: Boosting performance of local search algorithms*, in Proc. OptMAS (2018) [90].

performance. This makes ADCOPs especially important to consider when designing a self-organizing multi-agent system.

In Section 3.2.1 we introduced a variety of DCOP solves, and stated that we can classify them as either *complete* or *incomplete* solvers. In this chapter we introduce a new class of incomplete (A)DCOP solvers, that combine features of different solvers and combine them into *hybrid solvers*. Specifically, we show that we can use different *initialization* methods for existing (iterative, local search) (A)DCOP algorithms, which has a profound impact on their overall performance. In order to achieve this, we use different initialization methods that are *not* iterative in approach, and are hence very fast in determining a solution. Empirically we show that compared to existing state-of-the-art DCOP solvers, we can reduce algorithm running times *and* improve the solution quality.

5.2. Problem Statement

For the definition and formal problem statement that are addressed in DCOPs, we refer the reader to Section 3.2. There exist already DCOP-solving algorithms, with varying ranges of efficiency in terms of solution quality, computational effort, communication overhead or convergence speed. Selecting the right algorithm for solving the problem at hand depends on the topology, type of constraints and the problem scale; often this is a matter of trial-and-error. Often there are multiple solvers that outperform other solvers on at least one of the performance criteria, and there is no single best strategy. Therefore, we foresee the need to combine different solving algorithms into one solution, and to study which techniques are best put together.

5.3. A New Class of DCOP Solvers: Hybrid Solvers

By combining different algorithms into a *hybrid* solver, we aim to get a method which takes "the best of both worlds". Particularly, we propose to improve the performance of existing local search algorithms by modifying the initialization methods to find the initial value assignment. In the field of Constraint Satisfaction Problems, which is closely related to DCOPs, the approach of using an initialization and repairing it is well known, and can yield great benefits [105]. From the fields of evolutionary algorithms [120], clustering [17], neural networks [33, 146], meta-learning [41] and other machine learning techniques we know that initialization methods can have a great effect on the performance of an algorithm. However, to the best of our knowledge, there has been little to no work on the effects of different initialization methods for (A)DCOPs. In this chapter we will study the effect different initialization methods may have on the performance of (existing) DCOP algorithms in the form of hybrid DCOP solvers.

Definition 5.1 (Hybrid DCOP Solver). We define a hybrid DCOP solver as a solver which executes sequentially other (existing) DCOP solvers.

5.3.1. Motivation for a Hybrid DCOP Solver

Most (if not all) local search DCOP algorithms use an initial random assignment for all the variables, which will be iteratively improved upon. Instead, an initial assignment can be computed by a non-iterative DCOP algorithm, such as a simple greedy algorithm, or a more elaborate greedy algorithm such as the one introduced in [89]. Since these methods will assign a value only once, and then terminate, they quickly provide a good initialization assignment from which one can start another DCOP method.

We hypothesize that the combination of different initialization methods for iterative algorithms in DCOP solution search, will be beneficial because of two effects:

1. *Solution quality improvement over initial assignment*: most probably a simple initialization method will find a sub-optimal solution, and many local search algorithms will be able to improve it. Algorithms that are known to provide monotonically decreasing solution costs (any algorithm that uses a coordinated change approach, e.g. MGM-2, ACLS, GCA-MGM) are guaranteed to find better (or equal) solutions compared with the initial value assignment; and

2. *Increased convergence speed for local search algorithms*: DCOP algorithms that use local search will most likely converge faster when a good solution is used for initial DCOP value assignment. This will lead to a shorter total running time for algorithms that are initialized with a better assignment.

5.4. Initialization of DCOP Solvers: Classification

In our experiments we combine different *initialization methods* with existing *local search algorithms* which we shall also refer to as *iterative methods*. Since the aim of this study is to improve the solution quality *and* convergence speed of solvers, we do not take into account complete solvers. To understand why only certain combination of DCOP solvers improves the solution, we need first to classify (i) initialization methods, and (ii) types of DCOP iterative solvers.

5.4.1. DCOP Classification: Initialization Methods

The initialization methods of DCOP algorithms can be classified into the following categories.

- **Random** A de facto standard method for all DCOP solvers. It does not take into account any constraints and starts with the random variable assignment.

- k **Step Look-ahead:** We define a look-ahead initialization algorithm as the one in which one randomly chosen initial node is triggered first, and only after it has chosen a value it will activate its neighbors. When choosing a value, it takes into consideration the effect on all of its neighbors that are reachable within k steps (edges or hops). Three special cases of k step look-ahead are already known:
  - **Zero Step Look-ahead (ZSLA):** A zero step look-ahead algorithm ($k = 0$) is the one in which an agent optimizes only for the constraints it is directly involved in. Such algorithms are also referred to as *greedy, breadth-first* algorithms. This is essentially an asynchronous version of the DSA algorithm [155] with $p = 1$;
  - **Single Step Look-ahead (SSLA):** A single-step-look-ahead algorithm ($k = 1$) is defined as one in which an agent optimizes not only for the constraints it is involved in, but also the constraints its one-hop neighbors are in. One such algorithm is the CoCoA algorithm and its variants CoCoA_UF introduced in Chapter 3 and CoCoA_CA introduced in Chapter 4;
  - **Max Step Look-ahead (MSLA):** If k is equal to the height of the graph's minimal spanning tree, and the algorithm would be started at the root of the spanning tree, the algorithm becomes a complete algorithm, and is in fact equivalent to DPOP [116]. Strictly this method should not be considered a initialization strategy, since it is already a complete method.

5.4.2. DCOP Classification: Existing Iterative Methods

Classifying iterative methods used in DCOP solvers we can divide them into two main groups.

- **Symmetric DCOP Solvers:** Solvers that only take into account symmetric constraints, which assume costs are the same for all agents involved. These include DSA [155], MGM and MGM-2 [100] and generalized DBA [113];
- **Asymmetric DCOP Solvers:** Solvers that also take into account asymmetric constraints; here costs may be different for the involved agents. Examples of such solvers include GCA-MGM [57], and ACLS [56] with its new version ACLS-UB (which is also a novel contribution of this work and described in Section 5.4.3).

Remark. We are naturally aware of another popular DCOP solver: the max-sum algorithm [38] or any one of its variants. However, max-sum is unable to utilize the benefit of initializing, as it tries to approximate the global utility of any value, and uses this to determine the best variable assignment. There are extensions of max-sum that are able to build upon an initial assignment by using value propagation [159]. In a

Algorithm 4 ACLS-UB Algorithm

On agent i (A_i) when activated
1: $X_i \leftarrow$ chooseRandomValue()
2: **while** (no termination condition is met) **do**
3: send X_i to all neighboring agents $\forall A_j \in \mathcal{M}_i$
4: $v \leftarrow$ chooseRandomValue()
5: send v to $\forall A_j \in \mathcal{M}_i$
6: wait for incoming constraint cost δ_j from A_j
7: $\Delta_i \leftarrow \sum_j^{j \in \mathcal{M}_i} \delta j$
8: **if** $\Delta_i <$ current cost **and** random$[0, 1] < p$ **then**
9: assign $X_i \leftarrow v$
10: **end if**
11: **end while**

On A_j when receiving v from A_i
12: send constraint cost $\delta_j(X_j \cap v)$

recent paper [21] the effect of initialization was studied in variant called max-sum_ADSSVP. The authors find that the timing, and approach to initialization has a great effect on the performance, both in terms of solution quality and convergence speed. For our evaluation however, we leave max-sum out of the comparison, and refer to max-sum_ADSSVP to provide a complete overview of different hybrid algorithms.

5.4.3. Novel Iterative DCOP Solver: ACLS-UB

In addition to existing DCOP algorithms listed above we introduce a variant of the ACLS, denoted as *unbiased* (ACLS-UB).

In the original ACLS algorithm [56], at every iteration an agent chooses a variable assignment that would lower its local costs and proposes it as a new value to its neighbors. Neighbors respond with the effect on their side, after which the proposition which has the best effect on the regional cost function is selected. In the ACLS-UB algorithm a value assignment is proposed from *all* possible values, instead from the subset that improves its local state. The ACLS-UB algorithm is described using pseudo code in Algorithm 4.

ACLS-UB works by iteratively proposing a random value from its domain D_i, and sends that value to its neighbors. The neighbors respond by sending the effect of the assignment on their local costs, taking into account all known value assignments. When these local effects are received by the initial agent, it sums over all received effects, and assigns the proposed value with probability p only if it will reduce the current local cost.

Relation of ACLS-UB to Other Solvers

The main difference between ACLS and ACLS-UB is in line 4 of Algorithm 4, where any random value is picked from the domain. In the long run, the effect of this pick is that the effect of all values from the domain are used to retrieve the induced effect on the neighbors' local cost.

Intuitively ACLS-UB works very similar to CoCoA [89, 93], with the major difference that CoCoA operates in one single iteration instead of iteratively trying different values. Another difference is that in ACLS-UB the neighbors will send back the value of the constraint cost, whereas CoCoA will send back the lowest induced cost for any assignment in conjunction with the proposed value and the CPA. This extra look-ahead is not efficient in ACLS-UB, since the next-hop neighbors *will* in fact already have an assignment, and the lowest cost will be too optimistic. Note that the unique-first approach of CoCoA is not required in ACLS-UB, as it can easily recover from any earlier suboptimal assignments in later iterations, whereas CoCoA cannot.

5.5. Hybrid DCOP Solvers: Introduction and Initial Results

In order to understand whether there is any benefit from hybrid solvers, we performed the following experiments[1]. For any problem, we initiate 200 problem instances which are initialized by three methods: *random*, *ZSLA* (i.e. greedy) and *SSLA* (i.e. CoCoA) and subsequently solved by other DCOP solvers (depending on the experiment). We report the average result of all problems. We assume a solver has converged when no better solutions have been found for more than 100 iterations, and define the moment of "convergence" as the first iteration in which the solution was within 1% of the minimal solution. In this way we can compare the convergence speed of different algorithms, and do not have to specify the number of iterations beforehand, in a way similar to the any-time solution as proposed in [157].

As performance metrics we will score solvers on the following metrics:

- **The number of iterations required to converge**, denoted as I;
- **Final cost of the solution when converged**, denoted as S;
- **Number of transmitted messages during the run**, denoted as M;
- **Number of constraint evaluations**, denoted as E; and
- **Running time until convergence**, denoted as T in seconds.

[1] For reproducibility and validation of our results, all (Java) code for the algorithms is available at https://github.com/coenvl/jSAM/tree/OptMAS18, and for the experimental setups (MATLAB) at https://github.com/coenvl/mSAM/tree/OptMAS18.

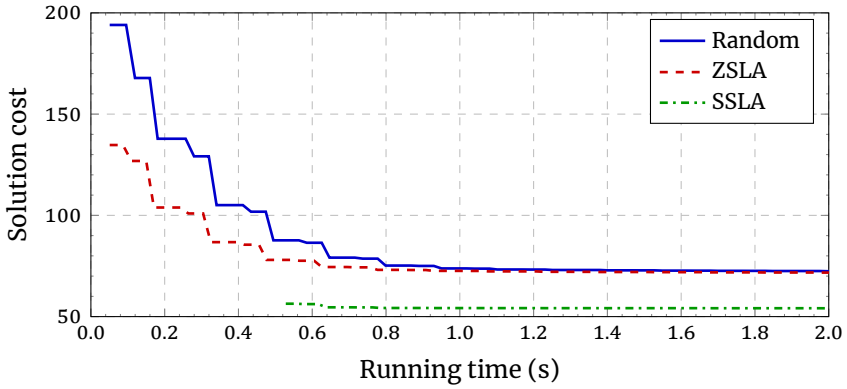

Figure 5.1: In a three-color graph coloring experiment, when using an SSLA for initialization, the MGM-2 algorithm improves both speed and solution quality.

Table 5.1: Graph coloring experiment results

Algorithm	I	S	M*	E*	T
Random_DSA	157	49	10.4	362.9	3.5
ZSLA_DSA	164	49	10.7	381.6	3.7
SSLA_DSA	115	47	**14.5**	381.3	3.1
Random_MGM-2	55	72	26.3	120.2	1.7
ZSLA_MGM-2	42	71	21.0	94.4	1.3
SSLA_MGM-2	7	54	12.2	121.0	0.7

* = $\times 10^3$

The constraint evaluations are indicative of the computational complexity and can also be referred to as Non-Concurrent Constraint Checks (NCCCs) [104].

5.5.1. Experiment Results

Experiment 1: Symmetric DCOP

We use a (symmetric) graph-coloring problem with three colors, which have to be assigned to 200 variables. The constraints between the variables are chosen as the nodes were connected via a Delaunay triangulation, when the variables are points chosen randomly on a two-dimensional plane. The results of MGM-2 ($p = 0.5$) is shown in Figure 5.1, of which the numeric results are shown in Table 5.1 together with DSA (variant C, with $p = 0.5$).

Experiment 2: Asymmetric DCOP

An asymmetric problem is chosen where the constraints are created using a scale-free graph generation method [5], and are assigned semi-random

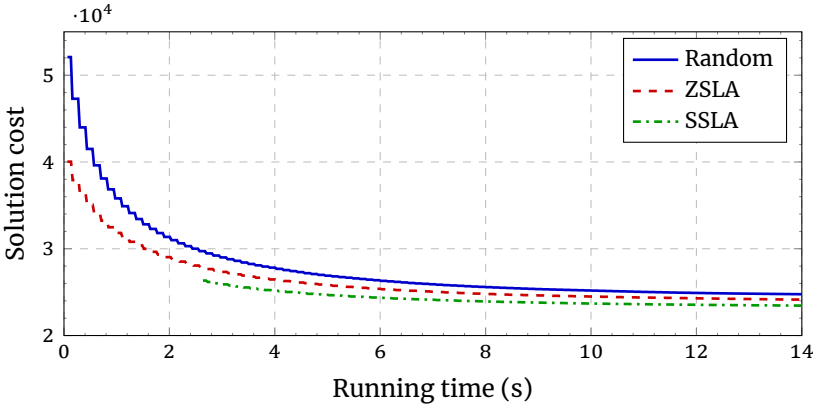

Figure 5.2: In a semi-random experiment (Section 5.5.1, Experiment 2), when using an SSLA for initialization, the ACLS-UB algorithm shows faster convergence to a better solution.

5

Table 5.2: Semi-randomized asymmetric experiment results

Algorithm	I	S*	M*	E*	T
Random_ACLS	95	26.4	151	1 321	4.3
ZSLA_ACLS	75	26.1	121	995	3.5
SSLA_ACLS	48	24.2	107	10 571	**4.9**
Random_ACLS-UB	333	24.7	531	5 747	14.8
ZSLA_ACLS-UB	299	24.1	477	5 164	13.3
SSLA_ACLS-UB	207	23.5	358	12 963	11.9
Random_GCA-MGM	1 154	21.7	1 389	5 557	55.1
ZSLA_GCA-MGM	1 022	21.6	1 232	4 935	48.8
SSLA_GCA-MGM	781	22.0	970	13 599	40.0

* = $\times 10^3$

asymmetric costs such that there is a high probability that a conflict of interests occurs. This problem is created specifically to benchmark asymmetric problems, and is described in more detail in [56, Section 5.2]. The result of the ACLS-UB ($p = 0.5$) algorithm is shown in Figure 5.2, and the results of all algorithms is presented in Table 5.2.

Hybrid Solvers—Discussion of Initial Results

Based on the results from the first two experiments, we see that local search DCOP solvers using initialization methods other than random, reduced the required number of iterations before they converged, their execution time and communication overhead, but surprisingly also found a final solution with a lower cost. The only two exceptions (marked in

bold in Table 5.1 and 5.2) are (i) when using the SSLA with DSA, in which case the messages of the SSLA increases the very low communication overhead of DSA, and (ii) when using an SSLA with ACLS, in which case the added run time of the SSLA increases the convergence time. The speed performance gain can be easily explained: a better initialization will reduce the amount of variable "tweaking" needed. However, how come the final solution is also *lower* and solution result dependent on the initialization method? In the following sections we will investigate this using three hypotheses to explain this phenomenon:

- **Hypothesis 1:** A lower initial solution will always lead to a lower final solution;
- **Hypothesis 2:** Using initialization methods other than random increases the explored portion of the solution space;
- **Hypothesis 3:** The initialization method itself finds a starting point in the search space, from which a lower local minimum is reachable.

Let us experimentally verify these three hypotheses in detail.

5.5.2. Hypothesis 1: Solution Cost Correlation

The SSLA algorithm finds a lower initial cost than the ZSLA initializer, which in turn finds a lower cost than a random assignment. Hence, the first hypothesis is that a lower initial costs will (on average) lead to lower final costs. The initial state is known to be of great influence on the final solution, and a correlation between the initial cost and the final cost could explain why ZSLA or SSLA initialization methods lead to better final solutions.

To test this hypothesis we performed an experiment by repeatedly invoking the algorithms on the exact same problem setup, but with different random initializations. We gather information on the cost at initialization and of the eventual outcome. In Figure 5.3 we show the minimum, average, and maximum results of the algorithms solving 200 instantiations (the same problems for every algorithm) of three color graph coloring problems with a Delaunay graph of size $n = 100$. For this small experiment we only compare the DSA algorithm instantiated with a randomized approach with an algorithm that uses CoCoA_CA for initialization.

