UNIVERSIDADE FEDERAL DE SANTA CATARINA
CAMPUS TRINDADE
GRADUATE PROGRAM IN AUTOMATION AND SYSTEM ENGINEERING

Alireza Olama

A Distributed Framework for Sparse Convex Optimization: Algorithms and Software Tools

Florianopolis
2023

Alireza Olama

A Distributed Framework for Sparse Convex Optimization: Algorithms and Software Tools

Ph.D. Dissertation submitted to the Graduate Program in Automation and System Engineering of Universidade Federal de Santa Catarina for the Ph.D. in Automation and Systems Engineering.
Supervisor:: Prof. Eduardo Camponogara, Dr.
Co-supervisor:: Paulo Renato Da Costa Mendes, Dr.

Florianopolis
2023

Ficha de identificação da obra elaborada pelo autor,
através do Programa de Geração Automática da Biblioteca Universitária da UFSC.

Olama, Alireza
   A Distributed Framework for Sparse Convex Optimization
: Algorithms and Software Tools / Alireza Olama ;
orientador, Eduardo Camponogara, coorientador, Paulo
Renato Da Costa Mendes, 2023.
   146 p.

   Tese (doutorado) - Universidade Federal de Santa
Catarina, Centro Tecnológico, Programa de Pós-Graduação em
Engenharia de Automação e Sistemas, Florianópolis, 2023.

   Inclui referências.

   1. Engenharia de Automação e Sistemas. 2. Otimização
convexa esparsa. 4. otimização distribuída. 5. kit de
ferramentas de otimização convexa esparsa. I. Camponogara,
Eduardo . II. Renato Da Costa Mendes, Paulo . III.
Universidade Federal de Santa Catarina. Programa de Pós
Graduação em Engenharia de Automação e Sistemas. IV. Título.

Alireza Olama

A Distributed Framework for Sparse Convex Optimization: Algorithms and Software Tools

O presente trabalho em nível de doutorado foi avaliado e aprovado por banca examinadora composta pelos seguintes membros:

Prof. Marcus Vinicius Soledade Poggi de Aragão, Dr.
Pontifícia Universidade Católica do Rio de Janeiro

Prof. Marcus Ritt, Dr.
Universidade Federal do Rio Grande do Sul

Prof. Erlon Cristian Finardi, Dr.
Universidade Federal de Santa Catarina

Certificamos que esta é a **versão original e final** do trabalho de conclusão que foi julgado adequado para obtenção do título de Ph.D. in Automation and Systems Engineering.

ICP-Edu
Documento assinado digitalmente
Julio Elias Normey Rico
Data: 16/08/2023 09:10:27-0300
CPF: ***.840.859-**
Verifique as assinaturas em https://v.ufsc.br

Prof. Julio Elias Normey Rico, Dr.
Coordenador do Programa

ICP-Edu
Documento assinado digitalmente
Eduardo Camponogara
Data: 15/08/2023 18:19:51-0300
CPF: ***.988.180-**
Verifique as assinaturas em https://v.ufsc.br

Prof. Eduardo Camponogara, Dr.
Orientador

Florianopolis, 2023.

ACKNOWLEDGEMENTS

I would like to express my heartfelt gratitude to Professor Eduardo Camponogara, my esteemed advisor, whose unwavering support, invaluable guidance, and profound insights have been instrumental in shaping this thesis. Your dedication to fostering intellectual growth and your commitment to excellence have been truly inspiring, and I am deeply thankful for the privilege of learning under your mentorship. I extend my sincere appreciation to Professor Giuseppe Notarstefano for his enlightening guidance in distributed optimization during my time at the University of Bologna. Your expertise and encouragement have played a pivotal role in broadening my horizons and refining my academic pursuits. I am immensely grateful to Professor Jan Kronqvist for his invaluable collaboration and contributions to this work. Your insights and expertise have enriched this thesis beyond measure, and I am privileged to have had the opportunity to work alongside you. Your dedication to advancing knowledge and fostering collaboration has been truly inspiring. I must also extend my deepest appreciation to my beloved wife, whose unwavering encouragement and patient support have been my pillars of strength throughout this academic endeavor. Your belief in me and your steadfast presence have been my driving force, and I am grateful beyond words for your love and understanding. Last but not least, I want to express my heartfelt thanks to my family for their constant love, support, and encouragement. This thesis would not have been possible without the contributions, guidance, and unwavering support of these remarkable individuals, and for that, I am truly humbled and grateful.

RESUMO

Esta tese aborda problemas de otimização convexa distribuída que incorporam uma restrição de esparsidade. Conhecido como Otimização Convexa Esparsa (SCO, na sigla em inglês), esse problema é definido em uma rede de computadores, onde vários agentes trabalham juntos para resolver o problema de otimização de forma colaborativa. Devido à restrição de esparsidade ser uma combinação de um número finito de subespaços, o problema SCO se enquadra na classe de otimização combinatória, que geralmente é considerada NP-difícil. Esta tese desenvolveu algoritmos distribuídos eficientes e ferramentas de software para resolver problemas SCO com dados descentralizados. Os algoritmos foram projetados para funcionar em uma rede computacional ponto a ponto, onde cada nó lida com uma parte específica do problema em paralelo, colaborando com outros nós. Inspirada em avanços anteriores em otimização inteira mista e computação de alto desempenho, esta tese introduz um framework de Programação Inteira Mista (MIP) distribuída para encontrar soluções exatas para problemas SCO. O framework apresenta novos algoritmos e heurísticas distribuídos, que são implementados em uma ferramenta de software chamada Conjunto de Ferramentas para Otimização Convexa Esparsa (SCOT, na sigla em inglês), especificamente projetada para resolver problemas SCO. Em particular, os algoritmos propostos estendem algoritmos de Aproximação Externa de Múltiplas e Únicas Árvores (OA) incorporando um algoritmo totalmente descentralizado chamado Método dos Multiplicadores de Direção Alternada Híbrido Relaxado (RH-ADMM, na sigla em inglês). Isso leva ao desenvolvimento de dois algoritmos distribuídos de programação não linear inteira mista: Aproximação Externa Primal Distribuída (DiPOA, na sigla em inglês) e Aproximação Externa Híbrida Distribuída (DiHOA, na sigla em inglês). Além disso, várias técnicas de reformulação e heurísticas são descritas e analisadas, visando aproveitar a separabilidade de funções não lineares e melhorar o desempenho dos algoritmos.

Palavras-chave: Otimização convexa esparsa. programação não linear inteira mista. otimização distribuída. kit de ferramentas de otimização convexa esparsa. aprendizado de máquina

RESUMO EXPANDIDO

INTRODUÇÃO

Aplicações modernas do mundo real frequentemente envolvem problemas de controle e Aprendizado de Máquina que se resumem a otimização matemática com restrição de esparsidade, levando a problemas de Otimização Convexa Esparsa (SCO, do Inglês *Sparse Convex Optimization*). O SCO encontra aplicações em diversos domínios, incluindo regressão logística, análise de *microarrays*, análise estatística, aprendizado de máquina, controle esparso, escalonamento de unidades de energia, design de redes de sensores, otimização de portfólio e amostragem compressiva. A versatilidade do SCO se estende ao processamento de sinais, análise de imagens, visão computacional e processamento de linguagem natural. Apesar do alto desempenho, resolver problemas de SCO pode ser desafiador devido à não-convexidade e descontinuidade, frequentemente exigindo estratégias de aproximação.

OBJETIVOS

O objetivo da tese é abordar três limitações principais nos métodos exatos do estado da arte para problemas de SCO. A primeira limitação diz respeito à natureza distribuída das aplicações. Os métodos existentes são centralizados e não são adequados para cenários envolvendo dados distribuídos ou preocupações com privacidade de dados. A tese tem como objetivo explorar a otimização distribuída em redes para superar essa limitação. A segunda limitação diz respeito à escalabilidade, uma vez que os métodos exatos atuais baseados em *solvers* de programação inteira mista (MIP, do inglês *Mixed-Integer Programming*) enfrentam dificuldades com o crescimento exponencial no número de soluções possíveis à medida que o tamanho do problema e a esparsidade aumentam. Esta tese desenvolveu heurísticas e algoritmos para problemas em larga escala. A terceira limitação envolve o foco em funções objetivo lineares ou quadráticas e restrições lineares, deixando o comportamento não linear em aberto. Esta tese explora algoritmos específicos de programação não linear inteira mista (MINLP, do inglês *Mixed-Integer Non-Linear Programming*) para lidar com problemas complexos de otimização envolvendo variáveis contínuas e discretas no contexto do SCO.

METODOLOGIA

A metodologia desta tese gira em torno da abordagem de problemas de SCO por meio de uma série de algoritmos distribuídos e o desenvolvimento de um framework de software distribuído:

No Capítulo 3, o algoritmo DiGST é apresentado como uma abordagem inicial. Ele adota uma metodologia completamente distribuída com iterações de baixo custo computacional. O algoritmo divide o problema SCO em uma etapa de otimização convexa irrestrita distribuída e uma etapa de projeção de esparsidade. O algoritmo de rastreamento de gradiente é empregado para computação distribuída, garantindo a privacidade dos dados por meio de computação e comunicação locais. No entanto, a aplicabilidade do DiGST a cenários em larga escala é limitada devido a desafios na sintonia do parâmetro de penalização e no tratamento de restrições lineares/não lineares gerais.

Reconhecendo as limitações do DiGST, esta tese propõe um *framework* baseado em MINLP e otimização convexa em larga escala. Esse *framework* é composto por vários algoritmos distribuídos, heurísticas e um conjunto de ferramentas de software chamado SCOT. Cada componente do *framework* é apresentado nos capítulos seguintes.

No Capítulo 4, o algoritmo RH-ADMM é introduzido como um componente fundamental do *framework*. O RH-ADMM é projetado para resolver eficientemente problemas de otimização convexa em larga escala que envolvem restrições de acoplamento. O desenvolvimento deste componente é crucial, pois algoritmos subsequentes dentro do framework dependem fortemente do RH-ADMM para abordar subproblemas convexos de maneira distribuída.

O Capítulo 5 estende a metodologia ao introduzir o algoritmo DiPOA, capaz de resolver problemas SCO de forma distribuída. O DiPOA aproveita a arquitetura de vários núcleos dos processadores modernos para lidar com problemas SCO em larga escala com restrições lineares e não lineares. Para aprimorar o desempenho do DiPOA, o algoritmo DiHOA é introduzido, baseado no DiPOA e em algoritmos LP/NLP BnB. O DiHOA constrói uma árvore BnB (do inglês, *Branch-and-Bound*) inicial com aproximações externas de segunda ordem e introduz dinamicamente mais aproximações usando o RH-ADMM de maneira distribuída. A estratégia de árvore única melhora significativamente a eficiência do DiPOA, conforme demonstrado em benchmarks numéricos.

No Capítulo 6, o *framework* de *software* SCOT é introduzido. O SCOT é uma ferramenta pioneira capaz de lidar com problemas SCO em ambientes de computação distribuída. Ele utiliza a tecnologia MPI, do inglês *Message Passing Interface*, para facilitar computações paralelas e distribuídas. O SCOT é versátil, permitindo que os usuários integrem seus próprios algoritmos e *solvers* de otimização distribuída, sendo adequado para implantação em plataformas de computação de alto desempenho.

CONSIDERAÇÕES FINAIS
A tese conclui demonstrando a superioridade dos algoritmos distribuídos propostos, nomeadamente DiPOA e DiHOA, em termos de qualidade de solução e escalabilidade em comparação com os *solvers* MINLP centralizados existentes.

Palavras-chave: Otimização convexa esparsa. programação não linear inteira mista. otimização distribuída. kit de ferramentas de otimização convexa esparsa. aprendizado de máquina

ABSTRACT

This thesis addresses distributed convex optimization problems that incorporate a sparsity constraint. Referred to as Sparse Convex Optimization (SCO), this problem emerges from a network of computing nodes where various agents work together to solve the optimization problem collaboratively. Due to the sparsity constraint being a combination of a finite number of subspaces, the SCO problem falls under the class of combinatorial optimization, which is typically considered NP-hard. This thesis develops efficient distributed algorithms and software tools to solve SCO problems with decentralized data. The algorithms were designed to work on a peer-to-peer computational network where each node handles a specific portion of the problem in parallel while collaborating with other nodes. Inspired by previous advancements in mixed-integer optimization and high-performance computing, this thesis introduces a distributed Mixed-Integer Programming (MIP) framework to find exact solutions for SCO problems. The framework presents novel distributed algorithms and heuristics, which were implemented in a software tool called the Sparse Convex Optimization Toolkit (SCOT), specifically designed to solve SCO problems. In particular, the proposed algorithms extend multi- and single-tree Outer Approximation (OA) algorithms by incorporating a fully decentralized algorithm called the Relaxed Hybrid Alternating Direction Method of Multipliers (RH-ADMM). Such developments led to the design of two distributed mixed-integer nonlinear programming algorithms: Distributed Primal Outer Approximation (DiPOA) and Distributed Hybrid Outer Approximation (DiHOA). Additionally, various reformulation and heuristic techniques were introduced to leverage the separability of nonlinear functions and enhance performance.

Keywords: Sparse convex optimization. mixed-integer nonlinear programming. distributed optimization. sparse convex optimization toolkit. machine learning.

LIST OF FIGURES

LIST OF TABLES

CONTENTS

1 INTRODUCTION

A broad class of modern real-world applications of control and Machine Learning (ML) problems consists of mathematical optimization problems with a constraint that allows only up to a certain number of variables to be nonzero. This form of constraint is called a *sparsity constraint* and we refer to any convex optimization problem containing a sparsity constraint as a *Sparse Convex Optimization* (SCO) problem (BIENSTOCK, 1996a; BERTSIMAS; MUNDRU, 2021a; BERTSIMAS; CORY-WRIGHT, 2022; SUN, X.; ZHENG; LI, D., 2013; TILLMANN et al., 2021). Formally, we define the SCO problem as a mathematical programming problem that consists of finding the κ-sparse optimal solution of a convex optimization problem of the following form,

$$\begin{aligned} \min_{\mathbf{x}\in\mathcal{R}^n} \ & f(\mathbf{x}) \\ \text{subject to } & \mathbf{x}\in\Omega \\ & \|\mathbf{x}\|_0 \leq \kappa \end{aligned} \tag{SCO}$$

where $\mathbf{x} \in \mathcal{R}^n$ is the vector of decision variables, $f : \mathcal{R}^n \to$ is a continuously differentiable convex function, and Ω is a closed convex set with no empty interior which represents the general constraints for $\mathbf{x}$. We use the ℓ_0 norm (*i.e.*, $\|\mathbf{x}\|_0 = |supp(\mathbf{x})| = |\{j : x_j \neq 0, j = 1, \ldots, n\}|$) to define the sparsity constraint, which imposes the number of non-zero elements of $\mathbf{x}$ to be less than a given integer κ. The SCO problems have numerous practical applications in various fields, including:

- *Logistic (Linear) regression*: In this application, the goal is to find a sparse logistic (linear) regression model for binary classification problems, where only a few variables are selected as predictors (BERTSIMAS; KING, 2017b).
- *Block-wise linear regression*: This involves fitting a linear regression model where the variables are grouped into blocks, and only a small number of variables from each block are selected as predictors (KIM, Yuwon; KIM, J.; KIM, Yongdai, 2006).
- *Microarray analysis*: SCO is used to identify important genes from gene expression data, where only a small number of genes are relevant to a specific disease or condition (MA, S.; SONG, X.; HUANG, 2007).
- *Statistical analysis and machine learning*: SCO is widely used in various statistical analysis and machine learning problems, such as best subset selection, feature selection, model regularization, factor analysis, sparse principal component analysis, matrix and tensor completion, and sparsity-based clustering (BERTSIMAS; MUNDRU, 2021a, 2021b; BERTSIMAS; KING; MAZUMDER, 2016; BERTSIMAS et al., 2022; BERTSIMAS; CORY-WRIGHT, 2022).

- *Sparse control*: In this application, SCO is used to design controllers for systems with sparse input or state variables, such as power systems and chemical processes (AGUILERA et al., 2017).
- *Unit commitment*: This involves scheduling the operation of power generation units in a power grid to meet the demand while minimizing the operating cost, subject to various operational constraints. SCO is used to select the optimal subset of units to operate at each time step (FRANGIONI; GENTILE, 2006; FRANGIONI; GENTILE; LACALANDRA, 2008).
- *Sensor network design*: SCO is used to select a small number of sensors from a large set of candidates to optimize the performance of a sensor network, subject to various resource constraints (LEWIS, 2004).
- *Portfolio optimization*: This involves selecting a subset of assets from a large set of candidates to optimize the performance of a portfolio, subject to various risk and return constraints. SCO is used to identify the optimal subset of assets to invest in (BERTSIMAS; CORY-WRIGHT, 2022).
- *Compressed sensing*: In this application, the goal is to reconstruct a sparse signal from a small number of linear measurements. SCO is used to find the sparsest solution that satisfies the measurement constraints (FOUCART; RAUHUT, 2013).

While the applications mentioned here are diverse and important, they are by no means exhaustive. There are numerous other fields where SCO has found valuable applications, such as signal processing, image analysis, computer vision, and natural language processing. In signal processing, SCO has been used for speech recognition, channel equalization, and audio signal processing. In image analysis and computer vision, SCO has found applications in object recognition, feature selection, and image segmentation. In natural language processing, SCO has been used for sentiment analysis, text classification, and named entity recognition. The versatility of SCO and its ability to handle high-dimensional, sparse data make it an attractive tool for a wide range of applications (TILLMANN et al., 2021).

Although sparse optimization is a powerful tool in many practical applications, finding the optimal solution can be challenging due to the non-convexity and discontinuity of the sparsity constraint (see Figure 1). This results in a combinatorial optimization problem that is difficult to solve, especially for large instances of practical problems. Various studies have shown that finding the optimal solution is generally considered an NP-hard task (BAI, Y.; LIANG; YANG, Z., 2016a; BERTSIMAS; CORY-WRIGHT, 2022; NATARAJAN, 1995). Due to the numerical challenges stemming from the sparsity constraint, many solution algorithms attempt to reformulate or approximate the original problem to simplify its handling. In the following section, we review some common ap-

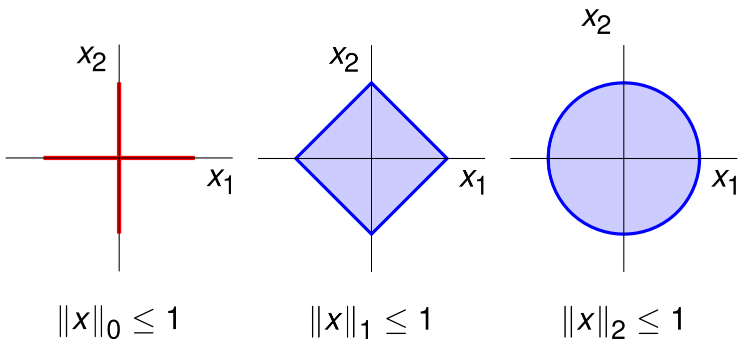


Figure 1 – Geometric interpretation of sparsity constraint in comparison with ℓ_1 and ℓ_2 norms

proaches that have been proposed in the literature for this purpose, along with their advantages and limitations.