From this experiment we see that for some iterations we *do* find a solution with the random strategy which is as good as the SSLA-initialized solution, however on average the final solution is worse. The spread of the initial and the final solution corresponds with the statistical spread of the random assignments, and some random initialization lead to better final solutions than others. If we look at the correlation between the initial cost and the final cost of the individual runs, then we find that there is no correlation between the cost of the initial random assignment and

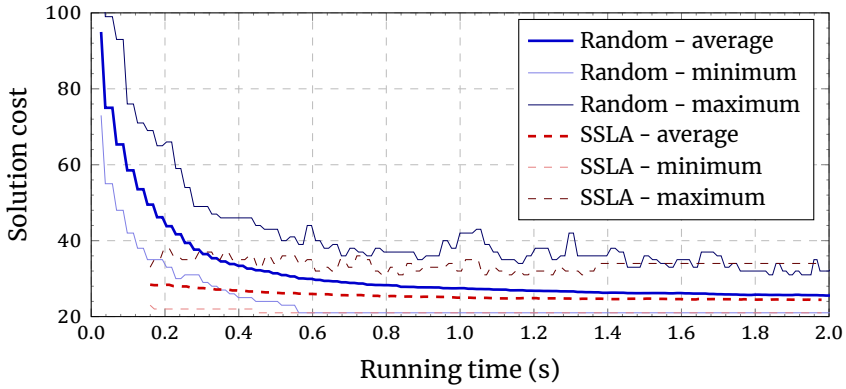

Figure 5.3: Starting the DSA algorithm from a different starting point will lead to a different outcome. This graph shows the minimum, average and maximum solution costs during the experiments.

5

the final minimal cost. The Pearson correlation coefficient between the solution cost at the beginning and the end is 0.15. With these results we *reject hypothesis 1*.

5.5.3. Hypothesis 2: Increased Solution Space Exploration

A DCOP problem is generally a matter of solution space exploration. The solvers that are capable of effectively searching a larger portion of the solution space, will reasonably find a better final solution than solvers that cannot. Since local search algorithms search only a small fraction of the search space, the increase in search space exploration by a SSLA may be of large influence. Put differently, a better overview of the trends in the solution space may lead to insights as to where the best optimum lies. If we can show that the solvers using SSLA explore a larger part of the solution space than randomly initialized solvers, this may explain why they find solutions with the lower final cost.

To verify this, we construct an experiment in which we captured the CPA *every* time a constraint check is performed, so that we can store every explored value assignment. We did observe that SSLA searches a slightly larger portion of the solution space than ZSLA, which in turn searches a larger part of the solution space than random. However, the SSLA algorithm sometimes searches a *smaller* part of the solution space, largely because its successor algorithm converges so quickly. Therefore, with these results we *reject hypothesis 2* as well, also because the differences are so marginally small that they cannot explain the significant effect on the results.

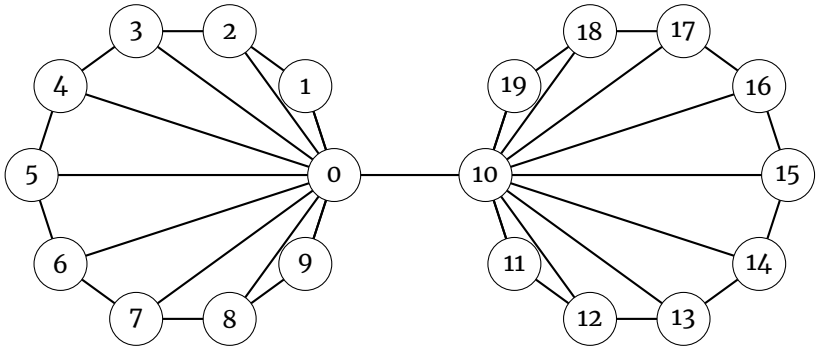

Figure 5.4: An example graph in which two dense clusters of nodes are connected by a single bridge.

5.5.4. Hypothesis 3: Selection of Starting Point

The initial assignment determines the area of the solution space that is reachable through local search. It is possible that SSLA is capable of finding initial assignments that have relatively good local minima. To explain what we mean by this, let us sketch the following example.

Definition 5.2 (Bridge edge). A *bridge edge* is defined as an edge that, when removed from the graph, the graph will no longer be connected.

Suppose we have a graph with high modularity, meaning it consists of clusters of densely connected nodes, which are connected through relatively low number of bridge edges. The nodes on these bridges could initially induce a high performance penalty; and it may be impossible for a local search algorithm to escape from these expensive assignments because of the many low cost constraints around it on the surrounding nodes.

As a minimal example suppose we have a three color graph coloring problem with a graph as shown in Figure 5.4. As we see node 0 and 10 have a constraint with many other nodes, and are connected to one another. If through some unfortunate random assignment, they are both given the same initial color (for example red) and the surrounding nodes are mostly other colors, then we expect that no local search algorithm will change that initial assigned color. This expectation is confirmed through a series of experiments in which we use the graph as depicted in Figure 5.4, letting the algorithms solve the graph-coloring problem. In one set of the experiments, the agents are hardwired to initialize with an assignment in which $X_0 = X_{10}$, and $\forall_{i \neq 10} X_i \neq X_0$. As we can see in Figure 5.5 for that subset, the local search algorithm is unable to find a solution in which this constraint is resolved. We also see, and in fact can guarantee that an SSLA algorithm will never assign the same color to endpoint vertices of a bridge, and will thus lead to better solutions.

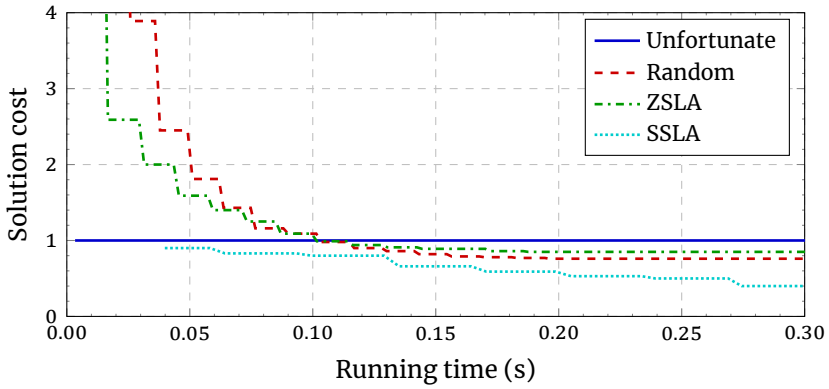

Figure 5.5: The results of the GCA-MGM algorithm trying to solve the graph coloring problem in graph from Figure 5.4, with various initialization strategies. The "*unfortunate*" strategy is hardwired to get a conflict on the bridge.

Proposition 5.1. *A SSLA will never assign the same color to the endpoints of a bridge.*

Proof. (*Sketch*) When an SSLA starts, any random agent is activated first. If either bridge endpoint is selected first, then logically they will not be activated simultaneously. If any other node is selected, then this node will execute the algorithm and select a random initial assignment. After that it activates all of its neighbors, which will execute the algorithm, until at some iteration the first bridge endpoint is selected. At this moment the other endpoint cannot be activated, or the edge would not have been a true bridge.

Because the algorithm is active in one bridge endpoint, but never in both at the same time, one must assign a value before the other. Moreover, when an endpoint of the bridge eventually has to assign a value, no nodes from the other component can be assigned a value yet, because the bridge is the only connecting edge. Therefore, when the second bridge endpoint is activated it will only have the first endpoint as a constraining value, and will thus always pick a different value. □

Although this exact order of events will not hold for pseudo-bridges that connect clusters within a graph, there will be an ordering in which the nodes will be activated, as long as the detour path between the vertices on the pseudo-bridge is longer than three. In many graphs with high modularity, the values of nodes in the bridging constraints will therefore be chosen with low costs, and the coloring within the clusters can simply be permuted. Therefore, the local search algorithms that continue from these solutions are generally of higher quality, than those from random initial assignments.

If this final hypothesis is true, then we expect some different results in the performance of different types of graphs, especially for various densities. We would expect the benefit of an SSLA to decrease with problem graphs of higher densities, since in these graphs bridges or pseudo-bridges occur less frequently.

5.6. Graph Density

In our final experiment we use once more the graph coloring problem with a domain size $|D| = 3$, and instantiate randomly connected graphs with $n = 200$ with nine varying densities between 0.01 and 0.3. We generate 50 graphs of every density, let the different solver combinations (initialization and iteration) solve the same graphs, and report the average performance of all instances. The convergence criteria were identical to the experiment described in Section 5.5.

In Figure 5.6 we show the averaged results for the MGM-2 solver, when solving the graphs with different densities. We can indeed conclude that for graphs with a low to medium density (up to 0.1), there is a benefit using SSLA initialization for the final solution cost. The increase in convergence speed deteriorates much faster, since the complexity of the SSLA is exponential with the node degree, and this increases with graph density. Note, the time the SSLA initialization takes is shown as the starting point of the line. For other local search algorithms (DSA, ACLS, GCA-MGM), similar results were found.

5.7. Conclusions

In this chapter we introduced a new class of hybrid algorithms which combine the benefits of different DCOP algorithms. Particularly, we studied the effect of different initialization strategies on the performance of different DCOP algorithms. We found that using the combination of non-iterative SSLA algorithms with iterative local search algorithms, not only combines the fast convergence of the SSLA with the eventual better solution quality of the iterative approach, but that using another initialization improves the quality of the final solution.

Two possible hypotheses that could have explained these observations were proposed and rejected: (i) a correlation between the initial cost and the final solution cost does not exist, hence the effects are not from finding *any* initialization with a low cost, and (ii) somewhere in the complete solution space is the optimal solution, but using SSLA does not significantly increase the searched solution space. Instead, we hypothesize that using an SSLA (such as CoCoA and CoCoA_CA) selects an initialization that is in a region of the solution space that has a lower local minimum than the local minimum from a random starting point in the solution space. Furthermore, we propose that this is caused by a reduction of conflicting values assigned on bridge vertices. In our final experiment we show that the ef-

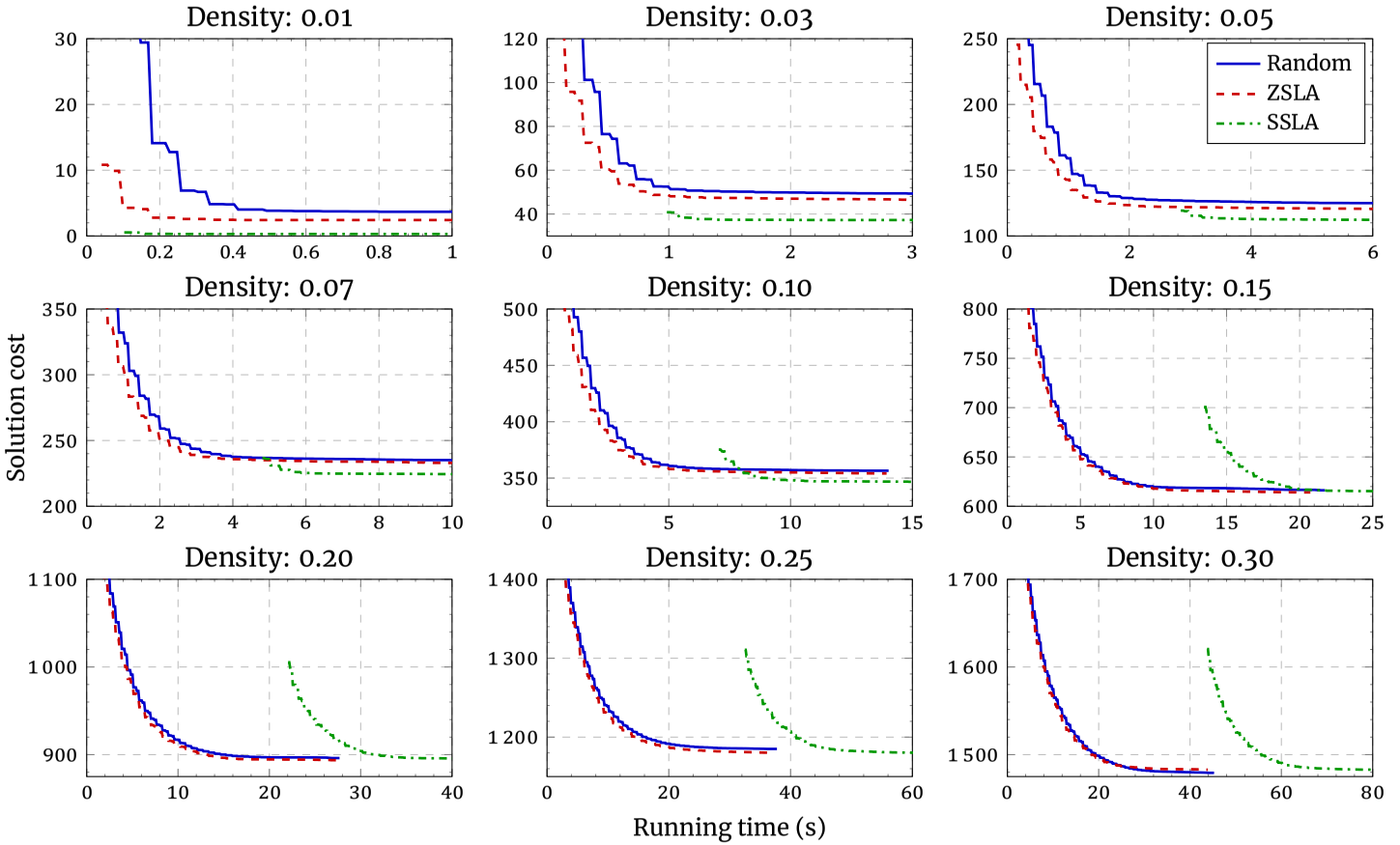

Figure 5.6: The solution cost versus the time of the MGM-2 algorithm when solving random graphs of different densities shows the impact of initialization methods. Up to a density of 0.1 there is a clear improvement on the solution cost.

fect is most abundant on low density graphs, in which (pseudo-)bridges are more present, and the solution cost of the search space is less homogeneous. Not only did we determine in what situations an SSLA is effective, we also gained some insight into why an SSLA performs the way it does.

Our hybrid approach seems well suited for applications in which the problem graphs are not too dense (i.e. up to a density of 0.1), and where both convergence speed and solution cost are required. In fact, we may use it as a general strategy for initial value assignment instead of using random values in problems with low graph densities.

With this chapter, we conclude the work on CoCoA and using DCOPs as a coordination mechanism for collective decision-making. With the addition of hybrid solvers, we have a strategy for effecting self-organization, with a minimal communication and computational overhead using non-iterative SSLA, combined with an appropriate (depending on the type of problem) iterative DCOP solver. However, not all problems can be cast as a DCOP (efficiently). In the next chapter we present a use case for self-organization in the energy domain. Even though such a problem can and has been formulated as a DCOP, the resulting problem graph would be fully connected (density of 1), and hence the solution would not scale well [44]. Instead, we will look at a different decision-making strategy for achieving self-organization based on a decentralized market mechanism.

6

Self-organizing Smart Grid Planning

6.1. Introduction

In this chapter we propose a method to realize self-organizing planning in the energy domain. Specifically, we define an agent-based approach for planning the power consumption or production program for a smart grid of connected households.

Previously, in Chapter 3 and Chapter 4, we introduced some methods to coordinate decision-making using DCOPs. In such a solution all agents that are participating in the system need to coordinate their decisions with all agents that they affect. i.e. with all agents they share a constraint with. We have also shown in Chapter 5 that the performance of (Hybrid) DCOP solvers drastically deteriorate with increasing problem graph densities, which confirms what was found in other studies [22, 42, 44, 85]. In this chapter, instead of using DCOP as a decision-making formalism for driving self-organization, a market-based strategy is proposed for cooperatively finding solutions that are feasible for all involved agents. A hierarchical multi-agent based approach is proposed, that uses local pricing to incentivize device controlling agents to schedule their usage in order to follow an agreed upon target profile and mitigate congestion. As we remarked in Chapter 1, not all agents in a self-organizing multi-agent systems need to be peers. In this chapter a hierarchy tree is used where at the top a "market operator" gathers information from all agents, and makes sure the *common* constraint is satisfied. Intermediate agents make sure the physical grid constraints are respected, and device agents at the bottom represent any end-user constraints or preferences. The resulting

This chapter is based on the article by C. J. van Leeuwen, J. Stam, A. Subramanian and K. Kok available at https://arxiv.org/abs/2009.02166

algorithm is not as re-usable as the solution presented in earlier chapters, but is tailored to the problem at hand: finding a feasible dispatch in the power grid.

Three trends set a challenge for future power grids. Firstly, the transition towards sustainable energy sources leads to more renewable energy, but also to a larger fraction of unpredictable and intermittent production. Secondly, the electrification of various systems such as transport (electric vehicles), heating (heat pumps), and in general an increase of electricity-consuming devices leads to a huge growth of power consumption. And thirdly, the distribution of energy generation leads to a very different pattern in the load of the power transmission grid, than it was designed for.

The control of a vast number of small power units, both consuming and producing, is extremely difficult to do completely top-down, so a centralized control strategy cannot be used [81]. At the same time the power infrastructure is aging, and was not built for the emerging pattern of distributed *prosumers* [51]. This is why we need self-organizing control algorithms to schedule the use of electric devices while taking into account constraints of the power distribution infrastructure. This aids *distribution system operators* (DSO) and *transport system operators* (TSO) to maintain power balance and make sure there is no congestion, i.e. the grid capacity is not overloaded.

6.2. Problem Statement

The problem at hand is a variation of the economic power dispatch problem [2, 124], where the operation of a set of generators is optimized, such that power is provided to consumers in the most cost-effective manner. Traditionally this problem would include controllable generators (power generation plants), constraining transmission resources (transformers, stations, cables), and end-consumers having a static load. Currently, with distributed energy resources, and demand response—the possibility to control the load of consumers using flexibility of smart devices—the problem changes significantly.