1.1 STATE-OF-THE-ART

In the realm of SCO problems, the majority of algorithmic solutions are often categorized as either *approximation* or *exact* methods. The former seeks to provide a good convex representation of the sparsity constraint and attains the desired sparsity through solving a sequence of *convex* optimization problems. The exact methods, on the other hand, try to view the SCO problem as a Mixed Integer Programming (MIP) problem since it can be reformulated in such a way by introducing suitable binary variables (BERTSIMAS; KING; MAZUMDER, 2016; BERTSIMAS; CORY-WRIGHT, 2022; BERTSIMAS; KING, 2017b; BERTSIMAS; MUNDRU, 2021a).

The ℓ_1 convex relaxation method is a popular approximation technique as it approximates the sparsity constraint using the unit ℓ_∞ norm. These methods are widely applied in solving feature selection problems found in the statistical learning literature (TIBSHIRANI, 1996; BOYD, S. et al., 2011; TIAN; ZHANG, Yuqi, 2022). One of the important reasons behind the popularity of ℓ_1 based methods is their computational efficiency and scalability to practical-sized problems. However, in spite of their favorable computational properties, these methods can have some shortcomings which motivates the development of exact methods. For example, they cannot guarantee that ℓ_1 based methods find the correct sparsity for general problems. Moreover, in some applications, the desired sparsity structure is different from general sparsity and cannot be easily obtained by a ℓ_1 regularization. An example of such a sparse structure is *group sparsity* in which a block or a group of independent variables are either all zero or all nonzero. Some notable applications with group sparsity are block-wise linear regression (KIM, Yuwon; KIM, J.; KIM, Yongdai, 2006), logistic regression (BERTSIMAS; KING, 2017a), compressed sensing (ELDAR; KUTYNIOK, 2012), and microarray analysis (MA, S.; SONG, X.; HUANG, 2007). Considering recent advances in mixed-integer optimization algorithms and technologies, the MIP problems can be solved efficiently by current mixed-integer optimization solvers such as Gurobi (GUROBI OPTIMIZATION, 2020). The resulting MIP formulation is flexible and can be adjusted based on the application's needs.

This thesis focuses on exact methods for solving optimization problems, with a specific emphasis on MIP reformulations for SCO problems. We review the current state-of-the-art and aim to contribute to the development of more efficient optimization methods. As the first attempt, (BIENSTOCK, 1996b) proposed a Mixed Integer Quadratic Programming (MIQP) problem to reformulate the SCO problem with a quadratic objective and linear constraints. The problem is then solved by employing a tailored branch-and-cut algorithm. (BERTSIMAS; SHIODA, 2009) proposed a tailored algorithm based on the branch-and-bound method to solve the SCO problem appearing in sparse linear regression and portfolio selection problems. (AGUILERA et al., 2017) provided a MIQP formulation for solving a sparsity constrained model predictive control problem. Although the methodology works efficiently for the given application, it lacks generalization to more realistic scenarios.

As another instance, (BERTSIMAS; CORY-WRIGHT, 2022) considered a scalable algorithm to solve cardinality constrained portfolio optimization problems. This paper also proposed a MIQP framework along with multiple improvements and heuristics such as high-quality warm-starts, a prepossessing technique, and so on. (AYTUG, 2015) proposed a MIQP based framework to solve a sparsity constrained support vector machine problem. In this paper, the authors utilized the Generalized Benders Decomposition (GBD) algorithm to select the best subset of features during the model training. As another application, (BERTSIMAS; KING; MAZUMDER, 2016) proposed an efficient algorithm to obtain an exact solution for the best subset selection problem in linear regression. In this paper, the authors proposed a MIQP reformulation for the resulting SCO problem. The resulting MIQP problem is then solved by a tailored branch and bound algorithm. As another instance, a Mixed-Integer Nonlinear Programming (MINLP) model to solve sparse classification problems is proposed by (BERTSIMAS; KING, 2017b; BERTSIMAS; PAUPHILET; VAN PARYS, 2021). The resulting MINLP model is then solved by utilizing the Outer Approximation (OA) algorithm (GROSSMANN, I., 2002; KRONQVIST et al., 2019). (KAMIYA; MIYASHIRO; TAKANO, 2019) consider a feature subset selection problem for the multinomial logistic model with ℓ_2 regularization. This problem is then transformed into a SCO problem which in turn is solved with the OA algorithm. Additionally, (BERTSIMAS; MUNDRU, 2021a) provide a MINLP formulation to solve a sparse convex regression problem. Besides the decomposition methods (*e.g.*, OA algorithm), some works focused on single-tree methods such as branch-and-bound, branch-and-cut, and branch-and-price algorithms. For example, (WANG, F.; CAO, 2020) used the branch-and-bound method, with domain cut and partition scheme, to solve a quadratic sparsity constrained problem that has application to portfolio optimization.

As another approach, (SANT'ANNA et al., 2020) proposed an MINLP modeling framework along with a SCO problem to model and solve the index tracking problem. The problem is then solved by a variant of the branch-and-cut algorithm. There exist relatively few works where a general convex MINLP model is used to reformulate the SCO problem. For instance, instead of formulating the SCO problem for a particular application, (BAI, Y.; LIANG; YANG, Z., 2016b) considered a general convex formulation for the SCO problems. The authors proposed a splitting augmented Lagrangian method to solve the resulting SCO problem. The efficiency of the proposed algorithm is then tested on portfolio selection and compressed sensing problems.

1.2 MOTIVATION

Following our literature review in the previous section, we have identified three primary categories of limitations in the current state-of-the-art exact methods. In the subsequent sections, we will provide a detailed discussion of these categories.

Distributed Nature of Applications

The majority of the existing exact methods discussed in the previous section focused on *centralized* solutions to SCO problems. This means that they require all of the data to be available and processed by a central node or processor, which can lead to significant communication overheads and limit the scalability of the method. However, the applicability of such solutions may be limited in modern real-world scenarios where data is either inherently distributed or available in large volumes (LIU et al., 2022). For instance, in certain instances of regression and classification problems, the dataset is distributed over a network, and data privacy policies do not allow the collection of the entire dataset in a single point. Similar situations can also arise in applications such as energy management of renewable power systems, distributed compressed sensing, and so on, making a distributed solution imperative.

Due to these limitations, there has been a growing interest in distributed optimization over networks in recent years, particularly with the emergence of Big Data (NOTARSTEFANO; NOTARNICOLA; CAMISA, et al., 2019; NOTARNICOLA et al., 2017). Distributed optimization aims to solve an optimization problem over a network of computing nodes, where each node performs local computations with access to only a portion of the problem data. The nodes can also exchange information with other nodes in the network. Several distributed optimization algorithms have been proposed in the literature, and a comprehensive overview of these can be found in (NOTARSTEFANO; NOTARNICOLA; CAMISA, et al., 2019). However, it is essential to note that the distributed reformulation of the SCO problem can increase the complexity of the problem due to the coupling behavior in either the objective function or constraints.

Scalability

The next limitation of current exact methods for ℓ_0-based sparse optimization problems is related to the scalability of these methods. In general, exact methods based on MIP solvers have been shown to have poor scalability as the problem size grows. This is due to the exponential increase in the number of possible solutions with respect to the problem size. Specifically, the number of binary variables required to encode a solution grows exponentially with the number of non-zero entries in the solution, which is in turn an upper bound on the sparsity of the solution. This leads to a combinatorial explosion in the number of possible solutions as the sparsity level increases, making it computationally intractable to solve large-scale problems. Therefore it is essential to develop various heuristics and algorithms that can be used for problems with large amounts of data.

Tailored MINLP Algorithms

The last limitation of the current literature that we aim to address in this thesis is that the existing exact methods have been primarily applied to problems with a linear or quadratic objective function and linear constraints leading to MILP or MIQP problems. Such a focus can be attributed to the relatively tractable nature of these problems, which allows for the development of efficient algorithms and solutions. However, the applicability of such solutions is limited in scenarios where the optimization problem exhibits nonlinear behavior. In particular, there are only a few works that have considered tailored MINLP algorithms in the context of SCO. MINLP problems involve optimization problems that include both continuous and discrete variables and have a nonlinear objective function or constraints. The use of MINLP models in SCO is of significant interest, as it allows for the incorporation of various real-world constraints and objectives that cannot be easily represented in a linear or quadratic optimization framework. Notably, a recent work by Bai et al. (BAI, Y.; LIANG; YANG, Z., 2016b) proposes a method for solving general convex MINLP problems in the context of SCO, providing a framework for tackling complex optimization problems that include both continuous and discrete variables. Despite the recent advances, further research is required to develop more efficient and robust algorithms for solving general convex MINLP problems in the context of SCO.

1.3 APPLICATIONS

In this thesis, we examine two significant applications of SCO problems that have broad applications in statistics and machine learning. Our attention is directed toward the problems of Distributed Sparse Linear Regression (DSLinR) and Distributed Sparse Logistic Regression (DSLogR), which we elaborate on below. The DSLinR problem aims to find a sparse linear regression model in a decentralized network of agents. Specifically, given the local dataset X_i and response vector b_i of the i-th agent and a regularization parameter λ, the objective is to minimize the sum of squared residuals subject to a sparsity constraint that limits the non-zero coefficients in the solution vector $\boldsymbol{\theta}$. The DSLinR problem is defined as,

$$\begin{aligned} \min_{\boldsymbol{\theta}} \sum_{i=1}^{N} \|\mathrm{X}_i\boldsymbol{\theta} - \mathrm{b}_i\|_2^2 + \frac{\lambda}{2}\|\boldsymbol{\theta}\|_2^2 \\ \text{subject to } \|\boldsymbol{\theta}\|_0 \leq \kappa \end{aligned} \tag{DSLinR}$$

On the other hand, the DSLogR problem is a sparse logistic regression problem that also operates in a decentralized network. The objective is to minimize the negative log-likelihood of the logistic regression model subject to a sparsity constraint. The model is defined by the solution vector $\boldsymbol{\theta}$ and the local dataset X_i, and the corresponding response vector $\boldsymbol{\Gamma}_i$.

$$\begin{aligned} \min_{\boldsymbol{\theta}} \sum_{i=1}^{N} \left[\log\left(1 + e^{-(\boldsymbol{\theta}^T\mathrm{X}_i)\boldsymbol{\Gamma}_i}\right)\right] + \frac{\lambda}{2}\|\boldsymbol{\theta}\|_2^2 \\ \text{subject to } \|\boldsymbol{\theta}\|_0 \leq \kappa \end{aligned} \tag{DSLogR}$$

In both DSLogR and DSLinR, the dataset is assumed to be distributed among agents in a computational network, and the solution's sparsity is of significance. Sparse classification and regression problems are tightly connected to SCO problems as it is often desired to identify a critical subset of features contributing to the response. Furthermore, the sparse solution usually leads to more interpretable models and improves prediction accuracy by eliminating unnecessary features.

1.4 CONTRIBUTIONS OF THE THESIS

The central contribution of this thesis is the development of a distributed optimization framework that is capable of addressing SCO problems across computational networks. This framework involves the development of several novel distributed algorithms and heuristics, which are implemented within a dedicated software tool named the Sparse Convex Optimization Toolkit (SCOT). To the author's knowledge, SCOT is the first tool that is specifically designed and implemented to solve SCO problems.

The proposed distributed framework incorporates techniques from distributed convex optimization and mixed-integer nonlinear programming. Each chapter builds on the previous one, leading to several key contributions which include:

- The development of the Distributed Sparse Gradient Tracking (DiSGT) algorithm, which solves SCO problems without linear and/or nonlinear constraints using inexpensive computational steps and information exchange with neighboring nodes.
- The development of the Relaxed-Hybrid Alternating Direction Method of Multiplier algorithm (RH-ADMM), a fully distributed algorithm that solves large-scale convex optimization problems with coupling constraints. This algorithm is used as a distributed numerical engine for subsequent algorithms.
- The design of the Distributed Primal Outer Approximation (DiPOA) algorithm, which solves SCO problems with both linear and nonlinear constraints. This algorithm extends the centralized multiple-tree Outer Approximation (OA) algorithms for mixed-integer nonlinear programming problems to distributed environments.
- The development of the Distributed Hybrid Outer Approximation (DiHOA) algorithm, which improves DiPOA performance by utilizing lazy constraints and single-tree OA strategy.
- The development of various heuristics, including a specialized feasibility pump, event-triggered second-order cut generation, bound tightening, and practical infeasibility detection, which enhance the performance and solution quality of DiPOA and DiHOA algorithms.
- The development of SCOT, an open-source software tool that implements the proposed distributed optimization framework and its algorithms.

1.5 LIST OF PUBLICATIONS

This thesis is based on the results presented in the following papers.

- **A. Olama**, E. Camponogara, and J. Kronqvist, “Sparse Convex Optimization Toolkit: A Mixed-Integer Framework,”*Optimization Methods and Software*, 2023. `https://doi.org/10.1080/10556788.2023.2222429`
- **A. Olama**, G. Carnevale, G. Notarstefano, and E. Camponogara,“A Tracking Augmented Lagrangian Method for ℓ_0 Sparse Consensus Optimization,” in *9th International Conference on Control, Decision and Information Technologies (CoDIT)*, Roma, Italy, 2023.
- **A. Olama**, E. Camponogara, and P. R. Mendes, “Distributed primal outer approximation algorithm for sparse convex programming with separable struc-

tures," *Journal of Global Optimization*, pp. 1-34, 2023 (to appear). `https://doi.org/10.1007/s10898-022-01266-5`

- **A. Olama**, N. Bastianello, P.R. Mendes, and E. Camponogara, "Relaxed Hybrid Consensus ADMM for Distributed Convex Optimisation with Coupling Constraints," *IET Control Theory & Applications*, pp. 2828-2837, 2019. `https://doi.org/10.1049/iet-cta.2018.6260`

1.6 DISSERTATION ORGANIZATION

The structure of this thesis is as follows: Chapter 2 briefly covers the main mathematical background utilized in this thesis. Chapter 3 introduces the DiGST algorithm, which is used for SCO problems. Chapter 4 discusses distributed convex optimization using operator theory and fixed point iterations and introduces the RH-ADMM algorithm for solving constrained-coupled convex optimization problems. Building on the ideas presented in Chapter 4, Chapter 5 presents the DiPOA and DiHOA algorithms, as well as several heuristics for improving their performance. Chapter 6 presents SCOT, including its architecture, software components, design, and use cases. Finally, Chapter 7 provides some concluding remarks.

2 MATHEMATICAL BACKGROUND

This section provides an overview of important concepts that are central to this thesis. We will begin by discussing distributed optimization and its variants, followed by exploring the relationship between operator theory and convex optimization. Lastly, we will review key topics in mixed-integer linear and nonlinear optimization.

2.1 DISTRIBUTED COMPUTATION MODELS FOR OPTIMIZATION

This section serves as an introduction to the conceptual framework for distributed optimization in peer-to-peer networks. For a more comprehensive survey, interested readers may refer to (NOTARSTEFANO; NOTARNICOLA; CAMISA, et al., 2019). In a distributed computing scenario, we work with a group of N units, known as agents or processors, each with communication and computation capabilities. Communication between agents is modeled using graph theory. Essentially, a graph $\mathcal{G}$ with N nodes, each representing an agent, is constructed. An agent i can communicate with another agent j if an edge exists between them in the graph $\mathcal{G}$, i.e., if i and j are connected. In a distributed algorithm, the agents initialize their local states and proceed with an iterative process. During each iteration, computation and communication steps are performed in a synchronized manner, with all the agents executing the same actions. The iterative process continues until the algorithm reaches convergence or the desired result is obtained.

2.1.1 Distributed Computation Model

Here, we define the communication model for distributed algorithms considered in this thesis. We model a network as an undirected graph $\mathcal{G} = (\{1,\ldots,N\},\mathcal{E})$, where $\{1,\ldots,N\}$ is the fixed set of agent identifiers and $\mathcal{E} \subseteq \{1,\ldots,N\} \times \{1,\ldots,N\}$ is the set of edges over the vertices $\{1,\ldots,N\}$, which represents the communication links. If there is an edge $(i,j) \in \mathcal{E}$, we say that i is the *in-neighbor* of j and j is the *out-neighbor* of i.

Definition 1. *A graph $\mathcal{G}$ is said to be connected if for every pair of nodes (i,j) there exist a path of edges that goes from i to j.*

Agents can run distributed algorithms according to several communication protocols on a given network topology. If the algorithm's steps depend explicitly on a particular time, we call it a *synchronous* algorithm; otherwise, it is called an *asynchronous* algorithm. *synchronous* otherwise it is called *asynchronous*. We use two commonly employed optimization setups, namely *cost-coupled* and *constraint-coupled* optimization, in this thesis. These optimization problems are extensively used in various applications, such as estimation, learning, decision-making, and control in smart networks. A distributed optimization algorithm for these problem classes comprises an iterative process based on the previously defined distributed computation model. The objective for the agents is to achieve a solution to the problem under consideration. In each of the two optimization setups, this objective leads to different expressions, which will be formally defined below.

2.1.2 Cost-Coupled Optimization

The cost-coupled optimization setup expresses the cost function as the sum of local contributions f_i with a global optimization variable $\mathbf{x}$ and is defined as,

$$\begin{aligned} \min_{\mathbf{x}\in\mathbb{R}^n} \quad & \sum_{i=1}^{N} f_i(\mathbf{x}) \\ \text{subject to } & \mathbf{x} \in \mathcal{X} \end{aligned} \tag{1}$$

where $\mathcal{X} \subseteq \mathbb{R}^n$. Each agent is assumed to know its own local contribution $f_i : \mathbb{R}^n \to \mathbb{R}$, while the global constraint set $\mathcal{X}$ is known by all agents. In some cases, the constraint set may be more structured, taking the form of $\mathcal{X} = \bigcap_{i=1}^{N} \mathcal{X}_i$, where each agent knows only its own constraint set $\mathcal{X}_i$. Let $\mathbf{x}^*$ denote an optimal solution to problem (1). For this optimization set-up, the goal is to design a distributed algorithm where each agent updates a local estimate $\mathbf{x}_i^k$ that converges (asymptotically or in finite time) to $\mathbf{x}^*$, by means of local computation and limited communications.