A formal definition of the problem is as follows:

$$\min_{\boldsymbol{x}} \quad \sum_{i \in \mathcal{N}} J_i(\boldsymbol{x}_i), \tag{6.1a}$$

$$\text{s.t.} \quad \sum_{i \in \mathcal{N}} \boldsymbol{x}_i = \boldsymbol{\Theta}, \tag{6.1b}$$

$$\mathbf{A}\boldsymbol{x} \leq \boldsymbol{b}, \tag{6.1c}$$

$$x_i^{\min} \leq \boldsymbol{x}_i \leq x_i^{\max}, \qquad \forall i \in \mathcal{N}, \tag{6.1d}$$

$$\mathcal{C}_i(\boldsymbol{x}_i) = 0, \qquad \forall i \in \mathcal{N}, \tag{6.1e}$$

Table 6.1: Notations and parameters used in this chapter

Symbol	Description	Typical value	Unit
$\mathbf{A}$	topology matrix of child relations		
α	forecast accuracy	0.9	
$\boldsymbol{b}$	vector of congestion thresholds	$[2, 3, \dots, 3.5] \times 10^4$	W
b_i	congestion threshold of i	3×10^4	W
$\mathcal{C}_i$	constraint function of i		
$\mathcal{C}_i^{\mathrm{Z}}$	constraint function of i of type Z*		
γ_i	PV operation cost of i	0.2	
$\boldsymbol{e}_i$	$1 \times T$ vector of energy of i	$[0.0, 1.1, \dots, -0.3] \times 10^3$	Wh
e_{it}	energy of i at t	200	Wh
$e_i^{\min}$	minimum energy of i	0.0	Wh
$e_i^{\max}$	maximum energy of i	2×10^3	Wh
$\boldsymbol{\epsilon}$	$1 \times T$ vector of errors	$[0, 11, \dots, -225]$	W
$\epsilon^{\max}$	upper bound on the error	10^{-3}	W
η_i	storage efficiency of i	0.9	
$\boldsymbol{\Theta}$	target power of the cluster	$[1.9, -1, \dots, 2.6] \times 10^4$	W
i	an agent index	$0, 1, \dots, n$	
J_i	cost function of i		
J_i^{Z}	cost of i of type Z*		
λ_i	storage leakage of i	360†	W
$\mathcal{M}_i$	set of children of i		
$\mathcal{N}$	set of all agents		
$\boldsymbol{\rho}$	$1 \times T$ vector of prices	$[0.5, 0.3, \dots, 0.8]$	
T	time horizon	24	
t	power time unit (PTU) index	$0, 1, \dots, T$	
τ	duration of a PTU	1	h
$\boldsymbol{x}_i$	$1 \times T$ vector of scheduled powers of i	$[1.1, -0.5, \dots, 0.4] \times 10^3$	W
$\bar{\boldsymbol{x}}$	$1 \times T$ vector of aggregated powers of child nodes	$[24.0, -18, \dots, 8.7] \times 10^3$	W
$\boldsymbol{x}_i'$	$1 \times T$ vector of expected powers of i	$[1.0, -0.4, \dots, 0.8] \times 10^3$	W
$\hat{\boldsymbol{x}}$	$1 \times T$ vector of average historic powers	$[0.8, 0.6, \dots, -0.5] \times 10^3$	W
x_{it}	scheduled power of i at t	-1.7×10^3	W
$x_i^{\min}$	minimum power for i	-2×10^3	W
$x_i^{\max}$	maximum power for i	4×10^3	W
ψ_i	heat pump coefficient of performance (COP) of i	4.0	

Note that powers are indicated from the grid perspective. That means positive powers indicate power going from the grid to the device, and negative powers are from the device back to the grid.

* Z indicates a type of agent, which can either be MO for market operator, CO for congestion, LOAD for a static load agent, PV for an agent with photovoltaic solar panels, or ST for a storage agent.

† 360 W is chosen as an example thermal leakage of a house, which can be considered as an energy storage vessel.

with all parameters defined in Table 6.1, the objective function as defined in (6.1) is the same as for the traditional economic dispatch, which can be summarized as: to find a set of electrical powers $\boldsymbol{x}_i$ for every device $i \in \mathcal{N}$ in the grid, that minimizes the sum of all costs $J_i(\boldsymbol{x}_i)$. In the original dispatch problem defined, $\mathcal{N}$ defines only the producers. However, in our problem formulation (6.1) $\mathcal{N}$ denotes the full set of devices in the grid, including *producers and consumers*. This means that contrarily to the traditional economic dispatch problem, not only the generators are controllable, but also the flexible loads of end consumers.

Note that the costs in (6.1a) do not necessarily refer to the financial cost of a dispatch. Rather, this generic model only optimizes some *social welfare*, which defines a desirable outcome for all participants involved. Depending on the situation at hand, one could minimize the amount of greenhouse gasses emitted, or maximize the amount of renewable energy used. However, for the rest of this chapter the costs are defined as the energy losses of the devices. By minimizing the amount of energy losses we attempt to find a dispatch that is both economic and sustainable. Similarly, in this chapter when we refer to the *price* of energy, this does not necessarily refer to a financial price with a real currency, instead we will refer to a more abstract concept of the price of energy, which is used simply as a steering signal.

Let us discuss the constraints of (6.1), i.e. (6.1b–6.1e). The constraint (6.1b) states that the sum of all powers in the system has to meet a specific target $\boldsymbol{\Theta}$. In a balanced (island) grid this target has to be zero for every PTU, but in a connected grid this target is the contracted load with the transmission operator. Put differently, $\boldsymbol{\Theta}$ is the net power input of the grid to the rest of the world.

Constraint (6.1c) defines the limitations of the physical power grid; a topology matrix A specifies which device is connected to which grid components, and b is the rating of those components—the maximum amount of power that it can safely transmit.

Constraints (6.1d) and (6.1e) represent the constraints of the devices in the grid. Specifically, constraint (6.1d) states that a device cannot produce or consume more power than it physically can, which is represented by lower and upper bound, $x_i^{\min}$ and $x_i^{\max}$, respectively. However, (6.1e) also takes into account another *private* constraint $\mathcal{C}$. We refer to this constraint as private, as it only concerns the state of a single device, and may concern information that should not be shared with other parties. E.g. an end-user should not need to share its intent to run his or her washing machine with its neighbors. This constraint differs per device, and specifies device limitations such as a battery that cannot hold more than a certain amount of energy, and cannot discharge when already empty. Flexible loads also may have constraints considering the time at which they can turn on or off based on the consumer's settings. We will elaborate on these constraints, as well as the cost functions of the device agents in Section 6.3.2.

6.2.1. Related Work

In existing studies, different strategies are used for energy management. Four different main categories are defined in [81] based on whether there is a distributed aspect of decision-making and whether there is one- or two-way communication. One of the strategies defined in [81] uses *Transactive Control*, where distributed systems decide locally on their device management, using two-way communication in a market-based control scheme. The authors compare this approach with traditional top-down switching, price-reaction and centralized optimization strategies and show that transactive control is capable of using the full flexibility potential of the smart grid devices, while maintaining the end-user privacy. This claim is consistent with earlier studies [3, 151] that have shown that using a market-based control in a multi-agent system can provide equally optimal results as a centrally optimized system, under certain conditions.

Multi-agent based methods are already getting attention in the smart grid domain due to many desirable properties: robustness, user-friendliness, attack resistance and scalability [26, 119]. The economic dispatch problem is very well suited to be represented using multi-agent systems [16]. There are many studies that use a multi-agent based approach to model and solve the problem, such as the two-way message passing agents using consensus algorithms to find an allocation that is optimal [63]. Other methods use a completely decentralized method involving reinforcement [98], utility maximization [95] or model predictive control [140].

In [142], a strategy is proposed to schedule the power consumption and production of generators and loads based on a method called *Negotiated Predictive Dispatch*. In this approach wind and conventional generators, as well as static and flexible loads, are controlled on the transmission level. Agents propose power schedules, which are aggregated by a central market operator, which then updates a price program using gradient descent, meaning the price of congested PTUs increase, and that of underused PTUs decrease. In doing so the authors show that they are able to balance power production and consumption, while satisfying power grid constraints. However, this approach focuses on the transmission level, whereas we consider the distribution-level power grid to be at a much more imminent risk of congestion. At the transmission level, the scale is much larger than at the distribution-level, both geographically and considering the power levels involved. Moreover, the power grid has a very different topology at the transmission level. A major drawback of this approach when applied at the distribution-level, is that using a global power price for congestion mitigation is not only unfair for agents in non-congested areas, but it would also be converging to a solution much too slowly. A similar approach to the negotiated dispatch is provided by [12] in which a gradient descent on the pricing is used, followed by a

local optimization of agents.

Another approach is given by [79], in which power programs are proposed by device agents that are able to satisfy cluster constraints to reach a target energy consumption, while also staying within the limits of the grid. In their approach, called *Profile Steering*, agents are only motivated to accomplish the cluster goal, which is to consume or produce energy at a specific target amount, and will propose alternative power programs whenever constraints are violated. Proposals that reduce the constraint violations the most, are then selected as the new candidate programs. In our view, the problem of this approach lies in the lack of motivation for the participant to sacrifice private rewards for running an alternative program. The only objective is the cluster goal, which means that the price of having limited resources is paid by a select set of individuals that offer the most flexibility, or conversely, the profit is reaped by those who are able to maximally make use of remaining capacity.

The authors of [44] propose an approach based on DCOPs (Distributed Constraint Optimization Problem) to solve the economic dispatch problem, as well as the real-time demand-response balancing problem. Their algorithm is able to find optimal solutions for a system of controllable generators and (predicted) loads, taking into account transmission network constraints. They show that finding the optimal solution using Dynamic DCOPs is possible, but their solution does not scale very well. A relaxed version of the problem, in which soft constraints may be (temporarily) violated, scales better, but still for relatively small time horizons, even considering their implemented solution on a GPU.

We note that related work on the topic of this chapter listed here is not exhaustive. For further detailed background, a comprehensive overview of different mechanisms for solving optimization problems in smart grids, particularly where demand response is involved, we refer to [69, 114, 139].

6.3. Self-Organizing Economic Dispatch

In order to solve the problem defined in (6.1), we propose a heuristic self-organizing method for coordinating the power scheduling of smart grid devices on the distribution-level energy grid. This multi-agent based approach allows for great scalability, while ensuring privacy and final control of the end-user.

At the distribution grid level, devices are typically consumer devices, such as PV-panels, household batteries, heat pumps, ventilation or air-conditioning units. The controllability, or flexibility of such devices is often limited, and bound by the device limitations and the user preferences. With increasing numbers of such devices, using a strictly “top-down” approach is intractable, since the search space grows exponentially with each added device. For this reason we use a

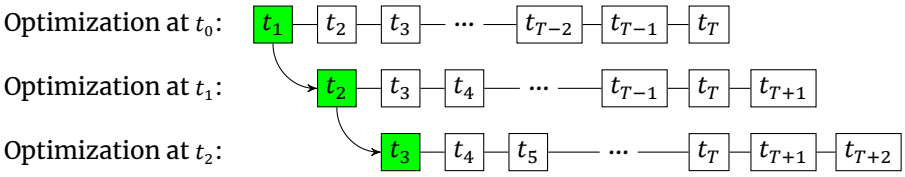

Figure 6.1: The principle of a receding horizon market is that every time step the time horizon for optimization shifts by one. Planned power programs (white) are turned into fixed contracts for the current time step (green), which determines the outcome of the algorithm. This figure is redrawn from [142].

hierarchical approach, in which the market operator delegates its task to intermediate "congestion" agents which then independently solve subproblems—which is to make sure that the total power throughput at their allocated point does not exceed a certain threshold. Making such subdivisions is justified, since in the power grid, transformers are effectively branches of the topology, and two nodes under different transformers are independent of one another; hence, transformers are the logical point to place congestion agents in the control hierarchy.

Our approach, denoted as Local Pricing Receding Horizon (LP-RH) is based on the economic incentive that end-user devices should have, to provide its owner with a service, in the most affordable way. Put differently, the system will use price differences to stimulate agents to schedule their power consumption or production in a balanced way, such that any grid constraints are satisfied. The overall scheme is simply to gather expected power programs from the connected devices, and then iteratively adjust energy prices to steer the agents into a certain power program. The method is explained in more detail in the following section, and is similar to the Negotiated Price Dispatch proposed by [142].

6.3.1. Local Pricing Receding Horizon

In order to create a power planning taking into account forecasts and/or predicted power programs of clients, we use a Receding Horizon (RH) approach, depicted in Figure 6.1 which means that at any point we only take a fixed horizon of T program time units (PTU) into account. A PTU is typically 15 minutes or one hour, throughout this paper we will use PTU length of one hour so $T = 24$. However, none of the mentioned approaches are limited to this convention. When the algorithm has converged and a power program is found for the next T PTUs that satisfies all constraints, the first PTU becomes the "current" situation, and the projected power program becomes a contract. The time horizon now shifts by one, and the entire system starts again.

The Local Pricing Receding Horizon algorithm is shown as pseudocode in Algorithm 5; an example graph on which the algorithm could run is shown in Figure 6.2. The algorithm describes how any agent finds a new

Algorithm 5 LP-RH Algorithm

Require: i {Run for this agent i}
Require: $\boldsymbol{\rho}$ {For a given price $\boldsymbol{\rho}$}
1: **if** i is a PV, load or storage agent **then**
2: $\boldsymbol{x}_i = \arg\min_{\boldsymbol{x}} C_i(\boldsymbol{x}, \boldsymbol{\rho})$ {Run local optimization}
3: **else**
4: **repeat**
5: **for all** $j \in \mathcal{M}_i$ **do**
6: $\boldsymbol{x}_j = LP\text{-}RH(j, \boldsymbol{\rho})$ {Recurse for child agents}
7: **end for**
8: $\boldsymbol{x}_i = \sum_{j \in \mathcal{M}_i} \boldsymbol{x}_j$
9: $\boldsymbol{\epsilon} = C_i(\boldsymbol{x}_i)$
10: $\boldsymbol{\rho} = adjustPrices(\boldsymbol{\epsilon}, \boldsymbol{\rho})$ {Gradient descent, see end of Section 6.3.1}
11: **until** $\boldsymbol{\epsilon} \leq \epsilon^{\max}$
12: **end if**
13: **return** $\boldsymbol{x}_i$

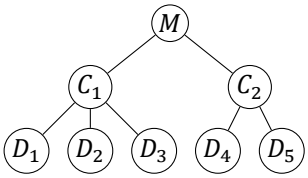

Figure 6.2: Simple example topology of a graph which could be running the LP-RH algorithm. One market operator (M) is connected to two congestion agents (C_i), which are connected to a total of five device agents (D_i). The device agents can represent PV panels, consumer loads, storage agents, or any other leaf nodes in the power grid. Topologies where congestion agents are connected to more congestion agents are also possible.

power program $\boldsymbol{x}_i$ to satisfy its local constraint C_i. Every device agent in the grid does this by solving a local optimization problem (line 2), taking into account local constraints as explained in Section 6.3.2.

Prices $\boldsymbol{\rho}$ are updated by the market operator agent and the congestion agents in order to get a power program $\boldsymbol{x}$ satisfying the grid constraints. When the market operator or any congestion agent runs the LP-RH algorithm, all $j \in \mathcal{M}_i$, where $\mathcal{M}_i$ are the immediate children of agent i, are requested to propose a power program based on the price $\boldsymbol{\rho}$ (line 6). Here the function is called recursively until all agents have determined a new power program. Note, that if any of the children are congestion agents, they determine their power program by forwarding the prices to *their* children and returning the sum of all received programs. Many agents will not know exactly how they will function in the future, as contexts might change, a user might behave differently than anticipated, or weather conditions may act up; this is why receding horizon method updates the time iteratively. With

Table 6.2: A summary of the three functions specifying the behaviors of the agents.

Agent type	Cost	Constraint	Power
Market Operator	$J^{\text{MO}} = 0$	$\mathcal{C}^{\text{MO}} = \bar{\boldsymbol{x}} - \boldsymbol{\Theta}$	$\bar{\boldsymbol{x}} = \sum_{i \in \mathcal{N}} \boldsymbol{x}_i$
Congestion agent	$J_i^{\text{CO}} = 0$	$\mathcal{C}^{\text{CO}} = \bar{\boldsymbol{x}}_i - b_i$ iff $\lvert\bar{\boldsymbol{x}}_i\rvert < b_i$	$\bar{\boldsymbol{x}}_i = \sum_{j \in \mathcal{M}_i} \boldsymbol{x}_j$
Load agent	$J_i^{\text{LOAD}} = 0$	$\mathcal{C}_i^{\text{LOAD}} = 0$	$\boldsymbol{x}_i$
PV agent	$J_i^{\text{PV}} = \boldsymbol{x}_i - \boldsymbol{x}_i'$	$\mathcal{C}_i^{\text{PV}} = 0$ iff $\tau \boldsymbol{x}_i \boldsymbol{\rho} \geq \gamma_i$	$\boldsymbol{x}_i = \boldsymbol{x}_i'$ or 0
Storage agent	$J_i^{\text{ST}} = \boldsymbol{x}_i(1 - \eta_i')$	$\mathcal{C}_i^{\text{ST}} = 0$ iff $e_i^{\min} < \boldsymbol{e}_i < e_i^{\max}$	$\boldsymbol{x}_i = f(\boldsymbol{\rho})$

the sum of all power programs, the market operator can determine the error $\boldsymbol{\epsilon}$, which is the sum of all local constraint violations $\mathcal{C}$ for every $t \in T$.

In line 9 the constraint $\mathcal{C}_i$ is used to compute the local constraint violation and stored as an error variable $\boldsymbol{\epsilon}$. The value of $\boldsymbol{\epsilon}$ is taken into account to adjust the prices in line 10. In the function $adjustPrices(\boldsymbol{\epsilon}, \boldsymbol{\rho})$, a gradient descent approach is used, which linearly interpolates the last two errors as a function of the price. We then choose the price at which the error is projected to reach zero. Note that this means we assume the reaction of the devices linearly depends on the prices, which will only hold under very specific circumstances. To overcome this issue, we iteratively repeat the process until $\boldsymbol{\epsilon} \leq \epsilon^{\max}$ and assume that a non-linear response can be described as a series of linear pieces. The parameter $\epsilon^{\max}$ then represents an upper bound on the error, which we can use as a convergence criterion.