2.1.3 Constraint-Coupled Optimization

Now, we present a different set-up which is called constraint-coupled where agents in a network aim to minimize the sum of local cost functions, each one depending only on a local vector satisfying local constraints. The decision vectors are then coupled to each other by means of separable coupling constraints. This feature leads easily to the so-called big-data problems having a highly dimensional decision variable that grows with the network size. However, since agents are typically interested in computing only their (small) portion of an optimal solution, novel tailored methods need to be developed to address these challenges. Formally, the constraint-coupled optimization problem is define as,

$$\begin{aligned} \min_{\mathbf{x}_1,\dots,\mathbf{x}_N} \quad & \sum_{i=1}^{N} f_i(\mathbf{x}_i) \\ \text{subject to } & \mathbf{x}_i \in \mathcal{X}_i \\ & \sum_{i=1}^{N} g_i(\mathbf{x}_i) \leq 0 \end{aligned} \tag{2}$$

where f_i and g_i are known by agent i only. Notice that problem (2) is challenging because of the coupling constraints $\sum_{i=1}^{N} g_i(\mathbf{x}_i) \leq 0$. If there were no coupling constraints, the optimization would trivially split into N independent problems. Let $(\mathrm{x}_1^* \ldots, \mathrm{x}_N^*)$ denote an optimal solution of problem (2). The goal is to design a distributed algorithm where each agent updates a local estimate x_i^k that converges (asymptotically or in finite time) to x_i^*, the i-th portion of$(\mathrm{x}_1^* \ldots, \mathrm{x}_N^*)$ by means of local computation and neighboring communication only. A special instance of this set-up has been investigated in the context of resource allocation, where the coupling constraint is linear, *e.g.*, $\sum_{i=1}^N \mathrm{x}_i = \mathrm{b}$, and there are no local constraints.

2.2 OPERATOR THEORY AND CONVEX OPTIMIZATION

In what follows, we provide an overview of the fundamentals of convex optimization and operators and fixed point theory that are used throughout the research. For more details see (BOYD, S.; BOYD, S. P.; VANDENBERGHE, 2004; BERTSEKAS, 2015; RYU; BOYD, S., 2016, text)

2.2.1 Convexity

Basic fundamentals of convex sets and convex functions are addressed here. We first present the definition of convex sets.

Definition 2. *A subset $\mathcal{C}$ of $\mathcal{R}^n$ is called convex if*

$$\alpha x + (1-\alpha) y \in \mathcal{C}, \qquad \forall x, y \in \mathcal{C}, \forall \alpha \in [0,1]$$

Note that the empty set is by convention considered to be convex. We often consider some special convex sets, which are defined in the following definitions.

Definition 3 (Hyperplane)**.** *A hyperplane is a set specified by a single linear equality, i.e., a set of the following form*

$$\{x \in \mathcal{R}^n \mid a^T x = b, a \in \mathcal{R}^n\}.$$

Definition 4 (Halfspace)**.** *A halfspace is a set specified by a single linear inequality, i.e., a set of the following form*

$$\{x \in \mathcal{R}^n \mid a^T x \leq b, a \in \mathcal{R}^n\}.$$

Definition 5 (Polyhedron)**.** *A polyhedron is a set specified by the intersection of a set of halfspaces, i.e., a set of the following form*

$$\{x \in \mathcal{R}^n \mid a_i^T x \leq b_i, a_i \in \mathcal{R}^n, i = 1,\ldots,m\}.$$

A bounded polyhedron is called a polytope. It is clear that the defined special sets are convex. Next, the basics of convex functions are presented.

Definition 6. *Let $\mathcal{C}$ be a convex subset of $\mathcal{R}^n$. A function $f : \mathcal{C} \to \mathcal{R}$ is convex if*

$$f(\alpha x + (1-\alpha)y) \leq \alpha f(x) + (1-\alpha)f(y), \quad \forall x,y \in \mathcal{C}, \forall \alpha \in [0,1] \tag{3}$$

Moreover, a function f is called strictly convex if the inequality (3) is strict for all $x,y \in \mathcal{C}$ with $x \neq y$ and $\alpha \in (0,1)$. Note that, according to the definition, convexity of the domain $\mathcal{C}$ is a prerequisite for convexity of the function f. For once or twice differentiable functions, there are some additional criteria for verifying convexity, which are presented in the following.

Proposition 1. *Let $\mathcal{C}$ be a convex subset of $\mathcal{R}^n$ and let $f : \mathcal{R}^n \to \mathcal{R}$ be differentiable over an open set that contains $\mathcal{C}$. Then f is convex over $\mathcal{C}$ if and only if*

$$f(z) \geq f(x) + \nabla f(x)^T (z-x), \forall x,z \in \mathcal{C}. \tag{4}$$

Proposition 2. *Let $\mathcal{C}$ be a convex subset of $\mathcal{R}^n$ and let $f : \mathcal{R}^n \to \mathcal{R}$ be continuously twice differentiable over an open set that contains $\mathcal{C}$. Then f is convex over $\mathcal{C}$ if and only if*

$$\nabla^2 f(x) \succeq 0, \ \forall x \in \mathcal{C}. \tag{5}$$

In other words, for a twice differentiable function, convexity can be checked through the sign of the Eigenvalues of the Hessian of f at point x. Below, the fundamentals of convex optimization problems are discussed.

2.2.2 Convex Optimization

In this section, the basic terminology and optimality conditions of the convex optimization problem are also discussed.

The general form of a convex optimization problem is written as follows

$$\begin{aligned} &\text{minimize } f(x) \\ &\text{subject to } g_i(x) \leq 0, \forall i = 1,...,m \\ &\qquad\qquad a_i^T x = b_i, \forall i = 1,...,p \end{aligned} \tag{6}$$

where $f : \mathcal{R}^n \to \mathcal{R}$ is called the objective function which is convex, $g_i : \mathcal{R}^n \to \mathcal{R}, i = 1,...,m$, define the inequality constrains which are all convex functions, and finally $a_i^T x = b_i$, $i = 1,...,p$, define the equality constrains which are affine functions (and therefore also convex). In summary, a convex optimization problem consists in minimizing a convex function over a convex set. If the objective function is zero, then the problem is called a feasibility problem. Regarding the problem (6), some important definitions and terminologies are defined, which are discussed in the following.

Definition 7. *A point $x \in \mathcal{R}^n$ is called feasible if it satisfies all equality and inequality constraints. The set of all feasible points is called the feasible set. Accordingly, the problem* (6) *is called feasible if the feasible set is not empty.*

Definition 8. *A point $x^* \in \mathcal{R}^n$ is called the optimal point if it is feasible and minimizes the objective function f. An optimal point is called global if it is optimal in the entire feasible set and it is called local if it is optimal in a feasible neighbor of x^*.*

Definition 9. *The optimal value, p^*, of the problem* (6) *is defined as*

$$p^* = \min\{f(x) \,|\, g_i(x) \leq 0, i = 1,...,m, a_i^T x = b_i, i = 1,...,p\}.$$

Proposition 3. *For any convex problem in the form of* (6) *every local optimal point is also global.*

Now that the general form of convex optimization problems are discussed, the duality theory for this problem can be introduced in the next section.

2.2.3 Lagrangian Duality and KKT Conditions

Consider the problem (6). The basic idea in Lagrangian duality is to take the constraints into account by augmenting the objective function with a weighted sum of the constraint functions. Accordingly, the Lagrangian Function $L : \mathcal{R}^n \times \mathcal{R}^m \times \mathcal{R}^p \to \mathcal{R}$ is defined as

$$L(x,u,v) = f(x) + \sum_{i=1}^{m} u_i g_i(x) + \sum_{i=1}^{p} v_i(a_i^T x - b_i) \tag{7}$$

where, u_i (v_i) is the Lagrange multiplier associated with the i-th inequality (equality) constraint. The vectors, u and v are called dual variables associated with the problem (6).

Proposition 4. *For any feasible point $x \in \mathcal{R}^n$, the Lagrangian function induces a lower bound on the objective function if u_i is a positive number.*

In order to find the best lower bound, the Lagrangian dual function $q : \mathcal{R}^n \times \mathcal{R}^p \to \mathcal{R}$ is constructed as follows.

$$q(u,v) = \min_x L(x,u,v) \tag{8}$$

Proposition 5. *The dual function $q(u,v)$ is a concave function and hence, a global maximum point exists.*

Proposition 6. *For any $u \geq 0$ the dual function yields a lower bound on the optimal value p^* of the problem* (6)*, i.e.,*

$$q(u,v) \leq p^*. \tag{9}$$

The lower bound provided by the dual function is not necessarily the best dual bound. According to the concavity property of the dual function, the best lower bound is obtained by maximizing the dual function. This is another optimization problem, which is called the dual problem being defined as

$$\underset{u,v}{\text{maximize}} \ \ q(u,v) \tag{10}$$

$$\text{subject to} : u \geq 0 \tag{11}$$

The optimal value of the dual problem, d^*, is the best lower bound on p^*. In particular, we have

$$d^* \leq p^*.$$

The difference $p^* - d^*$ is called the duality gap. In general and for general nonlinear problems (not necessarily convex) the duality gap is usually nonzero. However for convex problems and under certain conditions, called Slater's condition, the duality gap is zero, .i.e.,

$$d^* = p^*.$$

The former and the latter case are called weak and strong duality, respectively.

By using the concept of dual problem and strong duality, the necessary and sufficient conditions (KKT conditions) of optimality for the convex problem can be stated in the following proposition.

Proposition 7. *Consider the convex problem* (6) *and assume the strong duality holds. Then x^* is the optimal point if the following conditions are satisfied:*

$$\begin{aligned} g_i(x^*) &\leq 0, \ i = 1,...,m \\ a_i^T x &= b_i, \ i = 1,...,p \\ u_i^* &\geq 0, \ i = 1,...,m \\ u_i^* g_i(x^*) &= 0, \ i = 1,...,m \\ \nabla_X L(x^*,u^*,v^*) &= 0 \end{aligned}$$

In summary, for any optimization problem with differentiable objective and constraint functions for which strong duality holds, any pair of primal and dual optimal points must satisfy the KKT conditions.

2.3 OPERATOR THEORY AND FIXED POINT ITERATIONS

Now, we review some fundamental results on operator theory, and we refer the interested reader to (BAUSCHKE; COMBETTES, 2017; RYU; BOYD, S., 2016) for an in-depth treatment.

2.3.1 Operator Theory and Fixed Point Algorithms

Definitions and basic properties are introduced below.

Definition 10. *An Operator (or mapping) $T : \mathcal{R}^n \to \mathcal{R}^n$ maps every point in $\mathcal{R}^n$ to a point $Tx \in \mathcal{R}^n$.*

Definition 11 (Fixed points of an operator)**.** *Let $T : \mathcal{R}^n \to \mathcal{R}^n$ be an operator on $\mathcal{R}^n$, then the set of fixed points of T is defined as follows:*

$$\text{fix}(T) = \left\{x \in \mathcal{R}^n : Tx = x\right\}. \tag{12}$$

Definition 12 (Lipschitz mapping)**.** *Let $T : \mathcal{R}^n \to \mathcal{R}^n$ be an operator on $\mathcal{R}^n$, then T is L-Lipschitz continuous with $L \geq 0$ if*

$$\|Tx - Ty\| \leq L\|x - y\| \tag{13}$$

holds for all $x \in \mathcal{R}^n$ and $y \in \mathcal{R}^n$. The operator T is non-expansive if $L = 1$, and contractive if $L < 1$ strictly.

Definition 13 (Averaged mapping)**.** *A mapping $S : \mathcal{X} \to \mathcal{Y}$ is α-averaged if there exists a non-expansive mapping $T : \mathcal{X} \to \mathcal{Y}$ and $\alpha \in (0,1)$ such that*

$$S = (1-\alpha)I + \alpha T \tag{14}$$

where I is the identity operator.

Definition 14 (Convex, Closed and Proper (CCP) functions)**.** *A convex function $f : \mathcal{R}^n \to \mathcal{R}$ is said to be proper if it never attains $-\infty$, while it is closed if it has bounded epigraphs, that is for any $a \in \mathbb{R}$ the set $\{x \mid f(\mathrm{x}) \leq a\}$ is closed.*

Definition 15 (Proximal and reflective Operator)**.** *Assume $f : \mathcal{R}^n \to \mathcal{R}$ is a CCP function and let $\rho \geq 0$. The proximal operator of f with penalty parameter ρ is defined as follows:*

$$\text{prox}_{\rho f}(\mathrm{y}) = \arg\min_{\mathrm{x}} \left\{ f(\mathrm{x}) + \frac{1}{2\rho}\|\mathrm{x} - \mathrm{y}\|_2^2 \right\}, \tag{15}$$

and the corresponding reflective operator is defined as:

$$\text{refl}_{\rho f} = 2\,\text{prox}_{\rho f} - I. \tag{16}$$

Remark 1. *The reflective and proximal operators are non-expansive. The proof of this result can be found in (BAUSCHKE; COMBETTES, 2017).*

Definition 16 (Weak and strong convergence. (BAUSCHKE; COMBETTES, 2017))**.** *Let $\{x(k)\}_{k \in \mathcal{N}}$ be a sequence of points in $\mathcal{R}^n$. The sequence is said to converge weakly to a point $x^* \in \mathcal{R}^n$ if*

$$\langle x(k), y\rangle \to \langle x^*, y\rangle \quad \forall y \in \mathcal{R}^n \tag{17}$$

for k that tends to infinity. Moreover, the sequence is said to converge strongly if

$$\|x(k) - x^*\| \to 0. \tag{18}$$

Strong and weak convergence in finite-dimensional spaces are equivalent.

where $\langle . \rangle$ is the dot product of two arbitrary vectors.

Definition 17 (Banach-Picard Fixed-Point Iteration (BAUSCHKE; COMBETTES, 2017))**.** *The Banach-Picard iteration to find the fixed points of an operator T is defined as*

$$x(k+1) = Tx(k). \tag{19}$$

If T is a contractive operator, then the sequence of points generated by this iteration converges to the fixed point of T.

Theorem 1 (Krasnosel'skiĭ-Mann iteration (BAUSCHKE; COMBETTES, 2017, Theorem 5.14))**.** *Let $\mathcal{D}$ be a nonempty closed and convex subset of $\mathcal{R}^n$. Assume $T : \mathcal{D} \to \mathcal{R}^n$ to be a non-expansive operator such that* $\mathrm{fix}(T) = \emptyset$*. Moreover, let α be a constant parameter in* $(0,1)$*. The Krasnosel'skiĭ-Mann (KM) fixed point iteration is defined as*

$$\mathrm{x}(k+1) = (1-\alpha)\mathrm{x}(k) + \alpha T\mathrm{x}(k) \tag{20}$$

with initial condition $x(0) \in \mathcal{D}$.
The sequence of points generated by the Krasnosel'skiĭ-Mann iteration is such that $\{T\mathrm{x}(k) - \mathrm{x}(k)\}$ converges strongly to 0*, i.e. $\|\mathrm{x}(k) - \mathrm{x}^*\| \to 0$.*

Operator theory can be exploited to design convex optimization algorithms. The underlying idea is to define a nonexpansive operator whose fixed points coincide with the solutions to the convex problem of interest; then, we can apply fixed point methods to solve this problem. In the following, we review some important definitions and results of convex optimization theory, which can be found in (RYU; BOYD, S., 2016).

Definition 18 (m-Strongly Convex Function)**.** *A CCP C^2 function $f : \mathcal{R}^n \to \mathcal{R}$ is m-strongly convex if*

$$f(\mathrm{x}) \geq f(\mathrm{y}) + \langle \nabla f(\mathrm{x}), \mathrm{x}-\mathrm{y} \rangle + \frac{m}{2}\|\mathrm{x}-\mathrm{y}\|^2, \forall x,y \tag{21}$$

where $m > 0$.

Definition 19 (L-Strongly smooth Function)**.** *A CCP C^2 function $f : \mathcal{R}^n \to \mathcal{R}$ is L-strongly smooth if*

$$f(\mathrm{x}) \leq f(\mathrm{y}) + \langle \nabla f(\mathrm{x}), \mathrm{x}-\mathrm{y} \rangle + \frac{L}{2}\|\mathrm{x}-\mathrm{y}\|^2 \ \forall x,y \tag{22}$$

where $L > 0$.

Proposition 8 ((GISELSSON; BOYD, S., 2014, Proposition 1))**.** *Let $f : \mathcal{R}^n \to \mathcal{R}$ be a CCP function. Then the following statements are equivalent.*

- *f is m-strongly convex*

- *f^* is $\frac{1}{m}$-smooth.*

Remark 2. *According to the Theorem 1 and KM iteration* (20)*, it is possible to observe that for any $\alpha \in$* (0,1)*, the KM algorithm converges. In a special case when the relaxation parameter $\alpha = 1$, the KM iteration is transformed to the Banach-Picard iteration, which does not necessarily converge unless some additional assumption is considered on the operator.*

2.4 MIXED-INTEGER LINEAR PROGRAMMING

This section provides an overview of Mixed-Integer Linear Programming (MILP) concepts that are utilized in this thesis. MILP is a class of optimization problems that involves the incorporation of integer variables. These variables are used to represent discrete quantities and distinct decisions. A MILP problem can be stated as follows:

$$\begin{aligned} \text{min} \quad & \mathbf{c}_1^T \mathbf{x} + \mathbf{c}_2^T \mathbf{y} \\ \text{subject to} \quad & \mathbf{A}_1 \mathbf{x} + \mathbf{A}_2 \mathbf{y} \leq \mathbf{b} \\ & \mathbf{x} \in \mathbb{R}^n, \mathbf{y} \in \mathbb{Z}^m \end{aligned} \tag{MILP}$$

Optimization problems with industrial applications that can be formulated as MILP problems are, e.g., allocation problems, packing problems, process planning, scheduling, and the famous traveling salesman problem. The main method for solving MILP problems is the branch-and-bound (BnB) algorithm. This algorithm solves a MILP problem by relaxing the integer constraints and dividing the search space. The resulting linear programming (LP) relaxation, obtained by dropping the integer constraints, is then solved. If the solution of the LP relaxation is not integer-valued, the algorithm divides the search space into subproblems, each with a restricted set of feasible solutions. This process is repeated recursively until a solution that satisfies all constraints is found, or until it is proven that no such solution exists. By dropping the integer restrictions, the resulting LP-relaxation is given by

$$\begin{aligned} \text{min} \quad & \mathbf{c}_1^T \mathbf{x} + \mathbf{c}_2^T \mathbf{y} \\ \text{subject to} \quad & \mathbf{A}_1 \mathbf{x} + \mathbf{A}_2 \mathbf{y} \leq \mathbf{b} \\ & \mathbf{x} \in \mathbb{R}^n, \mathbf{y} \in \mathbb{R}^m \end{aligned} \tag{R-MILP}$$

that can be easily solved. Nonetheless, the LP relaxation may not yield an integer feasible solution, meaning that at least one of the integer variables may have a fractional value. Nevertheless, the LP relaxation still provides a valid lower bound (LB) for the original problem, as it relaxes the integer constraints.