6.3.2. Agent Behavior

Agents in the system are characterized by the device that they have to assign a power program for. In the distributed case, every agent locally optimizes a local cost function J_i, which is part of the global optimization problem (6.1). Also, local constraints $\mathcal{C}_i$ have to be taken into account which are represented by constraint (6.1e). The behavior of the agents can be defined by three functions for the costs, the local constraint and the power program; an overview of this is shown in Table 6.2. In our model we consider the following types of agents.

Market Operator

This is the root node of the tree (M in Figure 6.2), as far as the local distribution grid concerns. In the physical grid, it corresponds to the transformer that connects the local low voltage (LV) grid to the medium voltage (MV) grid. We assume that there is no energy loss at the market operator, so

$$J^{\text{MO}} = 0. \tag{6.2}$$

Its goal is to find a solution to (6.1), and its private cost would be the same as the target constraint in (6.1b). The market operator runs Algorithm 5, using a deviation of the target profile $\boldsymbol{\Theta}$ as an error, which is minimized by

the algorithm. Hence, its local constraint is defined as

$$\mathcal{C}^{\text{MO}} = \bar{\boldsymbol{x}} - \boldsymbol{\Theta}, \tag{6.3}$$

where $\bar{\boldsymbol{x}}$ denotes the power program of the market operator. Since the market operator is not a device in the grid itself, its power is the sum of the powers of its children, which in case of the market operator are all nodes in the cluster

$$\bar{\boldsymbol{x}} = \sum_{i \in \mathcal{N}} \boldsymbol{x}_i. \tag{6.4}$$

The value of (6.3) can either be negative or positive, which respectively means either the total power production or the total consumption is too high. The market operator sets an initial price of $\boldsymbol{\rho} = 0.5$ for all PTUs, and then uses Algorithm 5 to minimize the constraint value until it reaches zero, in order to satisfy the global constraint (6.1b).

Congestion Agent

This is an intermediate node on the grid tree (C in Figure 6.2) connected to a parent node who is either the market operator or another congestion agent. It corresponds to a component in the grid where congestion might potentially occur, such as a transformer. Equivalently to the market operator, we assume that no energy is lost here, so

$$J_i^{\text{CO}} = 0. \tag{6.5}$$

This agent has a constraint that aims to limit the power usage of that part of the grid, this corresponds to the constraint (6.1c). The congestion agent also uses Algorithm 5 to minimize the error of its local constraint, which is defined as

$$\mathcal{C}_i^{\text{CO}} = \begin{cases} \bar{\boldsymbol{x}}_i - b_i, & \text{if } |\bar{\boldsymbol{x}}_i| \geq b_i, \\ 0, & \text{otherwise.} \end{cases} \tag{6.6}$$

Here b_i is the congestion threshold of the agent i, which means it is the maximum power throughput of the grid at the point that i represents. Again $\bar{\boldsymbol{x}}_i$ denotes the power program of the congestion agent, but is now equal to the sum of all devices under the current node

$$\bar{\boldsymbol{x}}_i = \sum_{j \in \mathcal{M}_i} \boldsymbol{x}_j, \tag{6.7}$$

Similar to the constraint of the market operator in (6.3) the constraint value can become negative or positive, and is used to compute the error $\boldsymbol{\epsilon}$ in Algorithm 5.

Load Agent

This is an agent responsible for an uncontrollable load in the grid (represented by any device agent D in Figure 6.2). This could be a consumer household, an office, street lighting, or any other non-flexible load, and thus only participates in the problem as part of constraint (6.1b). In the distributed system however, it is responsible for making a forecast of the power usage, and this forecast will be updated with more accurate information as the time horizon shifts. The load agent has no attached cost in the global problem, and no need to locally compute any optimal behavior. This is equivalent to stating that its cost and constraint correspond to

$$J_i^{\text{LOAD}} = 0, \tag{6.8}$$

$$\mathcal{C}_i^{\text{LOAD}} = 0. \tag{6.9}$$

Its corresponding load profile $\boldsymbol{x}_i$ is fixed to some profile that constrains the global problem (6.1). In our experiments its values are taken from real households as described in Section 6.4.2.

PV Agent

For a PV agent $\boldsymbol{x}_i$ denotes the amount power produced in the time horizon. The PV agent (any D in Figure 6.2) has some flexibility to offer to the optimization function (6.1) by allowing curtailment in reference to the expected generation. We assume that curtailment is binary, in that either the PV generates power as normal, or it is switched off and produces no power at all. Curtailing means that there is potential energy lost, and hence the cost function of a PV agent is defined as

$$J_i^{\text{PV}} = \boldsymbol{x}_i - \boldsymbol{x}_i', \tag{6.10}$$

where $\boldsymbol{x}_i'$ indicates the *expected* power, when not curtailing. This expected power is taken from the scenario, which will be detailed in the Section 6.4.

When reacting to prices in the distributed system, a decision is made in order to decide whether to curtail based on the price profile. If the operational running costs γ_i of the PV is *more* than the power that would be generated by it, there is no point in running the generator (from an economic point of view). Hence, the local constraint and its corresponding decision rule of a PV agent can be written as

$$\mathcal{C}_i^{\text{PV}} = \begin{cases} 1, & \text{if } \tau \boldsymbol{x}_i \boldsymbol{\rho} < \gamma_i, \\ 0, & \text{otherwise}, \end{cases} \tag{6.11}$$

$$\boldsymbol{x}_i = \begin{cases} 0, & \text{if } \tau \boldsymbol{x}_i' \boldsymbol{\rho} < \gamma_i, \\ \boldsymbol{x}_i', & \text{otherwise}. \end{cases} \tag{6.12}$$

For the PV agent it also holds that the corresponding expected power program $\boldsymbol{x}_i'$ determines the global problem (6.1). Its values in our experiments are taken from real PV panels as described in Section 6.4.2.

Storage Agent

In the context of storage, $\boldsymbol{x}_i$ denotes the planned charge and discharge actions in the time horizon. A storage agent (again a leaf node D in Figure 6.2) provides flexibility by allowing to store some energy in a local storage like a battery or a heat buffer. There are limits to the amount of energy that can be stored, either because of the physical limitations of the storage device, or because of the end-user settings. Moreover, a storage agent has some efficiency, which defines energy loss when energy is put into it, or out from it. This means that we can define the cost function of the storage agent as the energy lost during charging or discharging

$$J_i^{\mathrm{ST}} = \boldsymbol{x}_i(1 - \eta_i'), \tag{6.13}$$

where

$$\eta_i' = \begin{cases} \eta_i, & \text{if } \boldsymbol{x}_i \geq 0, \\ \eta_i^{-1}, & \text{otherwise.} \end{cases} \tag{6.14}$$

This difference makes sure that the loss is correlated to the internal power of the battery when charging or discharging. Put differently, $\boldsymbol{x}_i$ defines the power at the grid side of the storage, and when an agent is charging, a lower power effectively charges the battery, and conversely when discharging a higher power is required to provide some power level to the grid.

In order to define the constraints of the storage agent we must define the update function for the amount of energy $\boldsymbol{e}_i$ stored as the cumulative sum of the powers

$$e_{it} = e_{i0} + \tau \sum_{t'=t_0}^{t'=t} \eta_i' \boldsymbol{x}_{it'} - \lambda_i. \tag{6.15}$$

Here λ_i represents the leakage or self-discharge rate of the storage agent i, and e_{i0} is the stored energy at the start of the experiment. Let us now define the following constraints on the storage agent

$$\mathcal{C}_i^{\mathrm{ST}} = \begin{cases} 0, & \text{if } e_i^{\min} < \boldsymbol{e}_i < e_i^{\max}, \\ 1, & \text{otherwise.} \end{cases} \tag{6.16}$$

The minimum and maximum energy values are defined by $e_i^{\min}$ and $e_i^{\max}$, respectively.

For a storage device, the power limits denote the maximum charge and discharge rates. They are defined by $x_i^{\max}$ and $x_i^{\min}$, respectively, in (6.1d). A special case of a storage device is a heat pump, which (electrically powered) heats a house in order to keep the temperature within comfortable levels. The heat pump only allows charging and only discharges through leakage, hence for a heat pump $x_i^{\min} = 0$ and $\lambda_i > 0$.

When a storage agent has to update its expected power program in line 2 of Algorithm 5, some response function ($f(\boldsymbol{\rho})$ in Table 6.2) is required, which is "economically sane" and has the following characteristics:

- return $x_i^{\max}$ for low prices and $x_i^{\min}$ for high prices,
- be a monotonically decreasing for increasing prices,
- have a "plateau" of zero power response for some intermediate price ($\boldsymbol{\rho} = 0.5$), which is wider for less efficient devices.

The final characteristic allows more efficient devices to respond to subtle price change, and have less efficient devices respond to more extreme prices. This way devices with a higher efficiency are used first when flexibility is needed, and (when parameterized correctly) less efficient devices will only be used when required. In our implementation we chose a relatively simple response, defined by four points in the price domain:

1. $\rho_1 = \rho_2 - \zeta$, the highest price at which the device will use the maximum charge rate ($\boldsymbol{x}_i = x_i^{\max}$),
2. $\rho_2 = \eta_i/2$, the lowest price at which the device will not charge or discharge ($\boldsymbol{x}_i = 0$),
3. $\rho_3 = 1 - \rho_2$, the highest price at which the device will not charge or discharge ($\boldsymbol{x}_i = 0$),
4. $\rho_4 = \rho_3 + \zeta$, the lowest price at which the device will use the maximum discharge rate ($\boldsymbol{x}_i = x_i^{\min}$),

where ζ is an arbitrary constant that defines the interval in which the agent linearly decreases its power response between the price points. By choosing the points in this way, we follow the requirements as specified before, and we can ensure that two types of devices, that have an efficiency η_i difference of more than ζ will be put to use separately. This power/price relation we chose for our numerical evaluation with $\zeta = 0.075$ is depicted in Figure 6.3.

An alternative strategy for the storage agent would be to make use out of any differences in the price, charging and discharging as soon as the price differences are large enough to overcome its efficiency. From a strictly economic perspective, this is the optimal strategy to maximize its own benefit. However, this leads to very "binary" behavior with minimal and maximal charging rates [95] and thus, little room for optimizing from the market operator and congestion agent. Therefore, a linear strategy is implemented as depicted in Figure 6.3, allowing to solve the overall optimization problem (6.1).

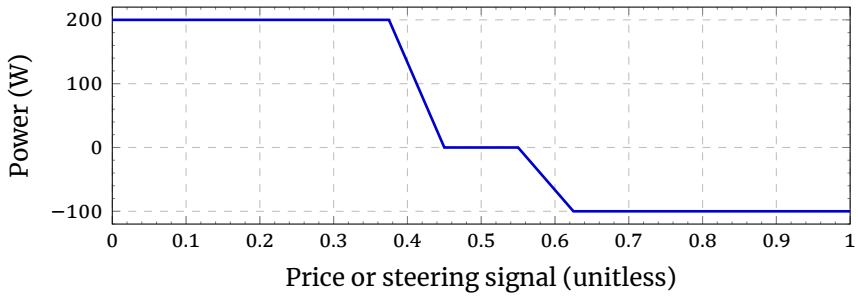

Figure 6.3: The strategy of the storage agent with $x_i^{\max} = 200\,\text{W}$, $x_i^{\min} = -100\,\text{W}$ and $\eta_i = 0.9$ shows the power response for an increasing price. The *plateau* at 0 W extends from $\rho = \eta_i/2$ to $\rho = 1 - \eta_i/2$.

6.4. Experiment Setup

The LP-RH algorithm was empirically evaluated by running simulations of an LV grid with a set of realistic household load profiles and PV production profiles for a series of 24 PTUs. Simulations were randomized by selecting different load and production profiles, and random households were selected to have a heatpump or a household battery. The simulations are implemented in Matlab, and require around 30 seconds to simulate a single day. More information about the different implementation details are specified below.

6.4.1. Distribution Network Topology

For the topology of the network we use the European Low Voltage Test Feeder [66] network. This dataset is used to benchmark power and energy algorithms on realistic European distribution networks. In this chapter we superimposed six points on the topology, where we monitor and mitigate any potential congestion. These points are strategically chosen to separate the problem into independent subproblems. The resulting topology with the congestion points are shown in Figure 6.4.

6.4.2. Household Load and PV Profiles

The household consumption and production profiles are taken from a pilot study [121], in which 92 residential consumers were monitored over the course of a year (from March through November 2018). The data was preprocessed such that we have separated information on the consumption of houses, and of the PV installations. Data is anonymized and randomized per month, so that we can select data from any specific month for a base load of a household, or a residential PV installation.

In the experiment the 54 households from Figure 6.4 were assigned a random instance of the load *and* PV profiles from the same month (i.e. all households were equipped with PV panels). The daily load consumption

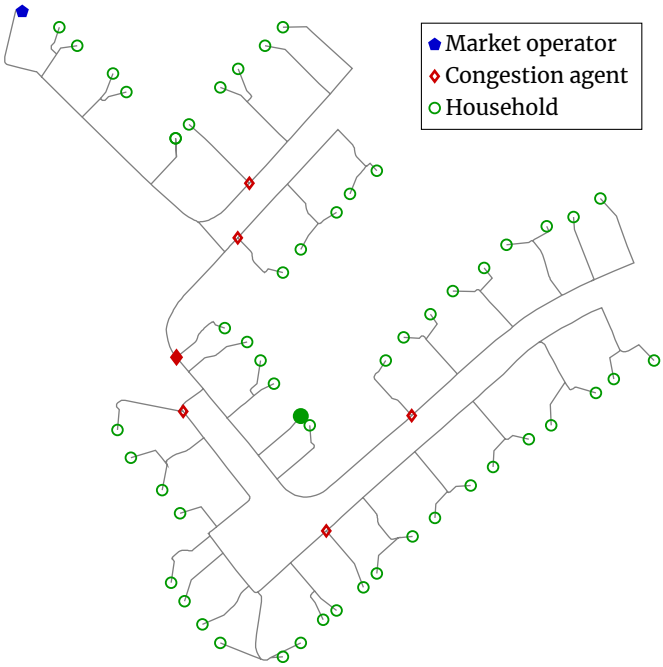

Figure 6.4: The topology of the IEEE LV feeder network used for the simulation. The figure depicts the connections between household consumers, the intermediate congestion agents, and the root market operator. The PV, load and storage agents are randomly placed at the households for different simulation runs. The "filled" markers represent the agents displayed in Figures 6.5–6.7.

varied between 4.98 kWh and 29.39 kWh, and the total PV production varied between 826 Wh and 18.8 kWh. Furthermore, 16 randomly selected households were assigned a household battery, and again 16 were chosen to have a heat pump installation.

Every household and PV installation in the simulation would select a random profile from the dataset, which was used as its power program. The objective $\boldsymbol{\Theta}$ was set to a total net consumption of the LV grid using the mean total power consumption of the houses including PV production. Choosing the target profile $\boldsymbol{\Theta}$ in this way corresponds to a situation in which the energy provider of the simulated neighborhood would agree to a contract for the average behavior of the households, and subsequently attempts to use flexibility to account for any deviations from the normal.

The batteries were all dimensioned with storage capacities of $e^{\max} = 10.8\,\text{kWh}$, and maximum charge and discharge rates of $x^{\max} = 4000\,\text{W}$ and $x^{\min} = -4000\,\text{W}$, respectively. The batteries charging efficiencies were all set at $\eta = 0.9$. The heat pumps were estimated to have a working energy capacity of $e^{\max} = 2\,\text{kWh}$, this means the difference between the thermal energy capacity of the house at the minimum and maximum

comfortable user temperature is 2ψ kWh, where $\psi = 4.0$ is the coefficient of performance of the heat pump. Then, the heat pumps have a $x^{\max} = 1600\,\mathrm{W}$ and $x^{\min} = 0\,\mathrm{W}$, an efficiency of $\eta = 1$ and a constant leakage rate of $\lambda = 360\,\mathrm{W}$. Finally, to ensure convergence, the maximal error value is set to $\epsilon^{\max} = 10^{-3}$ in the experiments for this chapter

6.4.3. Forecast Uncertainty

The predicted load and production power programs $\hat{\boldsymbol{x}}$ of the load and PV agents were generated by taking average profiles of the complete dataset. These average profiles are considered as taken from historic data and hence, provide a ground for predicting the power program of future PTUs. When agent i determines its power prediction $\boldsymbol{x}'_i$, it will compute a weighted average between its assigned power program $\boldsymbol{x}_i$ and the average profile $\hat{\boldsymbol{x}}$, such that

$$\boldsymbol{x}_{it} = (1-\alpha)\boldsymbol{x}'_{it} + \alpha\hat{\boldsymbol{x}}_t, \tag{6.17}$$

$$\alpha = \sqrt{\frac{t-1}{T-1}}, \tag{6.18}$$

such that at $t = 1$ the prediction equals the selected profile $\boldsymbol{x}'_{it}$. At $t = T$, the prediction is simply the average power $\hat{\boldsymbol{x}}_t$.

6.5. Centralized Solver

In order to address the performance of the LP-RH algorithm, a centralized optimization approach is also introduced to provide lower bounds to its results. A mixed integer linear program (MILP) solver was used to find these bounds for the problem stated in (6.1). The centralized optimization approach considers the exact same scenario that was solved by the decentralized algorithm; the energy loss is minimized and the same set of constraints apply. A fundamental difference lies in the availability of information. Whereas detailed information is only shared locally in the LP-RH algorithm, the centralized optimization approach assumes complete knowledge of the current system state for the decision-maker; i.e., no limits are imposed on the spacial flow of information within the network. With these features in mind, two versions of the MILP were formulated: Receding Horizon Centralized Solver and Perfect Information Centralized Solver.