2.4.1 Branch and Bound Algorithm

Suppose $(\mathbf{x}^*, \mathbf{y}^*)$ is the minimizer of the relaxed problem (R-MILP), and that one of the integer variables y_i takes on a fractional value, *i.e.*, round$(y_i^*) \neq y_i^*$. To find an integer solution of the problem, the search space is divided into two parts such that the fractional solution is excluded from the search space but all feasible integer solutions are still included in the search. Such a division of the search space is achieved by generating two new sub-problems, and adding one of the constraints $y_i \leq \lfloor y_i^* \rfloor$ and $y_i \geq \lceil y_i^* \rceil$ to each sub-problem. After solving the new sub-problems and updating the lower bound, if one of the sub-problems returns a feasible integer solution, it is a valid upper bound for the MILP problem in that region of the search space. However, if the difference between the upper and lower bounds is still not within the desired tolerance, the search continues by dividing the sub-problems with a non-integer solution into two new sub-problems. This procedure of dividing the search space into sub-regions is known as branching, and the variable on which the division is done is called the branching variable.

Often there are several branching options, *e.g.*, if several of the integer variables take on fractional values. A branching variable is then chosen based on some criteria, *e.g.,* most infeasible branching, pseudo branching, or strong branching. The solution procedure is often represented as a tree where the initial LP-relaxation corresponds to the root node, and the sub-problems are represented by nodes branching out. In case the optimal solution of one of the sub-problems (nodes) exceeds the current upper bound, then the optimal solution cannot be within that specific part of the search space, and there is no need to further explore the node. When the search is stopped along one node, it is referred to as pruning the node. Some of the sub-problems may also become infeasible, and such nodes can also be pruned from the search tree. A node that may still contain the optimal solution is referred to as an open node, and several strategies for determining the order in which to explore open nodes have been proposed.

2.4.2 Cutting-Plane Algorithm

The BnB algorithm was not the initial approach suggested for solving MILP problems, as a cutting plane algorithm was proposed by Gomory in 1960. Gomory had already presented a similar cutting plane algorithm for pure integer problems in 1958. The basic concept is still to solve LP-relaxations, but instead of branching, the algorithm obtains cuts that eliminate the non-integer solution of the LP-relaxation.To show how such cuts can be generated, consider a problem with the following constraints in the integer variables $\mathbf{y}$

$$\mathbf{A}_2\mathbf{y} \leq \mathbf{b} \tag{23}$$

where $a_1, a_2, \ldots, a_n$ are the columns of $\mathbf{A}_2$ matrix. Here it is assumed that all the integer variables y are restricted to non-negative integers. The individual constraints of (23) can be combined into a single constraint according to

$$\lambda^T \mathbf{A}_2\mathbf{y} \leq \lambda^T \mathbf{b} \tag{24}$$

where λ is a non-negative scaling vector. Rounding down the coefficients on the left hand side, given by $\lambda^T a_i$, results in a valid relaxed constraint. Once the right-hand side contains only integer coefficients, then an integer solution can only result in an integer value on the right-hand side. The coefficient on the left-hand side can then be rounded down to an integer value, thus strengthening the constraint and resulting in the following Chvátal-Gomory cut

$$\left[\lfloor\lambda^T a_1\rfloor \dots \lfloor\lambda^T a_n\rfloor\right] \leq \lfloor\lambda^T \mathbf{b}\rfloor \tag{25}$$

Chvátal-Gomory cuts will not cut off any feasible integer solution of the problem. However, they can cut off non-integer regions of the feasible set and strengthen the LP-relaxation. There is no unique way of constructing Chvátal- Gomory cuts, and by choosing different scaling vectors λ, an infinite number of cuts can be obtained. Gomory's cutting plane algorithm for integer problems uses an iterative procedure for generating cuts according to (25) to iteratively improve the LP-relaxation, and to obtain an integer solution.

Over the years, cutting planes have received considerable attention in the context of MILP problems, and several types of cuts have been proposed, including mixed-integer Gomory cuts, disjunctive cuts, lift-and-project cuts, intersection cuts, split cuts, and cover cuts. The primary objective of using these cuts is to obtain a tighter LP-relaxation by utilizing various properties of the MILP problems. However, using cutting planes alone to solve MILP problems is often not efficient because of the potential for an enormous number of cutting planes being required, and the cutting planes may become almost parallel, leading to issues of LP sub-problems degeneracy. Nowadays, most MILP solvers utilize the branch-and-cut technique, which employs cutting planes to strengthen the LP-relaxations in the branch and bound algorithm.

MILP problems are considered NP-hard, making them difficult to solve. However, the field of MILP has made significant progress in developing general-purpose solvers for these problems. There are several solvers available for MILP problems, such as CBC, CPLEX, Gurobi, SCIP, and XPRESS. Benchmark tests in (KOCH et al., 2011) have shown that these solvers have even been able to solve some MILP problems with more than 100,000 discrete variables. Solving MILP sub-problems is one of the key components in several methods for convex MINLP, and the ability to efficiently solve MILP problems is crucial for such methods. Convex MINLP methods utilizing MILP relaxation will be described in detail later on.

2.5 MIXED-INTEGER CONVEX PROGRAMMING

This section presents the basic elements of MINLP methods and algorithms. For more details see (GROSSMANN, I., 2002; KRONQVIST et al., 2019).

The most basic form of an MINLP problem when represented in algebraic form is as follows:

$$\min_{\mathrm{x,y}} f(\mathrm{x,y}) \tag{26)a}$$

$$\text{s.t. } g_j(\mathrm{x,y}) \leq 0, \quad j \in J \tag{26)b}$$

$$\mathrm{x} \in \mathcal{X},\ \mathrm{y} \in \mathcal{Y} \tag{26)c}$$

where $f : \mathcal{R}^n \rightarrow \mathcal{R}$ and $g_j : \mathcal{R}^n \rightarrow \mathcal{R}, j \in J$, are convex, differentiable functions, J is the index set of inequalities, and x and y are continuous and discrete variables, respectively. The set $\mathcal{X}$ is commonly assumed to be a convex and compact set, and the discrete set $\mathcal{Y}$ corresponds to a polyhedral set of integer points, which in most applications is restricted to binary values. In most applications of interest the objective and constraint functions f and g are linear in y: $f(\mathrm{x,y}) = c^T\mathrm{y} + r(\mathrm{x})$ and $g_j(\mathrm{x,y}) = B_j\mathrm{y} + h_j(\mathrm{x})$.

Methods that addressed the solution of problem (26) include the branch-and-bound method (BnB), Generalized Bender's Decomposition (GBD), Outer Approximation (OA), Extended Cutting Plane (ECP), LP/NLP BnB, and Extended Supporting Hyperplane (ESH). Since the algorithms developed in this thesis are derived based on the OA and LP/NLP BnB methods, they are briefly discussed here.

2.5.1 Outer Approximation

The OA method was first presented by (DURAN; GROSSMANN, Ignacio E, 1986) and is a decomposition technique, which obtains the optimal solution of the original problem by solving a sequence of Mixed Integer Linear Programming (MILP) and Nonlinear Programming (NLP) subproblems. The OA constructs an iteratively improving polyhedral outer approximation of the nonlinear functions. However, OA only uses the polyhedral approximation for choosing the integer combination y, while the corresponding continuous variables x are chosen by solving a convex NLP subproblem. In each iteration, the polyhedral outer approximation is used to construct problem (MILP-k), referred to as MILP master problem. A new integer combination y^k is then obtained by solving problem (MILP-k), which is defined as:

$$\min_{\mathrm{x,y},\alpha} \alpha \tag{27)a}$$

$$\text{s.t. } \alpha \geq f(\mathrm{x}^k,\mathrm{y}^k) + \nabla f(\mathrm{x}^k,\mathrm{y}^k)^T \begin{bmatrix} \mathrm{x}-\mathrm{x}^k \\ \mathrm{y}-\mathrm{y}^k \end{bmatrix} \tag{27)b}$$

$$g_j(\mathrm{x}^k,\mathrm{y}^k) + \nabla g_j(\mathrm{x}^k,\mathrm{y}^k)^T \begin{bmatrix} \mathrm{x}-\mathrm{x}^k \\ \mathrm{y}-\mathrm{y}^k \end{bmatrix} \leq 0,\ j \in J^k,\ k = 1,...,K \tag{27)c}$$

$$\mathrm{x} \in \mathcal{X},\ \mathrm{y} \in \mathcal{Y} \tag{27)d}$$

where $J^k \subseteq J$ and K is the number of feasible points generated by the NLP subproblem. Once the integer combination $\mathbf{y}^k$ is obtained, the following NLP subproblem is formed

$$\min_{\mathbf{x}} f(\mathbf{x},\mathbf{y}^k) \tag{28)a}$$

$$\text{s.t. } g_j(\mathbf{x},\mathbf{y}^k) \leq 0, \quad j \in J \tag{28)b}$$

$$\mathbf{x} \in \mathcal{X} \tag{28)c}$$

If problem (28) is feasible, a valid upper bound can be obtained from the solution $(\mathbf{x}^k,\mathbf{y}^k)$, and the solution is used to improve the polyhedral approximation in the master's problem.

Problem (28) may also be infeasible at some iteration. If $\mathbf{y}^k$ is an feasible integer combination, the corresponding continuous variable can be obtained by solving the following convex subproblem

$$\min_{\mathbf{x}} \quad u \tag{29)a}$$

$$\text{s.t. } g_j(\mathbf{x},\mathbf{y}^k) \leq u, \quad j \in J \tag{29)b}$$

$$\mathbf{x} \in \mathcal{X} \tag{29)c}$$

which minimizes the constraint violation with respect to the ℓ_∞-norm. The solution to problem (29) does not provide a lower bound. The OA algorithm is usually initiated by solving a continuous relaxation of the MINLP problem, giving an initial lower bound and a solution that can be used to construct polyhedral approximations. It is also possible to use integer cuts to exclude specific integer combinations. Solving the MILP master problems (27) provides a lower bound on the optimum, and the procedure is repeated until the upper bound and lower bound is within a given tolerance.