6.5.1. Receding Horizon Centralized Solver

The receding horizon centralized solver (RHCS) is the most similar to the LP-RH algorithm. It uses a receding horizon approach (as depicted in Figure 6.1) to find an ideal dispatch solving the consecutive sub-problems. Uncertainty about future device states is again simulated by (6.17).

Because this solver has perfect information at the *current* state for each iteration of the receding horizon, its solution represents a lower bound on the solution found by the LP-RH algorithm.

6.5.2. Perfect Information Centralized Solver

In the perfect information centralized solver (PICS), a single centralized optimization problem is solved for the complete dispatch. In addition to perfect spacial information, this version of the MILP also has perfect *temporal* information about the complete system state; this means that no uncertainty about future states is simulated. Referring back to (6.17), this is equivalent to setting $\alpha = 1$, which results in perfect predictions for each profile. The solution of this MILP represents an absolute lower bound for problem (6.1).

6.6. Results

In this section the results of the described experiments are shown. Before comparing the performance of the different solvers, which is done in Section 6.6.1, the behavior of the LP-RH algorithm is shown in this section. The graphs in this section show examples of single simulation runs—subject to randomized starting conditions—demonstrating the behavior of the different agent types. Powers are shown as power consumption, this means a net consumption is shown as positive power, and conversely negative powers indicates a net power production. In Figure 6.5 the final result of the market operator is shown, where the market operator has found a price profile such that the target profile is met exactly.

Figure 6.6 shows the power program of one of the three congestion agents that is directly connected to the market operator. Its power congestion limits are set such that in the peak moments of the day there is some congestion expected. This results in a price difference shown in the power program, as around the peak PV production ($t = 12, 13, 14$) the local price is slightly lower than the market price, leading to a lower net power production. Similarly, we can identify that at the end of the day ($t = 19, 20, 21, 22$) a positive power congestion was mitigated by increasing the price.

The power program of a heat pump agent is shown in Figure 6.7. This heat pump is connected directly to the congestion agent, the results of which are shown in Figure 6.6, hence its price profile should be identical. What is most obvious in this graph is that the high price at $t = 12$ leads to a zero power consumption, and quickly after that, the power consumption rises in order to maintain comfortable temperatures. Again, around $t = 19, 20, 21$ the relatively high prices lead to a power consumption of zero, which was apparently feasible because of the high power consumption leading up to it.

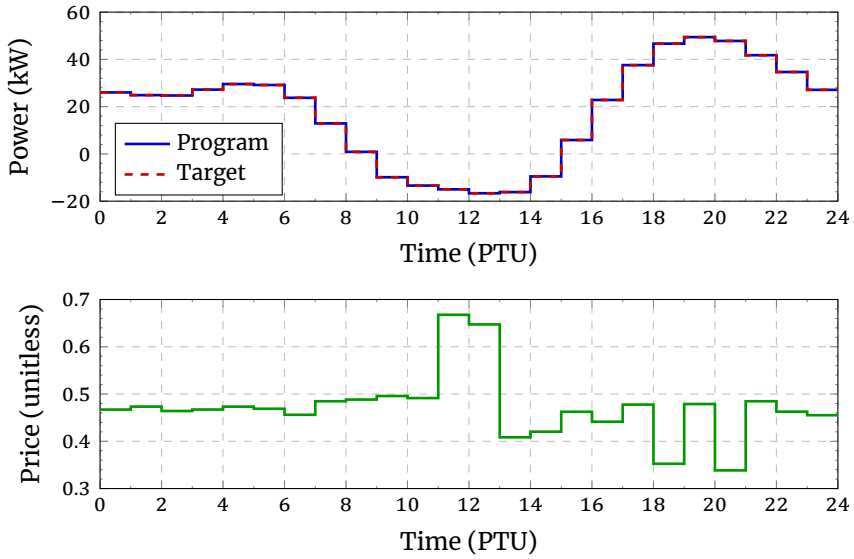

Figure 6.5: The power program of the market operator and the price profile as the outcome of Algorithm 5, show the results of a 24-hour simulation of the problem with the LV feeder network and 92 random households.

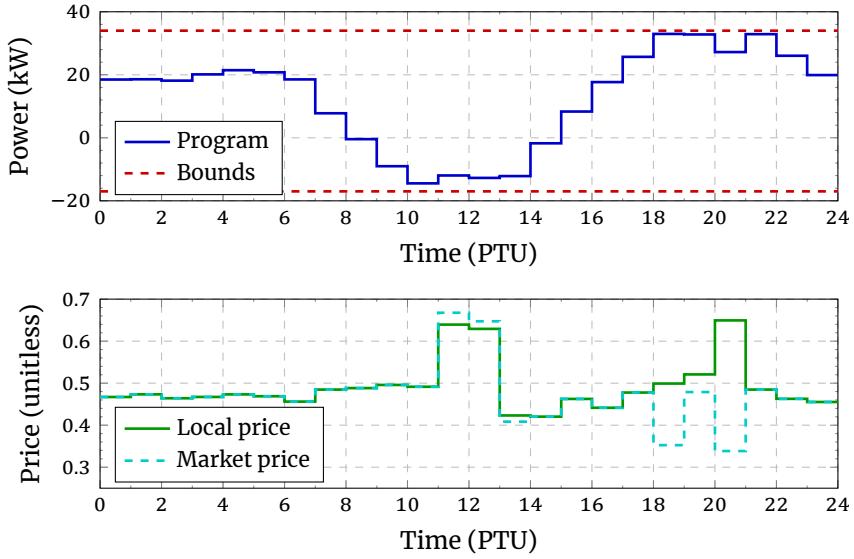

Figure 6.6: The power program of a congestion agent shows some periods of congestion at the production and consumption bounds (red dashed lines), and the corresponding changes in price profiles (from Algorithm 5) relative to the global price of the market operator.

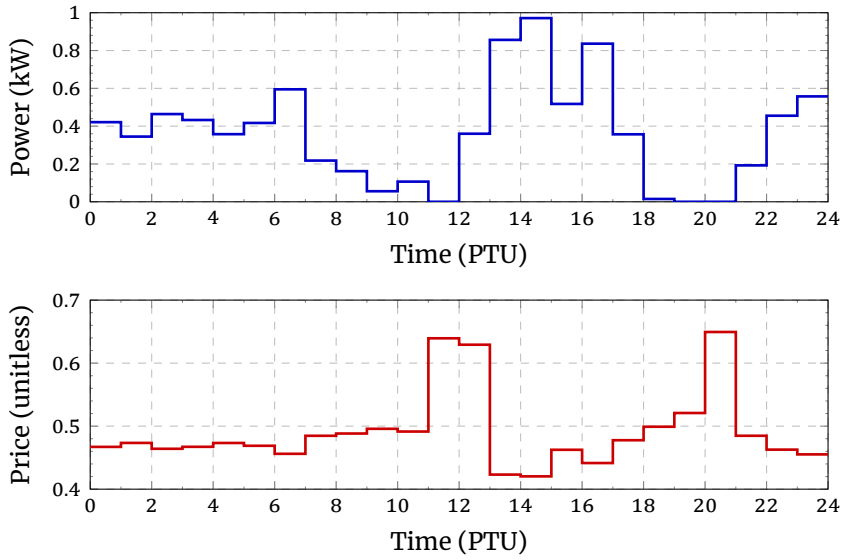

Figure 6.7: The power program of a heat pump agent shows the power being mostly used at moments where the price is low, relieving the need to charge when the price is high; or considering the view of the grid, when a lower power consumption is required.

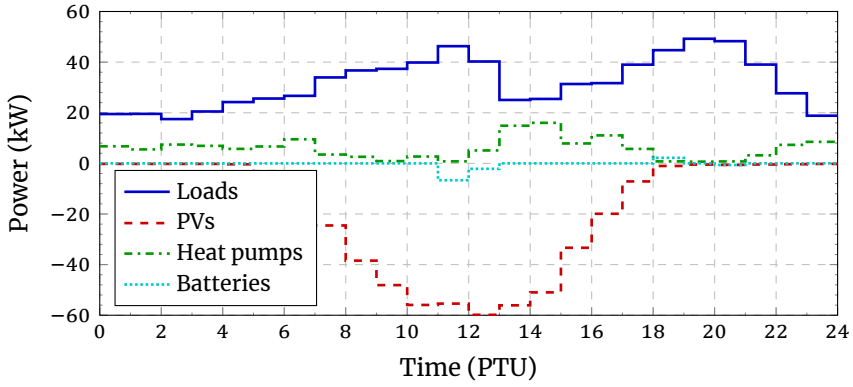

Figure 6.8: The total power consumption per device type show that the batteries are used far less than the heat pumps, since they have a lower efficiency. The PV panels did not have to be curtailed in this run.

Finally, Figure 6.8 shows the total power programs of all devices in this experiment summed up. In this figure, the power programs of the loads and the PVs are the direct result of the chosen profiles, and are the input for problem (6.1). We can see that for the majority of the experiment, all used flexibility is from the heat pumps with the high efficiency. Only at times of the congestion will the less efficient battery agents be used.

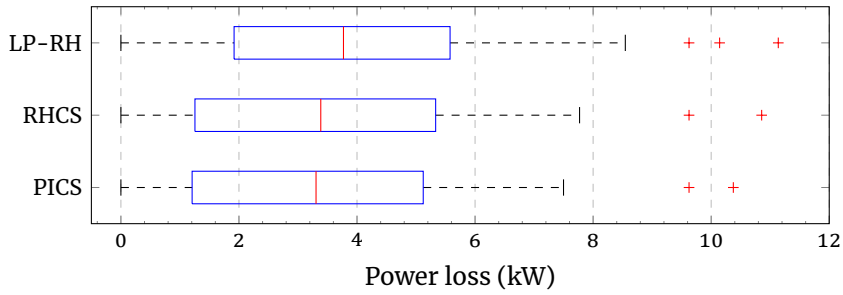

Figure 6.9: Compared with centralized optimization solvers, the LP-RH algorithm performs equally well. In this boxplot the median is shown as a line, the boxes indicate the 25% and 75% percentiles, and the tails are capped at a maximum length of the box width; outliers are drawn separately.

6.6.1. Comparison with Central Solvers

In a validation experiment 105 random problem instances (15 permutations for seven months) were created, and solved by the three different algorithms. In Figure 6.9 the results are shown for the *feasible* instances. A solution is considered feasible if all three solvers were able to find a solution that satisfies all constraints. The LP-RH algorithm did not find a correct solution for 26 problem instances, 11 of which were found to be overconstrained according to RHCS. For the feasible instances the LP-RH algorithm found solutions that were not significantly worse than the PICS, and in 21 instances found a solution with the exact same cost. In 27 instances LP-RH found a solution that was equally well as the RHCS or even slightly better—this seems to contradict the initial statement that RHCS acts as a lower bound for LP-RH; however, this is due to the way both solvers deal with uncertainty. The LP-RH algorithm allows the congestion agents to violate power constraints for future PTUs ($t > 1$), but not for the next PTU ($t = 1$), by allowing an additional error margin for future PTUs. The rationale is that we do not want to focus on potentially imperfect forecasts according to (6.17), as long as we ensure that eventually PTUs are not congested. The RHCS algorithm does not allow power constraints to be violated at any point in the future, and hence might react too strict when a future congestion is predicted, but does not actually occur.

In the set of *infeasible* problems, i.e. problems that do not have a valid solution satisfying all constraints, LP-RH does find solutions that minimize the power loss. However, since these solutions temporarily overload congestion agents (where $x_i > b_i$), or does not match the target profile $\mathbf{\Theta}$ exactly, they are not a fair comparison, since they do not strictly solve (6.1). In a separate run the problems were relaxed, by increasing the congestion thresholds of the congestion agents. This resulted in the LP-RH not being able to find a solution in only 14 problem instances, but the results were otherwise similar to the ones reported here in Figure 6.9.

6.7. Conclusions

Implementing a self-organizing multi-agent system for smart grids requires new dispatch algorithms, that do not rely on centralized control. In Chapter 2, we showed how a framework for self-organization can be used by a system to change its deployment, but still had no way to reason about the configuration itself. The LP-RH algorithm for power dispatch is just that: an algorithm for distributed decision-making. Note, that the algorithm is specifically written to optimize the problem at hand, although the underlying principles can apply to other problems.

We have introduced an algorithm for self-organization, by solving the economic dispatch problem using a decentralized market based approach with local pricing. The LP-RH algorithm using a hierarchical approach was shown to be able to solve the problem using a fairly simple interaction scheme in which pricing information is sent down the hierarchy tree, and planned or forecasted power programs are sent back up. Using a gradient descent approach, the market operator is capable of tuning the pricing to find feasible solutions to minimize the power losses in the grid.

In our experiments we found that in 20% of the problems LP-RH did not perform any worse than a perfect-information centralized solver. In the other 80% our algorithm did not perform significantly worse. The benefit of LP-RH over a centralized solver are in the robustness and scalability of the solution, as well as in the preserved privacy of the end-consumers.

In the implementation of the response of the storage agent, we intentionally did not choose to respond with an optimal power program given the price signal. Particularly, when a high price is expected in the future, the agent will not "proactively" charge to avoid having to charge later, or vice versa. This behavior could be implemented at the agent quite easily using a dynamic programming approach, but it would lead to very extreme behavior, e.g. very binary behavior of charging or not-charging at full capacity even for small price differences. This binary behavior is hard to deal with in the rest of the hierarchical tree, and does not lead to any problems per se, but might be improved upon in a future continuation of this work.

Other variations of the problem may include other device types, for instance time-shiftable devices such as washing machines or dishwashers. Also, using electric vehicles (EV) as an additional type of agent, providing energy flexibility is a very interesting extension, which will undoubtedly lead to other complications because of their high power ratings (i.e. they dominate the total power consumption when charging). Finally, an integration with a real time balancing algorithm such as [80] would be very fruitful to complete the needs of the future smart grid.

In this chapter a very specialized method for driving the self-organizing behavior was provided, compared to a generic reusable DCOP-based approach in the previous chapters. The same underlying principle however is that through local interactions and decisions made on partial

knowledge, a solution is sought to a global problem. In the next and final chapter we discuss the main outcomes of this thesis and look ahead to potential ways to extend this work.

7

Conclusions and Future Work

We can only see a short distance ahead,
but we can see plenty there that needs to be done.

Alan Turing

In order to keep up with the quickly growing networks that exist today, we need to minimize human intervention and maximize self-dependency in order to keep systems operational. Self-organization is key when it comes to making a group of agents cooperates under varying conditions, achieving changing goals, while forming new teams. There exist already many methods for making systems more adaptive to changing operating conditions, but when a group of agents is involved that have to adapt to a new condition as a team, these methods do not always apply. In this thesis different topics were discussed that are required in order to attain or improve existing self-organizing behaviors to a system consisting of cooperative agents.

In Chapter 1, we asked the main question of this thesis:

> How can we achieve self-organization, or improve the self-organization capabilities in a network of cooperative agents?

The answer to this question consists of two components, which are addressed in two challenges:

Challenge 1 Create a framework that allows self-organizing systems to redefine their deployment.

Challenge 2 Find or improve strategies to coordinate collective behavior.

Firstly, in this thesis we have proposed a framework for adding self-organization capabilities to multi-agent systems, specifically by adding a layer of intelligence to an agent, which is dedicated to making decisions on the configurations of the system at hand. We have shown that this framework provides the means of to reason about self-configuration, but it does not specify *how* to self-organize. Therefore, secondly, a reasoning mechanism is required, but there are different methods that allow to reason about such decisions. Throughout this thesis we have used three methods: logic based, distributed decision-making using DCOPs, and a market-based control algorithm. These three approaches are very different in their nature, and deciding when to use which approach depends largely on the problem at hand, the type of decision variables that can be altered, and the expert knowledge of the system designer.

We will further detail the outcomes of this thesis in the following section, addressing the main challenges as defined in Chapter 1.

7.1. Contributions

In Chapter 2, we addressed the first challenge: **creating a framework that allows self-organizing systems to redefine their deployment**. A framework for self-organizing multi-agent systems was proposed. Particularly, this framework was used to add self-organization to a sensor network, in which a group of sensors cooperates to estimate the state of a greenhouse climate. Using the provided mechanisms, agents are able to reason about the current context of the sensors and about how to most efficiently perform the task they were assigned. In the use case, a set of preexisting algorithms for performing the state estimation was provided, along with some knowledge about the conditions under which any method should be used and how it should be configured. The modular implementation of the framework, and the separation of the "primary path" from the "secondary path" allows the use of different reasoning techniques. The framework and corresponding tools are further detailed in [88, 91].

The second challenge was: **finding or improve strategies to coordinate collective behavior**. This challenge was addressed in Chapters 3 to 5 by using a solver for DCOPs and in Chapter 6 using a decentralized market based approach. Specifically, in Chapter 3 a new (A)DCOP algorithm called Cooperative Constraint Approximation (CoCoA) was introduced as one such candidate for reasoning about the primary task of an agent. The CoCoA algorithm is in its essence a simple mechanism to coordinate variable assignments in such a way that a local (1-hop) optimal assignment is being made. CoCoA is unique in its kind by not using an iterative approach to finding solutions, but instead finds a solution by spreading its activity through the problem graph, assigning values as it does so. Comparing this strategy to state-of-art DCOP algorithms showed that it not only

reduced the communication overhead, but also finds solutions that are nearly equivalent in costs.

The original CoCoA algorithm was extended as CoCoA_CA in Chapter 4 in order to be applied to wireless power transfer networks. In this problem it is strictly not allowed to violate a hard constraint, stating that the total electromagnetic radiation can not exceed safe levels. By adding an extra check in CoCoA_CA to make sure neighboring agents do not assign a value concurrently, in which case they would not take the other assignment in consideration, these hard constraints are satisfied in all cases. We furthermore showed that even though CoCoA_CA does not perform as well as a centralized LP, the distributed mechanism is better capable of adapting the behavior of agents, when perfect information is not available.

Another shortcoming of CoCoA was addressed in Chapter 5: the fact that CoCoA only assigns a value once, makes it impossible to recover from early choices. We have shown that using CoCoA as an initialization strategy for other (iterative) DCOP solvers, we do not only improve on the convergence rate of these other solvers and on the solution cost of CoCoA, but also on the solution cost of the other solvers. This effect turns out to stem from the fact that in many graphs there are *bridge* edges (or pseudo-bridge edges), which connect clusters of connected vertices. When the nodes on such bridge edges are assigned values that violate the constraint on that edge, it is very hard for an iterative solver to overcome that initial assignment. CoCoA, and to a lesser extent zero-step-lookahead solvers, are unlikely to assign a constraining value on such edges, and thus allow for a better solution.

Although DCOPs are a way of implementing coordinated decision-making, it is not always the best approach to solving this problem—especially "fully-connected" problems, where variables share constraints with all other variables, are difficult to solve. Such problems can be described as a DCOP, but solving them as such is very inefficient. For this reason, in Chapter 6 we introduced another self-organizing mechanism, specifically for solving the Distributed Economic Dispatch problem, with flexible distributed energy resources. We have shown how this problem can be formalized and optimized using a decentralized multi-agent market-based approach, in order to minimize the energy loss, and maximize social wellfare.

7.2. Future Work

In this thesis we have presented some mechanisms to implement self-organization in a multi-agent system. The proposed techniques have shown to be able to make decisions autonomously of the functioning of the system, adapt to changes in the environment, without external intervention, and to find solutions that are near optimal. This relieves much work from human operators, but there are still many questions that

remain unanswered. Based on our challenges from Chapter 1, we list some future research possibilities that we think should be addresses next.

Challenge 1: Create a framework that allows self-organizing systems to redefine their deployment. In Chapter 2, we proposed a framework for self-organizing systems in a state estimation context. This framework was founded on the principle of separation-of-concerns, and as such, few assumptions were made on the primary task of the system. The framework itself is quite conceptual, and lightweight. It could be extended by introducing more concrete mechanisms for implementation, and even add ready built modules and services. This will limit the flexibility, but may further alleviate tasks from the system designer. The benefits of such "frameworks" would undoubtedly exist, but are notoriously difficult to quantify.

Challenge 2: Find or improve strategies to coordinate collective behavior. A key contribution of this thesis is the CoCoA algorithm, and the hybrid solvers, which were shown to efficiently solve DCOP problems. However, different mechanisms for reasoning about self-organization were used in this thesis; we did not find a "one approach fits all" mechanism that works for any problem in all domains. If such an approach does exist, it would need to be able to learn about the problem at hand, instead of using an implicit or explicit model of the primary system. Examples of such a mechanism are sometimes referred to as Distributed Constraint Evaluation and Exploitation (DCEE) algorithms [110, 136] and may involve other machine learning methods such as Reinforcement Learning [36, 94], but this was not addressed in this thesis. Alternatively, if such a "catch-all" mechanism does not exist, further research is needed to find self-organization mechanisms for different problem domains.

Finally, the problem of finding coordination strategies with a small overhead for multi-agent decision-making is one that applies in many problem areas [99]. The First Order Logic reasoner is quite lightweight, and the LP-RH market based mechanism also appears to scale well, but we have not validated these approaches in very large deployments. Also, CoCoA is an efficient solver compared to other DCOP algorithms, but there may be different graph structures or constraint types, on which this is not the case. In part, such drawbacks can be diminished using the Hybrid DCOP approach proposed in Chapter 5, but more research about the type of problems and the effects on CoCoA thereof would increase the usability of the method.

References

[1] S. Abdallah and V. Lesser, *A multiagent reinforcement learning algorithm with non-linear dynamics*, Journal of Artificial Intelligence **33**(1):521–549 (2008).

[2] M. A. Abido, *Environmental/economic power dispatch using multiobjective evolutionary algorithms*, in *Power Engineering Society General Meeting* (IEEE, Toronto, Canada, Jul. 13–17, 2003) pp. 920–925.

[3] H. Akkermans, J. Schreinemakers, and K. Kok, *Emergence of control in a large-scale society of economic physical agents*, in *Proceedings of the International Conference on Autonomous Agents and Multiagent Systems* (New York, NY, USA, Jul. 19–23, 2004) pp. 1232–1234.

[4] A. Al-Fuqaha, M. Guizani, M. Mohammadi, M. Aledhari, and M. Ayyash, *Internet of things: A survey on enabling technologies, protocols, and applications*, IEEE Communications Surveys & Tutorials **17**(4):2347–2376 (2015).

[5] R. Albert and A.-L. Barabási, *Statistical mechanics of complex networks*, Reviews of Modern Physics **74**(1):47–98 (2002).

[6] K. R. Apt and R. N. Bol, *Logic programming and negation: A survey*, The Journal of Logic Programming **19**:9–71 (1994).

[7] A. Arora, P. Dutta, S. Bapat, V. Kulathumani, H. Zhang, V. Naik, V. Mittal, H. Cao, M. Demirbas, M. Gouda, Y. Choi, T. Herman, S. Kulkarni, U. Arumugam, M. Nesterenko, A. Vora, and M. Miyashita, *A line in the sand: a wireless sensor network for target detection, classification, and tracking*, Computer Networks **46**(5):605–634 (2004).

[8] A. P. Athreya, B. DeBruhl, and P. Tague, *Designing for self-configuration and self-adaptation in the internet of things*, in *International Workshop on Internet of Things* (ICST, Austin, TX, USA, Oct. 20–23, 2013) pp. 585–592.

[9] Y. Bar-Shalom and L. Campo, *The effect of the common process noise on the two-sensor fused-track covariance*, IEEE Transactions on Aerospace and Electronic Systems **22**(6):803–805 (1986).

[10] M. Ben-Ari, *First-order logic: Logic programming*, in *Mathematical Logic for Computer Science* (Springer, London, UK, 2012) pp. 205–222.

[11] E. Bertino, K.-K. R. Choo, D. Georgakopolous, and S. Nepal, *Internet of things (IoT): Smart and secure service delivery*, ACM Transactions on Internet Technology **16**(4):22:1–22:7 (2016).

[12] G. Binetti, A. Davoudi, F. L. Lewis, D. Naso, and B. Turchiano, *Distributed consensus-based economic dispatch with transmission losses*, IEEE Transactions on Power Systems **29**(4):1711–1720 (2014).

[13] J. C. Boerkoel and E. H. Durfee, *Distributed reasoning for multiagent simple temporal problems*, Journal of Artificial Intelligence **47**(1):95–156 (2013).

[14] C. Boutilier, *Planning, learning and coordination in multiagent decision processes*, in *Proceedings of the Conference on Theoretical aspects of rationality and knowledge* (De Zeeuwse Stromen, The Netherlands, Mar. 17–20, 1996) pp. 195–210.

[15] J. S. Bradbury, J. R. Cordy, J. Dingel, and M. Wermelinger, *A survey of self-management in dynamic software architecture specifications*, in *Proceedings of the Workshop on Self-managed systems* (ACM SIGSOFT, Newport Beach, CA, USA, Oct. 31–Nov. 1, 2004) pp. 28–33.

[16] N. Cai, N. T. T. Nga, and J. Mitra, *Economic dispatch in microgrids using multi-agent system*, in *North American Power Symposium* (IEEE, Champaign, IL, USA, Sep. 9–11, 2012) pp. 1–5.

[17] F. Cao, J. Liang, and G. Jiang, *An initialization method for the K-means algorithm using neighborhood model*, Computers & Mathematics with Applications **58**(3):474–483 (2009).

[18] R. Carli, A. Chiuso, L. Schenato, and S. Zampieri, *Distributed Kalman filtering based on consensus strategies*, IEEE Journal on Selected Areas in Communications **26**(4):622–633 (2008).

[19] A. Chechetka and K. Sycara, *No-commitment branch and bound search for distributed constraint optimization*, in *Proceedings of the International Conference on Autonomous Agents and Multiagent Systems* (Hakodate, Hokkaido, Japan, May 8–12, 2006) pp. 1427–1429.

[20] D. Chen, Y. Deng, Z. Chen, W. Zhang, and Z. He, *HS-CAI: A hybrid DCOP algorithm via combining search with context-based inference*, in *Proceedings of the Conference on Artificial Intelligence* (AAAI, New York, NY, USA, Feb. 7–12, 2020).

[21] Z. Chen, Y. Deng, and T. Wu, *An iterative refined max-sum_AD algorithm via single-side value propagation and local search*, in *Proceedings*

of the International Conference on Autonomous Agents and Multiagent Systems (São Paulo, Brazil, May 8–12, 2017) pp. 195–202.

[22] Z. Chen, C. He, Z. He, and M. Chen, *BD-ADOPT: a hybrid DCOP algorithm with best-first and depth-first search strategies*, Artificial Intelligence Review **50**(2):161–199 (2018).

[23] M. Choudhury, S. Mahmudand, and M. M. Khan, *A particle swarm based algorithm for functional distributed constraint optimization problems* (2019).

[24] D. Clark, *The design philosophy of the DARPA internet protocols*, ACM SIGCOMM Computer Communication Review **18**(4):106–114 (1988).

[25] L. Cohen and R. Zivan, *Max-sum revisited: The real power of damping*, in *Proceedings of the International Conference on Autonomous Agents and Multiagent Systems* (São Paolo, Brazil, May 8–12, 2017) pp. 1505–1507.

[26] C. M. Colson and M. H. Nehrir, *Comprehensive real-time microgrid power management and control with distributed agents*, IEEE Transactions on Smart Grid **4**(1):617–627 (2013).

[27] H. Dai, Y. Liu, G. Chen, X. Wu, and T. He, *Safe charging for wireless power transfer*, in *Proceedings of the International Conference on Computer Communications* (IEEE, Toronto, Canada, Apr. 27–May 2, 2014) pp. 1105–1113.

[28] H. Dai, Y. Liu, G. Chen, X. Wu, and T. He, *SCAPE: Safe charging with adjustable power*, in *Proceedings of the International Conference on Distributed Computing Systems* (Madrid, Spain, Jun. 30–Jul. 3, 2014) pp. 439–448.

[29] J. O. de Filho, Z. Papp, R. Djapic, and J. Oostveen, *Model-based design of self-adapting networked signal processing systems*, in *Proceedings of the International Conference on Self-Adaptive and Self-Organizing Systems* (Philadelphia, PA, USA, Sep., 2013) pp. 41–50.

[30] Y. Deng, Z. Chen, D. Chen, Xingqiong Jiang, and Q. Li, *PT-ISABB: A hybrid tree-based complete algorithm to solve asymmetric distributed constraint optimization problems*, in *Proceedings of the International Conference on Autonomous Agents and Multiagent Systems* (Montreal, Canada, May 13–17, 2019) pp. 1506–1514.

[31] P. Derler, E. A. Lee, and A. S. Vincentelli, *Modeling cyber–physical systems*, Proceedings of the IEEE **100**(1):13–28 (2012).

[32] G. Di Marzo Serugendo, M.-P. Gleizes, and A. Karageorgos, *Self-organization in multi-agent systems*, Knowledge Engineering Review **20**(2):165–189 (2005).

[33] P. Dolezel, P. Skrabanek, and L. Gago, *Weight initialization possibilities for feedforward neural network with linear saturated activation functions*, in *Proceedings of the Conference on Programmable Devices and Embedded Systems* (IFAC, Brno, Czech Republic, Oct. 5–7, 2016) pp. 49–54.

[34] K. R. Duffy, C. Bordenave, and D. J. Leith, *Decentralized constraint satisfaction*, IEEE/ACM Transactions on Networking **21**(4):1298–1308 (2013).

[35] H. Durrant-Whyte, B. Rao, and H. Hu, *Towards a fully decentralized architecture for multi-sensor data fusion*, in *International Conference on Robotics and Automation* (IEEE, Cincinnati, OH, USA, May 13–18, 1990) pp. 1331–1336.

[36] M. Elidrisi, N. Johnson, M. Gini, and J. Crandall, *Fast adaptive learning in repeated stochastic games by game abstraction*, in *Proceedings of the International Conference on Autonomous Agents and Multiagent Systems* (Paris, France, May 5–9, 2014) pp. 1141–1148.

[37] M. Endler and J. Wei, *Programming generic dynamic reconfigurations for distributed applications*, in *International Workshop on Configurable Distributed Systems* (IET, London, UK, May 25–27, 1992) pp. 68–79.

[38] A. Farinelli, A. Rogers, A. Petcu, and N. R. Jennings, *Decentralised coordination of low-power embedded devices using the max-sum algorithm*, in *Proceedings of the International Conference on Autonomous Agents and Multiagent Systems* (Estoril, Portugal, May 12–16, 2008) pp. 639–646.

[39] S. S. Fatima and M. Wooldridge, *Adaptive task resources allocation in multi-agent systems*, in *Proceedings of the International Conference on Autonomous Agents* (Montréal, Canada, May 28–Jun. 1, 2001) pp. 537–544.

[40] J. Ferber, *Multi-agent systems: An introduction to distributed artificial intelligence* (Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 1999).

[41] M. Feurer, J. T. Springenberg, and F. Hutter, *Initializing Bayesian hyperparameter optimization via meta-learning*, in *Proceedings of the Conference on Artificial Intelligence* (AAAI, Austin, TX, USA, Jan. 25–03, 2015) pp. 1128–1135.

[42] F. Fioretto, A. Dovier, and E. Pontelli, *Distributed multi-agent optimization for smart grids and home automation*, Intelligenza Artificiale **12**(2):67–87 (2019).

[43] F. Fioretto, E. Pontelli, and W. Yeoh, *Distributed constraint optimization problems and applications: A survey*, Journal of Artificial Intelligence **61**(1) (2018).

[44] F. Fioretto, W. Yeoh, E. Pontelli, Y. Ma, and S. Ranade, *A distributed constraint optimization (DCOP) approach to the economic dispatch with demand response*, in *Proceedings of the International Conference on Autonomous Agents and Multiagent Systems* (São Paulo, Brazil, May 8–12, 2017) pp. 999–1007.

[45] S. Fitzpatrick and L. Meertens, *Distributed coordination through anarchic optimization*, in *Distributed Sensor Networks. Multiagent Systems, Artificial Societies, and Simulated Organizations*, Vol. 9, edited by V. Lesser, C. Ortiz, and M. Tambe (Springer, Boston, MA, USA, 2003) pp. 257–295.

[46] J. Fransman, J. Sijs, H. Dol, E. Theunissen, and B. de Schutter, *Distributed Bayesian: a continuous distributed constraint optimization problem solver*, arXiv preprint (2020).

[47] L. Fu, P. Cheng, Y. Gu, J. Chen, and T. He, *Minimizing charging delay in wireless rechargeable sensor network*, in *Proceedings of the International Conference on Computer Communications* (Turin, Italy, Apr. 14–19, 2013).

[48] O. Galinina, K. Mikhaylov, K. Huang, S. Andreev, and Y. Koucheryavy, *Wirelessly powered urban crowd sensing over wearables: Trading energy for data*, IEEE Wireless Communications **25**(2):140–149 (2018).

[49] H. Gao, M. K. Matters-Kammerer, P. Harpe, D. Milosevic, A. van Roermund, J.-P. Linnartz, and P. G. Baltus, *A 60-GHz energy harvesting module with on-chip antenna and switch for co-integration with ULP radios in 65-nm CMOS with fully wireless mm-wave power transfer measurement*, in *Proceedings of the International Symposium on Circuits and Systems* (IEEE, Melbourne, Australia, Jun. 1–5, 2014) pp. 1640–1643.

[50] L. Gasser, C. Braganza, and N. Herman, *Implementing distributed AI systems using MACE*, in *Readings in Distributed Artificial Intelligence* (Morgan Kaufmann Publishers, Inc., 1988) pp. 445–450.

[51] Q. Gemine, D. Ernst, and B. Cornélusse, *Active network management for electrical distribution systems: Problem formulation, benchmark, and approximate solution*, Optimization and Engineering **18**(3):587–629 (2017).

[52] A. Gershman, A. Meisels, and R. Zivan, *Asynchronous forward bounding for distributed COPs*, Journal of Artificial Intelligence **34**(1):61–88 (2009).

[53] R. Glinton, K. P. Sycara, and P. Scerri, *Agent organized networks redux*, in *Proceedings of the Conference on Artificial Intelligence*, Vol. 8 (AAAI, Chicago, IL, USA, Jul. 13–17, 2008) pp. 83–88.

[54] S. Gollakota, M. Reynolds, J. Smith, and D. Wetherall, *The emergence of RF-powered computing*, Computer **47**(1):32–39 (2014).

[55] N. Griffiths and M. Luck, *Changing neighbours: improving tag-based cooperation*, in *Proceedings of the International Conference on Autonomous Agents and Multiagent Systems* (Toronto, Canada, May 10–14, 2010) pp. 249–256.

[56] T. Grinshpoun, A. Grubshtein, R. Zivan, A. Netzer, and A. Meisels, *Asymmetric distributed constraint optimization problems*, Journal of Artificial Intelligence **47**:613–647 (2013).

[57] A. Grubshtein, R. Zivan, T. Grinshpoun, and A. Meisels, *Local search for distributed asymmetric optimization*, in *Proceedings of the International Conference on Autonomous Agents and Multiagent Systems* (Toronto, Canada, May 10–16, 2010) pp. 1015–1022.

[58] S. He, J. Chen, F. Jiang, D. K. Yau, G. Xing, and Y. Sun, *Energy provisioning in wireless rechargeable sensor networks*, IEEE Transactions on Mobile Computing **12**(10):1931–1942 (2013).

[59] K. Hirayama and M. Yokoo, *Distributed partial constraint satisfaction problem*, in *Principles and Practice of Constraint Programming* (Linz, Austria, Oct. 29–Nov. 1, 1997) pp. 222–236.

[60] K. D. Hoang, F. Fioretto, P. Hou, M. Yokoo, W. Yeoh, and R. Zivan, *Proactive dynamic distributed constraint optimization*, in *Proceedings of the International Conference on Autonomous Agents and Multiagent Systems* (Singapore, Singapore, May 9–13, 2016) pp. 597–605.

[61] K. D. Hoang, P. Hou, F. Fioretto, W. Yeoh, R. Zivan, and M. Yokoo, *Infinite-horizon proactive dynamic DCOPs*, in *Proceedings of the International Conference on Autonomous Agents and Multiagent Systems* (São Paulo, Brazil, May 8–12, 2017) pp. 212–220.

[62] K. D. Hoang, W. Y. M. Yokoo, and Z. Rabinovich, *New algorithms for continuous distributed constraint optimization problems*, in *Proceedings of the International Conference on Autonomous Agents and Multiagent Systems* (Auckland, New Zealand, May 9–13, 2020) pp. 502–510.

[63] M. Hu, J.-W. Xiao, S.-C. Cui, and Y.-W. Wang, *Distributed real-time demand response for energy management scheduling in smart grid*, International Journal of Electrical Power & Energy Systems **99**:233–245 (2018).

[64] K. Huang and V. K. Lau, *Enabling wireless power transfer in cellular networks: Architecture, modeling and deployment*, IEEE Transactions on Wireless Communications **13**(2):902–912 (2014).

[65] K. Huang and X. Zhou, *Cutting the last wires for mobile communications by microwave power transfer*, IEEE Communications Magazine **53**(6):86–93 (2015).

[66] IEEE, *European low voltage test feeder network (v2)*, Available online: `https://sites.ieee.org/pes-testfeeders/files/2017/08/European_LV_Test_Feeder_v2.zip` (2015), accessed: Nov. 08, 2018.

[67] N. R. Jennings, *On agent-based software engineering*, Artificial Intelligence **117**(2):277–296 (2000).

[68] D. B. Johnson and D. A. Maltz, *Dynamic source routing in ad hoc wireless networks*, in *Mobile Computing*, edited by T. Imilienski and H. Korth (Springer, Boston, MA, USA, 1996) pp. 153–181.

[69] A. R. Jordehi, *Optimisation of demand response in electric power systems, a review*, Renewable and Sustainable Energy Reviews **103**:308–319 (2019).

[70] S. J. Julier and J. K. Uhlmann, *A non-divergent estimation algorithm in the presence of uknown correlations*, in *Proceedings of the American Control Conference* (IEEE, Albuquerque, NM, USA, Jun. 6, 1997) pp. 2369–2373.

[71] C. Kalialakis and A. Georgiadis, *The regulatory framework for wireless power transfer systems*, Wireless Power Transfer **1**(2):108–118 (2014).

[72] G. Karsai, F. Massacci, L. J. Osterweil, and I. Schieferdecker, *Evolving embedded systems*, Computer **43**(5):34–40 (2010).

[73] G. Karsai and J. Sztipanovits, *A model-based approach to self-adaptive software*, IEEE Intelligent Systems and their Applications **14**(3):4–53 (1999).

[74] I. Kash, A. D. Procaccia, and N. Shah, *No agent left behind: Dynamic fair division of multiple resources*, Journal of Artificial Intelligence **51**(1):579–603 (2014).

[75] J. O. Kephart and D. M. Chess, *The vision of autonomic computing*, Computer **36**(1):41–50 (2003).

[76] C. Kiekintveld, Z. Yin, A. Kumar, and M. Tambe, *Asynchronous algorithms for approximate distributed constraint optimization with quality bounds*, in *Proceedings of the International Conference on Autonomous Agents and Multiagent Systems* (Toronto, Canada, May 10–14, 2010) pp. 133–140.

[77] H. Kim, Y.-J. Kim, K. Yang, and M. Thottan, *Cloud-based demand response for smart grid: Architecture and distributed algorithms*, in *International Conference on Smart Grid Communications* (IEEE, Brussels, Belgium, Oct. 17–20, 2011) pp. 398–403.

[78] S. Kirti and A. Scaglione, *Scalable distributed Kalman filtering through consensus*, in *Proceedings of the International Conference on Acoustics, Speech and Signal Processing* (IEEE, Las Vegas, NV, USA, Mar. 31–Apr. 4, 2008) pp. 2725–2728.

[79] T. van der Klauw, M. E. T. Gerards, G. Hoogsteen, G. J. M. Smit, and J. L. Hurink, *Considering grid limitations in profile steering*, in *International Energy Conference* (IEEE, Leuven, Belgium, Apr. 4–8, 2016) pp. 1–6.

[80] K. Kok and A. Subramanian, *Fast locational marginal pricing for congestion management in a distribution network with multiple aggregators*, in *International Conference and Exhibition on Electricity Distribution* (Madrid, Spain, Jun. 3–6, 2019).

[81] K. Kok and S. Widergren, *A society of devices: Integrating intelligent distributed resources with transactive energy*, IEEE Power and Energy Magazine **14**(3):34–45 (2016).

[82] R. Kota, N. Gibbins, and N. R. Jennings, *Decentralized approaches for self-adaptation in agent organizations*, Transactions on Autonomous and Adaptive Systems **7**(1):1–36 (2012).

[83] S. Kraus and D. Lehmann, *Designing and building a negotiating automated agent*, Computational Intelligence **11**(1):132–171 (1995).

[84] I. Krikidis, *Simultaneous information and energy transfer in large-scale networks with/without relaying*, IEEE Transactions on Communications **62**(3):900–912 (2014).

[85] A. Kumar, A. Petcu, and B. Faltings, *H-DPOP: Using hard constraints for search space pruning in DCOP*, in *Proceedings of the Conference on Artificial Intelligence* (AAAI, Chicago, IL, USA, Jul. 132–17, 2008) pp. 325–330.

[86] R. N. Lass, E. Sultanik, and W. C. Regli, *Dynamic distributed constraint reasoning*, in *Proceedings of the Conference on Artificial Intelligence* (AAAI, Chicago, IL, USA, Jul. 13–17, 2008) pp. 1466–1469.

[87] T. Léauté and B. Faltings, *Protecting privacy through distributed computation in multi-agent decision making*, Journal of Artificial Intelligence **47**:649–695 (2013).

[88] C. J. van Leeuwen, V. H. Díaz, R. Kotian, R. del Toro Matamoros, Z. Papp, and Y. Rieter-Barrell, *An illustrative application example: Cargo state monitoring*, in *Runtime Reconfiguration in Networked Embedded Systems*, edited by Z. Papp and G. Exarchakos (Springer Singapore, 2016) Chap. 6, pp. 137–168.

[89] C. J. van Leeuwen and P. Pawełczak, *CoCoA: A non-iterative approach to a local search (A)DCOP solver*, in *Proceedings of the Conference on Artificial Intelligence* (AAAI, San Fransisco, CA, USA, Feb. 4–11, 2017).

[90] C. J. van Leeuwen and P. Pawełczak, *Hybrid DCOP solvers: Boosting performance of local search algorithms*, in *Proceedings of the International Workshop on Optimization in Multiagent Systems* (Stockholm, Sweden, Jul. 14, 2018).

[91] C. J. van Leeuwen, Y. Rieter-Barrell, Z. Papp, A. Pruteanu, and T. Vogel, *Model-based engineering of runtime reconfigurable networked embedded systems*, in *Runtime Reconfiguration in Networked Embedded Systems*, edited by Z. Papp and G. Exarchakos (Springer Singapore, 2016) Chap. 1, pp. 1–28.

[92] C. J. van Leeuwen, J. Sijs, and Z. Papp, *A reconfiguration framework for self-organizing distributed state estimators*, in *Proceedings of the International Conference on Information Fusion* (IEEE, Istanbul, Turkey, Jul. 9–12, 2013) pp. 499–506.

[93] C. J. van Leeuwen, K. S. Yıldırım, and P. Pawełczak, *Self adaptive safe provisioning of wireless power using DCOPs*, in *Proceedings of the International Conference on Self-Adaptive and Self-Organizing Systems* (Tucson, AZ, USA, Sep. 18–22, 2017).

[94] F. Lezama, E. Munoz de Cote, A. Farinelli, J. Soares, T. Pinto, and Z. Vale, *Distributed constrained optimization towards effective agent-based microgrid energy resource management*, in *Conference on Artificial Intelligence* (EPIA, Villa Real, Portugal, Sep. 3–6, 2019) pp. 438–449.

[95] N. Li, L. Chen, and S. H. Low, *Optimal demand response based on utility maximization in power networks*, in *Power Engineering Society General Meeting* (IEEE, Detroit, MI, USA, Jul. 24–28, 2011) pp. 1–8.

[96] Q. Liu, M. Golinński, P. Pawełczak, and M. Warnier, *Green wireless power transfer networks*, IEEE Journal on Selected Areas in Communications **34**(5):1740–1756 (2016).

[97] Q. Liu, K. S. Yıldırım, P. Pawełczak, and M. Warnier, *Safe and secure wireless power transfer networks: Challenges and opportunities in RF-based systems*, IEEE Communications Magazine **54**(9):74–79 (2016).

[98] W. Liu, P. Zhuang, H. Liang, J. Peng, and Z. Huang, *Distributed economic dispatch in microgrids based on cooperative reinforcement learning*, IEEE Transactions on Neural Networks and Learning Systems **29**(6):2192–2203 (2018).

[99] M. Lützenberger, T. Küster, N. Masuch, and J. Fähndrich, *Multi-agent system in practice: when research meets reality*, in *Proceedings of the International Conference on Autonomous Agents and Multiagent Systems* (Singapore, Singapore, May 9–13, 2016) pp. 796–805.

[100] R. T. Maheswaran, J. P. Pearce, and M. Tambe, *Distributed algorithms for DCOP: A graphical-game-based approach*, in *Proceedings of the International Conference on Parallel and Distributed Computing Systems* (ISCA, San Fransisco, CA, USA, Sep. 15–17, 2004) pp. 432–439.

[101] R. T. Maheswaran, M. Tambe, E. Bowring, J. P. Pearce, and P. Varakantham, *Taking DCOP to the real world: Efficient complete solutions for distributed multi-event scheduling*, in *Proceedings of the International Conference on Autonomous Agents and Multiagent Systems* (New York, NY, USA, Jul. 19–23, 2004) pp. 310–317.

[102] S. Mahmud, M. Choudhury, M. Khan, L. Tran-Thanh, N. R. Jennings, *et al.*, *AED: An anytime evolutionary DCOP algorithm*, arXiv preprint (2019).

[103] M. J. Mataric, *Designing emergent behaviors: From local interactions to collective intelligence*, in *Proceedings of the International Conference on Simulation of Adaptive Behavior* (Honolulu, HI, USA, Apr. 13, 1993) pp. 432–441.

[104] A. Meisels, E. Kaplansky, I. Razgon, and R. Zivan, *Comparing performance of distributed constraints processing algorithms*, in *Proceedings of the International Conference on Autonomous Agents and Multiagent Systems* (Bologna, Italy, Jul. 15–19, 2002) pp. 86–93.

[105] S. Minton, M. D. Johnston, A. B. Philips, and P. Laird, *Minimizing conflicts: a heuristic repair method for constraint satisfaction and scheduling problems*, Artificial Intelligence **58**(1-3):161–205 (1992).

[106] D. Mishra, S. De, S. Jana, S. Basagni, K. Chowdhury, and W. Heinzelman, *Smart RF energy harvesting communications: challenges and opportunities*, IEEE Communications Magazine **53**(4):70–78 (2015).

[107] P. J. Modi, *Distributed Constraint Optimization for Multiagent Systems*, Ph.D. thesis, University of Southern California, Los Angeles, CA, USA (2003).

[108] P. J. Modi, W.-M. Shen, M. Tambe, and M. Yokoo, *ADOPT: asynchronous distributed constraint optimization with quality guarantees*, Artificial Intelligence **161**(1–2):149–180 (2005).

[109] M. Y. Naderi, P. Nintanavongsa, and K. R. Chowdhury, *RF-MAC: A medium access control protocol for re-chargeable sensor networks powered by wireless energy harvesting*, IEEE Transactions on Wireless Communications **13**(7):3926–3937 (2014).

[110] D. T. Nguyen, W. Yeoh, H. C. Lau, S. Zilberstein, and C. Zhang, *Decentralized multi-agent reinforcement learning in average-reward dynamic DCOPs*, in *Proceedings of the Conference on Artificial Intelligence* (AAAI, Québec, Canada, Jul. 27–31, 2014).

[111] I. C. on Non-Ionizing Radiation Protection (ICNIRP), *Guidelines for limiting exposure to time-varying electric, magnetic, and electromagnetic fields (up to 300 GHz)*, Health Physics **74**(4):494–522 (1998).

[112] G. Nordstrom, J. Sztipanovits, and G. Karsai, *Metalevel extension of the multigraph architecture*, in *Proceedings of the Engineering of Computer Based Systems Conference* (IEEE, Jerusalem, Israel, Apr., 1998) pp. 61–68.

[113] S. Okamoto, R. Zivan, and A. Nahon, *Distributed breakout: Beyond satisfaction*, in *Proceedings of the International Joint Conference on Artificial Intelligence* (New York, NY, USA, Jun. 9–15, 2016) pp. 447–453.

[114] S. Parhizi, H. Lotfi, A. Khodaei, and S. Bahramirad, *State of the art in research on microgrids: A review*, IEEE Access **3**:890–925 (2015).

[115] J. P. Pearce and M. Tambe, *Quality guarantees on k-optimal solutions for distributed constraint optimization problems*, in *Proceedings of the International Joint Conference on Artificial Intelligence* (Hyderabad, India, Jan. 6–12, 2007) pp. 1446–1451.

[116] A. Petcu and B. Faltings, *A scalable method for multiagent constraint optimization*, in *Proceedings of the International Joint Conference on Artificial Intelligence* (Edinburgh, Scotland, UK, Jul. 30–Aug. 5, 2005) pp. 266–271.

[117] A. Petcu and B. Faltings, *Superstabilizing, fault-containing distributed combinatorial optimization*, in *Proceedings of the Conference on Artificial Intelligence* (Pittsburgh, PA, USA, Jul. 9–13, 2005) pp. 449–454.

[118] A. Petcu and B. Faltings, *Optimal solution stability in dynamic, distributed constraint optimization*, in *Proceedings of the International Conference on Intelligent Agent Technology* (IEEE/WIC/ACM, Fremont, CA, USA, Nov. 2–5, 2007) pp. 321–327.

[119] M. Pipattanasomporn, H. Feroze, and S. Rahman, *Multi-agent systems in a distributed smart grid: Design and implementation*, in *Power Systems Conference and Exposition* (IEEE, Seattle, WA, USA, Mar. 15–18, 2009) pp. 1–8.

[120] S. Rahnamayan, H. R. Tizhoosh, and M. M. Salama, *A novel population initialization method for accelerating evolutionary algorithms*, Computers & Mathematics with Applications **53**(10):1605–1614 (2007).

[121] A. Rassa, C. van Leeuwen, R. Spaans, and K. Kok, *Developing local energy markets: a holistic system approach*, IEEE Power and Energy Magazine **17**(5):59–70 (2019).

[122] A. Ribeiro, I. D. Schizas, S. I. Roumeliotis, and G. B. Giannakis, *Kalman filtering in wireless sensor networks: Reducing communication cost in state-estimation problems*, IEEE Control Systems Magazine **30**(2):66–86 (2010).

[123] A. Rogers, A. Farinelli, R. Stranders, and N. R. Jennings, *Bounded approximate decentralised coordination via the max-sum algorithm*, Artificial Intelligence **175**(2):730–759 (2011).

[124] D. W. Ross and S. Kim, *Dynamic economic dispatch of generation*, IEEE Transactions on Power Apparatus and Systems **PAS-99**(6):2060–2068 (1980).

[125] G. Rossi, *Uses of Prolog in implementation of expert systems*, New generation computing **4**(3):321–329 (1986).

[126] A. Sarker, A. B. Arif, M. Choudhury, and M. M. Khan, *C-CoCoA: A continuous cooperative constraint approximation algorithm to solve functional DCOPs*, in *Proceedings of the International Conference on Autonomous Agents and Multiagent Systems* (Auckland, New Zealand, May 9–13, 2020) pp. 1990–1992.

[127] D. Sculley, G. Holt, D. Golovin, E. Davydov, T. Phillips, D. Ebner, V. Chaudhary, M. Young, J.-F. Crespo, and D. Dennison, *Hidden technical debt in machine learning systems*, in *Advances in neural information processing systems* (Montréal, Canada, Dec. 7–12, 2015) pp. 2503–2511.

[128] F. K. Shaikh, S. Zeadally, and E. Exposito, *Enabling technologies for green internet of things*, IEEE Systems Journal **11**(2):983–994 (2015).

[129] J. Sijs, *State Estimation in Networked Systems*, Ph.D. thesis, Eindhoven University of Technology, Eindhoven, the Netherlands (2012).

[130] J. Sijs and M. Lazar, *A distributed Kalman filter with global covariance*, in *Proceedings of the American Control Conference* (IEEE, San Francisco, USA, Jun. 29–Jul. 1, 2011) pp. 4840–4845.

[131] J. Sijs and M. Lazar, *State fusion with unknown correlation: Ellipsoidal intersection*, Automatica **48**(8):1874–1878 (2012).

[132] J. Sijs and Z. Papp, *Towards self-organizing Kalman filters*, in *Proceedings of the International Conference on Information Fusion* (IEEE, Singapore, Singapore, Jul. 9–12, 2012) pp. 1012–1019.

[133] M. Simonot and V. Aponte, *A declarative formal approach to dynamic reconfiguration*, in *Proceedings of the International Workshop on Open Component Ecosystems* (Amsterdam, the Netherlands, Aug. 24, 2009) pp. 1–10.

[134] J. R. Smith, *Wirelessly Powered Sensor Networks and Computational RFID* (Springer, New York, NY, USA, 2013).

[135] R. Stärk, *A direct proof for the completeness of SLD-resolution*, in *International Workshop on Computer Science Logic* (Kaiserslautern, Germany, Oct. 2–6, 1989) pp. 382–383.

[136] R. Stranders, L. Tran-Thanh, F. M. D. Fave, A. Rogers, and N. R. Jennings, *DCOPs and bandits: Exploration and exploitation in decentralised coordination*, in *Proceedings of the International Conference on Autonomous Agents and Multiagent Systems* (Valencia, Spain, Jun. 4–8, 2012) pp. 289–296.

[137] A. Tahbaz-Salehi and A. Jadbabaie, *Consensus over ergodic stationary graph processes*, IEEE Transactions on Automatic Control **55**(1):225–230 (2010).

[138] T. Tassa, T. Grinshpoun, and R. Zivan, *Privacy preserving implementation of the max-sum algorithm and its variants*, Journal of Artificial Intelligence **59**:311–349 (2017).

[139] J. S. Vardakas, N. Zorba, and C. V. Verikoukis, *A survey on demand response programs in smart grids: Pricing methods and optimization algorithms*, IEEE Communications Surveys and Tutorials **17**(1):152–178 (2014).

[140] M. A. Velasquez, J. Barreiro-Gomez, N. Quijano, A. I. Cadena, and M. Shahidehpour, *Distributed model predictive control for economic dispatch of power systems with high penetration of renewable energy resources*, International Journal of Electrical Power & Energy Systems **113**:607–617 (2019).

[141] H. J. Visser and R. J. M. Vullers, *RF energy harvesting and transport for wireless sensor network applications: Principles and requirements*, Proceedings of the IEEE **101**(6):1410–1423 (2013).

[142] J. Warrington, S. Mariéthoz, and M. Morari, *Negotiated predictive dispatch: Receding horizon nodal electricity pricing for wind integration*, in *International Conference on the European Energy Market* (IEEE, Zagreb, Croatia, May 25–27, 2011) pp. 407–412.

[143] T. Wu, T. S. Rappaport, and C. M. Collins, *Safe for generations to come: Considerations of safety for millimeter waves in wireless communications*, IEEE Microwave Magazine **16**(2):65–84 (2015).

[144] L. Xiao and S. Boyd, *Fast linear iterations for distributed averaging*, Systems and Control Letters **53**(1):65–78 (2004).

[145] L. D. Xu, W. He, and S. Li, *Internet of things in industries: A survey*, IEEE Transactions on Industrial Informatics **10**(4):2233–2243 (2014).

[146] J. Y. Yam and T. W. Chow, *A weight initialization method for improving training speed in feedforward neural network*, Neurocomputing **30**(1):219–232 (2000).

[147] D. Ye, M. Zhang, and A. V. Vasilakos, *A survey of self-organization mechanisms in multiagent systems*, IEEE Transactions on Systems, Man, and Cybernetics: Systems **47**(3):441–461 (2016).

[148] H. Yedidsion, R. Zivan, and A. Farinelli, *Explorative max-sum for teams of mobile sensing agents*, in *Proceedings of the International Conference on Autonomous Agents and Multiagent Systems* (Paris, France, May 5–9, 2014) pp. 549–556.

[149] W. Yeoh, P. Varakantham, X. Sun, and S. Koenig, *Incremental DCOP search algorithms for solving dynamic DCOP problems*, in *Proceedings of the International Joint Conference on Web Intelligence and Intelligent Agent Technology*, Vol. 2 (IEEE/WIC/ACM, Singapore, Singapore, Dec. 6–9, 2015) pp. 257–264.

[150] W. Yeoh and M. Yokoo, *Distributed problem solving*, AI Magazine **33**(3):53–65 (2012).

[151] F. Ygge and H. Akkermans, *Decentralized markets versus central control: A comparative study*, Journal of Artificial Intelligence **11**(1):301–333 (1999).

[152] M. Yokoo, E. H. Durfee, T. Ishida, and K. Kuwabara, *The distributed constraint satisfaction problem: Formalization and algorithms*, IEEE Transactions on Knowledge and Data Engineering **10**(5):673–685 (1998).

[153] C. Zhang, V. Lesser, and P. Shenoy, *A multi-agent learning approach to online distributed resource allocation*, in *Proceedings of the International Joint Conference on Artificial Intelligence* (Pasadena, CA, USA, Jul. 11-17, 2009) pp. 361–366.

[154] H. Zhang and J. C. Hou, *Maintaining sensing coverage and connectivity in large sensor networks*, Ad Hoc & Sensor Wireless Networks **1**:89–124 (2005).

[155] W. Zhang, G. Wang, X. Zhao, and L. Wittenburg, *Distributed stochastic search and distributed breakout: properties, comparison and applications to constraint optimization problems in sensor networks*, Artificial Intelligence **161**(1):55–87 (2005).

[156] R. Zivan, R. Glinton, and K. Sycara, *Distributed constraint optimization for large teams of mobile sensing agents*, in *Proceedings of the International Joint Conference on Web Intelligence and Intelligent Agent Technology*, Vol. 2 (IEEE/WIC/ACM, Milan, Italy, Sep. 15–18, 2009) pp. 347–354.

[157] R. Zivan, S. Okamoto, and H. Peled, *Explorative anytime local search for distributed constraint optimization*, Artificial Intelligence **212**:1–26 (2014).

[158] R. Zivan, T. Parash, and Y. Naveh, *Applying max-sum to asymmetric distributed constraint optimization*, in *Proceedings of the International Joint Conference on Artificial Intelligence* (Buenos Aires, Argentina, Jul. 25–31, 2015) pp. 432–438.

[159] R. Zivan and H. Peled, *Max/min-sum distributed constraint optimization through value propagation on an alternating DAG*, in *Proceedings of the International Conference on Autonomous Agents and Multiagent Systems* (Valencia, Spain, Jun. 4–8, 2012) pp. 265–272.

Acknowledgements

I would like to thank my supervisors Koen Langendoen and Przemysław Pawełczak for their support and guidance. I will definitely remember Przemek's motivating speeches, when things were looking down, and his joyful celebrations when we had good news. Thank you for all the efforts in co-authoring, proofreading and pushing me into writing the next papers and thesis chapters. I feel that we also got close personally, when there was anything on my mind, I could always reach you. Koen, thank you for your time and support and keen eye for detail, especially during the last stages of this thesis. And of course, thank you for giving me the opportunity to do my doctoral research in your group at the university.

I would also especially like to thank Zoltan Papp, who unfortunately is no longer with us, and did not get to see the results of his coaching. I will remember him as always being very enthusiastic and optimistic about his (and this) work, and has definitely helped me get on the right track to do my research and write this thesis.

Considering the SOSE project, many thanks go to Joris Sijs and Zoltan Papp for co-authoring the paper, and providing their expertise on state estimation. The project was a group effort as a part of the Adaptive Multi Sensor Networks (AMSN) TNO research program. Also, I would like to thank Julio Oliveira, Jan de Gier and Mark Zijlstra for their help with implementing the experiments. Furthermore, the project was made possible by Leon Kester, Ad van Heijningen and Paul Booij—thank you for your contributions.

The wireless power transfer chapter would not have been possible without the help of Kasım Sinan Yıldırım. Thank you very much for bringing the topic to my attention, and for providing this great application for my algorithm. I think we had a good time working together, and I can definitely say that writing this paper was the most fun of all the chapters in this thesis. Also, I have to thank you again for presenting the paper in Arizona when I couldn't: you did a great job in presenting our work, and brought home the Best Paper award.

Considering the final chapter of my thesis, I would like to thank Arun Subramanian for his help with implementing the agent-based algorithm and experiments. I would like to thank Joost Stam for helping me with the formal definition of the problem and the following mathematical analysis. And of course thanks goes to Koen Kok, for providing the scientific guidance and expertise in the energy domain.

Then I would like to thank all the other TNO colleagues and former colleagues who helped me in shaping my research, either directly or

indirectly, or by providing a relevant project or use case. A few people who I owe a special thanks for their contribution as a project leader that (partially) funded my research are Peter Laloli (AMSN), Yolanda Rieter-Barrell (DEMANES), Esther Zondervan (ERP Complexity) and Joost Adriaanse (CERBERO).

I would also like to thank all my colleagues at the TU Delft. Thanks for letting me be part of the group, even though I was sometimes not around for months on end, I always felt welcome and will remember the group outings. I would especially like to thank Andrei Pruteanu, who worked with me on the DEMANES project, and pointed me to DCOPs as a potential driver of self-organization. Also, I would like to thank Marco Cattani and Andreas Loukas for discussing my work, or telling me about theirs, or a chitchat about the more important things in life over a cup of coffee.

A special thanks goes to Elizabeth Busey who very kindly allowed me to use her artwork for the front cover of this thesis. Elizabeth uses the disciplines of printmaking, collage and alternative photography to explore the beauty of natural forms and their underlying mathematical and scientific explanations. See more of her work at http://www.elizabethbusey.com.

Outside my work environment, I would like to thank my dear friends Daan, Niels and Aljo and of course the JC, Folkert, Kees K., Kees L., Sjaak, Sjoerd and Wicher. Thank you for the good times, drinks and food, and all the city trips.

Obviously I would like to thank my parents Jan and Wyp, my sisters Jeanette and Marianne, and the rest of my family for their love and support. Because of you, I am the curious, investigative, problem-solving scientist that I am today. Also, I would like to thank my parents-in-law Arnold and Corine, for welcoming me into their family.

I would like to thank my loving wife, Inge. Thank you for your love and understanding and for the patience over the years. You have been my support for nearly twelve years now, and I hope you will for a long time after this. You know what it is like writing a dissertation, and you have helped me see it through. You have inspired me to keep putting my best into my work, and you will continue to do so. And of course I also have to thank you for bringing so much happiness into my life, which brings me to the last two people I would like to thank.

Alex en Chris, thank you for all your love and joy that you give me. You two are the best thing that happened to me, and you already make me and your mother very proud. Having you has definitely not helped to finish my thesis, but both of you are absolutely the best reason to get my mind off of it.

Curriculum Vitæ

Cornelis Jan van Leeuwen

07-10-1986 Born in Delfzijl, The Netherlands.

Education

2008–2010 Master Artificial Intelligence
University of Groningen
Master Thesis: Driver Modelling and Lane Change Maneuver Prediction
Supervisor: Dr. ir. B. Netten and Dr. M. A. Wiering

2004–2008 Bachelor Artificial Intelligence
University of Groningen
Bachelor Thesis: Object Recognition Based on Local Symmetries
Supervisor: Dr. G. Kootstra

1998–2004 High School
Ommelander College, Appingedam

Experience

2016 Research Scientist
TNO ICT,
Monitoring and Control Services, Groningen

2012–2020 Ph.D. Student (visiting researcher)
Delft University of Technology,
Embedded and Networked Systems

2010–2016 Research Scientist
TNO Technical Sciences,
Intelligent Imaging, The Hague

List of Publications

Publications related to this thesis

[1] C. J. van Leeuwen, J. Sijs, and Z. Papp, *A reconfiguration framework for self-organizing distributed state estimators*, in *Proceedings of the International Conference on Information Fusion* (IEEE, Istanbul, Turkey, Jul. 9–12, 2013) pp. 499–506.
Chapter 2 of this thesis.

[2] C. J. van Leeuwen, J. M. de Gier, J. A. de Oliveira Filho, and Z. Papp, *Model-based architecture optimization for self-adaptive networked signal processing systems*, in *Proceedings of the International Conference on Self-Adaptive and Self-Organizing Systems* (London, UK, Sep. 8–12, 2014) pp. 187–188.

[3] C. J. van Leeuwen, Y. Rieter-Barrell, Z. Papp, A. Pruteanu, and T. Vogel, *Model-based engineering of runtime reconfigurable networked embedded systems*, in *Runtime Reconfiguration in Networked Embedded Systems* edited by Z. Papp and G. Exarchakos (Springer Singapore, 2016) Chap. 1, pp. 1–28.

[4] Z. Papp, R. del Toro Matamoros, C. J. van Leeuwen, J. de Oliveira Filho, A. Pruteanu, and P. Šůcha, *Designing reconfigurable systems: Methodology and guidelines*, in *Runtime Reconfiguration in Networked Embedded Systems*, edited by Z. Papp and G. Exarchakos (Springer Singapore, 2016) Chap. 2, pp. 29–68.

[5] C. J. van Leeuwen, V. H. Díaz, R. Kotian, R. del Toro Matamoros, Z. Papp, and Y. Rieter-Barrell. *An illustrative application example: Cargo state monitoring*, in *Runtime Reconfiguration in Networked Embedded Systems*, edited by Z. Papp and G. Exarchakos (Springer Singapore, 2016) Chap. 6, pp. 137–168.

[6] C. J. van Leeuwen and P. Pawełczak. *CoCoA: A non-iterative approach to a local search (A)DCOP solver*, in *Proceedings of the Conference on Artificial Intelligence* (AAAI, San Fransisco, CA, USA, Feb. 4–11, 2017).
Chapter 3 of this thesis.

[7] C. J. van Leeuwen, K. Sinan Yıldırım, and P. Pawełczak. *Self adaptive safe provisioning of wireless power using DCOPs*, in *Proceedings of the*

International Conference on Self-Adaptive and Self-Organizing Systems (Tucson, AZ, USA, Sep. 18–22, 2017).
Chapter 4 of this thesis. This paper received the "**Best Paper**" award.

[8] C. J. van Leeuwen and P. Pawełczak. *Hybrid DCOP solvers: Boosting performance of local search algorithms*, in *Proceedings of the International Workshop on Optimization in Multiagent Systems* (Stockholm, Sweden, Jul. 14, 2018).
Chapter 5 of this thesis.

[9] A. Rassa, C. J. van Leeuwen, R. Spaans, and K. Kok. *Developing local energy markets: a holistic system approach*, IEEE Power and Energy Magazine **17**(5):59–70 (2019).

Publications unrelated to this thesis

[10] J. Baan, B. Driessen, J. van Huis, and C. J. van Leeuwen. *SPITS road side sensor system*, in *Proceedings of the European Congress on Intelligent Transport Systems*, (Lyon, France, Jun. 6, 2011).

[11] H. Bouma, G. Burghouts, L. de Penning, P. Hanckmann, J. M. ten Hove, S. Korzec, M. Kruithof, S. Landsmeer, C. J. van Leeuwen, S. van den Broek, A. Halma, R. den Hollander, and K. Schutte. *Recognition and localization of relevant human behavior in videos*, in *Sensors, and Command, Control, Communications, and Intelligence Technologies for Homeland Security and Homeland Defense*, edited E. M. Carapezza, (Baltimore, MA, USA, Jun. 6, 2013) 87110B.

[12] G. Burghouts, L. de Penning, J. M. ten Hove, S. Landsmeer, S. van den Broek, R. den Hollander, P. Hanckmann, M. Kruithof, C. J. van Leeuwen, S. Korzec, H. Bouma, and K. Schutte. *A search engine for retrieval and inspection of events with 48 human actions in realistic videos*, in *Proceedings of the International Conference on Pattern Recognition Applications and Methods* (Barcelona, Spain, Feb. 15–18, 2013) pp. 413–418.

[13] C. J. van Leeuwen, A. Halma, and K. Schutte. *Anomalous human behavior detection: an adaptive approach*, in I. Kadar, editor, *Signal Processing, Sensor Fusion, and Target Recognition* (Baltimore, MA, USA, May 23, 2013) 874519.

[14] B. Netten, I. Passchier, H. Wedemeijer, S. Maas, and C. J. van Leeuwen. *Technical evaluation of cooperative systems experience from the DITCM test site*, in *Proceedings of the European Congress on Intelligent Transport Systems* (Dublin, Ireland, Jun. 4–7, 2013).

[15] B. Netten, A. Hegyi, M. Wang, W. Schakel, Y. Yuan, T. Schreiter, B. van Arem, and C. J. van Leeuwen. *Improving moving jam detection*

performance with V2I communication, in *Proceedings of the World Congress on Intelligent Transport Systems* (Tokyo, Japan, Oct. 14–18, 2013).

[16] I. Passchier, B. Netten, H. Wedemeijer, S. Maas, C. J. van Leeuwen, and P. P. Schackmann. *DITCM roadside facilities for cooperative systems testing and evaluation*, in *Proceedings of the International Conference on Intelligent Transportation Systems* (The Hague, Netherlands, Oct. 6–9, 2013) pp. 936–942.

[17] C. J. van Leeuwen, J. van Huis, and J. Baan. *Large scale track analysis for wide area motion imagery surveillance*, in *Optics and Photonics for Counterterrorism, Crime Fighting, and Defence*, edited by D. Burgess, G. Owen, H. Bouma, F. Carlysle-Davies, R. J. Stokes, and Y. Yitzhaky (Edinburgh, UK, Sep. 26–28, 2016) 99950J.

[18] P. Pillegi, J. Verriet, J. Broekhuijsen, C. J. van Leeuwen, W. Wijbrandi, and M. Konsman. *A digital twin for cyber-physical energy systems*, in *Proceedings of the Workshop Modeling and Simulation of Cyber-Physical Energy Systems* (IEEE, Montreal, Canada, Apr. 15, 2019).