2.5.2 LP/NLP-based branch-and-bound

When applying the OA algorithm to a convex MINLP problem, the majority of the total solution time is, usually, spent on solving the MILP master problems. A new approach to avoid solving multiple MILP sub-problems was presented by (QUESADA; GROSSMANN, Ignacio E, 1992). The method, known as LP/NLP-based branch-and-bound (LP/NLP-BnB), integrates OA within a branch-and-bound framework. Compared to OA, only one branch-and-bound tree is generated by dynamically updating the MILP master problem. The search is initialized by solving an integer relaxation of the MINLP problem, and the solution is used to construct an initial polyhedral outer approximation. The integer relaxation also provides a valid lower bound on the optimal objective value of the MINLP problem. The polyhedral outer approximation is used to construct the initial MILP master problem, which will be solved by a branch-and-bound procedure.

An LP relaxation is solved in each node of the branch-and-bound tree, and the search is stopped once an integer solution is obtained in one of the nodes. The integer solution is treated as normal in OA, by solving an NLP problem with the integer variables fixed. If it results in a feasible NLP problem, then it provides a valid upper bound, and new linearizations can be generated. If the NLP problem is infeasible, it is possible to add an integer cut to exclude the solution or solve the feasibility problem and generate new linearizations. The new linear constraints are then added to all open nodes in the branch-and-bound tree, and the LP relaxation is resolved for the node which resulted in the integer combination. The search continues with the improved polyhedral outer approximation, and the BnB procedure continues from the existing search tree.

As normally done in BnB, nodes can be pruned off in case the optimum of the LP relaxation exceeds the upper bound. However, the search cannot be stopped once an integer solution is obtained at a node; the search must continue until the LP relaxation results in a feasible integer solution or until the node can be pruned off. By constructing a single branch-and-bound tree, LP/NLP-BnB may end up exploring fewer nodes than the total number of nodes explored in the multiple branch-and-bound trees in normal OA. The LP/NLP-BnB can also be combined with cut-generating procedures for tightening the integer relaxation. Generally, LP/NLP-BnB is considered one of the most efficient techniques for convex MINLP, which is also supported by a recent benchmark test of convex MINLP solvers in (KRONQVIST et al., 2019).

2.6 CONCLUSION

This chapter briefly reviewed some of the important topics that are needed throughout this thesis, such as distributed computing models, operator theory, convex optimization, mixed-integer linear and nonlinear optimization, and algorithms. With the information presented in this chapter, we are now ready to begin with the main contributions of this thesis.

3 DISTRIBUTED SPARSE GRADIENT TRACKING ALGORITHM

3.1 INTRODUCTION

This chapter presents the Distributed Sparse Gradient Tracking (DiSGT) algorithm, which addresses the SCO problem with a sparsity constraint and semi-continuous variables similar to the cost-coupled setup (1) (OLAMA et al., 2023). To develop this algorithm, we drew inspiration from the penalty decomposition method introduced in (BAI, Y.; LIANG; YANG, Z., 2016a; LU; ZHANG, Yong, 2013) and the gradient tracking algorithm described in (NOTARSTEFANO; NOTARNICOLA; CAMISA, et al., 2019). The resulting `DiSGT` is a fully-distributed algorithm that solves the SCO problem through local communication and computation. Essentially, the `DiSGT` approach divides the problem into two subproblems by introducing appropriate auxiliary variables and applies the *Block Coordinate Descent* (BCD) method (LU; ZHANG, Yong, 2013; BAI, Y.; LIANG; YANG, Z., 2016a) in tandem with the gradient tracking algorithm to produce a novel fully-distributed scheme. Finally, we confirm the effectiveness of our distributed algorithm by performing some numerical simulations in a sparse logistic regression framework with synthetic data sets and a real-world problem arising in the context of electrical smart grids. It is worth noting that the DiGST algorithm is the initial method proposed in this thesis for solving SCO problems by directly addressing the sparsity constraint. However, DiGST is unable to handle linear and nonlinear constraints that may be present in the problem formulation. In the subsequent chapter, we present the RH-ADMM algorithm, which can efficiently solve constrained large-scale convex optimization problems and be used to tackle the continuous portion of SCO problems. The RH-ADMM algorithm is then integrated into the MINLP algorithms presented in Chapter 5.

The chapter is structured into the following sections: Section 3.2 introduces the sparse optimization problem that is being considered in this work. Section 3.3 outlines the development of the DiGST algorithm, while Section 3.4 presents the results of numerical experiments conducted on synthetic and real datasets to evaluate its effectiveness. Finally, Section 3.5 provides some concluding remarks and summarizes the main conclusions of the chapter.

3.2 PROBLEM DESCRIPTION

We consider a network of N agents aiming to cooperatively solve the sparse convex optimization problem with semi-continuous variables described by

$$\begin{aligned} &\min_{x \in \mathbb{R}^n} \sum_{i=1}^{N} f_i(\mathbf{x}) \\ &\text{subject to } x_l \in \{0\} \cup [a^l_{\mathtt{min}}, a^l_{\mathtt{max}}], l \in \{1, \dots, n\} \\ &\qquad \|\mathbf{x}\|_0 \leq \kappa, \end{aligned} \tag{30}$$

where $f_i : \mathbb{R}^n \to \mathbb{R}$ is the local convex objective function of agent i, and $\kappa \in \mathbb{N}$ is the maximum number of non-zero elements of the decision variable $\mathbf{x} \in \mathbb{R}^n$ with $\kappa < n$. Moreover, a variable x_l is referred to as a *semi-continuous* variable. According to the desired distributed paradigm, each agent i can only access its local objective function f_i and exchange data with a subset of the entire agents set according to the communication graph associated with the network. Indeed, we assume that the agents of the network communicate according to an *undirected* graph $\mathcal{G} = (\{1, \dots, N\}, \mathcal{E}, \mathcal{A})$, where $\{1, \dots, N\}$ is the set of agents, $\mathcal{E} \subseteq \mathcal{V} \times \mathcal{V}$ is the set of edges, and $\mathcal{A} \in \mathbb{R}^{N \times N}$ is the weighted adjacency matrix. Each agent $i \in \{1, \dots, N\}$ can receive data from agent j only if $(j,i) \in \mathcal{E}$ and, in this case, agent j is said to be a neighbor of agent i. Moreover, the entries of the weighted adjacency matrix A match the graph structure, namely the (i,j) entry satisfies $a_{ij} > 0$ if $(i,j) \in \mathcal{E}$, and $a_{ij} = 0$ otherwise. The symbol $\mathcal{N}_i$ denotes the set of neighbors of agent i and is defined as $\mathcal{N}_i := \{j \in \mathcal{V} \mid (i,j) \in \mathcal{E}\}$. The next section is devoted to designing an iterative distributed algorithm to address problem (30).

3.3 DISGT DEVELOPMENT

We start by introducing an auxiliary variable $\mathbf{y} \in \mathbb{R}^n$ which allows us to equivalently rewrite problem (30) as

$$\begin{aligned} &\min_{\mathbf{x} \in \mathbb{R}^n} \sum_{i=1}^{N} f_i(\mathbf{x}) \\ &\text{subject to } \mathbf{x} = \mathbf{y} \\ &\qquad \mathbf{y} \in \mathcal{C} \end{aligned} \tag{31}$$

where

$$\mathcal{C} = \{\mathbf{y} \in \mathbb{R}^n : y_l \in \{0\} \cup [a^l_{\mathtt{min}}, a^l_{\mathtt{max}}], \|\mathbf{y}\|_0 \leq \kappa\}.$$

With this formulation at hand and given an arbitrary constant $\rho > 0$, we introduce an Augmented Lagrangian function $L_\rho : \mathbb{R}^n \times \mathbb{R}^n \times \mathbb{R}^n \to \mathbb{R}$ associated to problem (31) which is defined as

$$L_\rho(\mathbf{x}, \mathbf{y}, \lambda) = \sum_{i=1}^{N} f_i(\mathbf{x}) + \lambda^\top(\mathbf{x} - \mathbf{y}) + \frac{\rho}{2} \|\mathbf{x} - \mathbf{y}\|_2^2 . \tag{32}$$

The next section is devoted to describing the BCD method (BAI, Y.; LIANG; YANG, Z., 2016a), namely, a centralized algorithm to find the saddle-points of L_ρ (32).

3.3.1 Block Coordinate Descent

In order to find a saddle-point of L_ρ satisfying the constraint $\mathrm{y} \in \mathcal{C}$, we could apply the block coordinate descent method, which reads as

$$\mathrm{x}^{t+1} = \arg\min_{\mathrm{x}} L_\rho(\mathrm{x}, \mathrm{y}^t, \lambda^t) \tag{33)a}$$

$$\mathrm{y}^{t+1} = \arg\min_{\mathrm{y}\in\mathcal{C}} L_\rho(\mathrm{x}_i^{t+1}, \mathrm{y}, \lambda^t) \tag{33)b}$$

$$\lambda^{t+1} = \lambda^t + \rho(\mathrm{x}^{t+1} - \mathrm{y}^{t+1}) \tag{33)c}$$

However, iterations (33) cannot be implemented in a distributed fashion. Indeed, the update (33)a requires the knowledge of all the local functions f_i in each agent of the network. In order to overcome this issue, we resort to the so-called gradient tracking algorithm to replace (33)a. Specifically, at each iteration, $t \in \mathbb{N}$, each agent $i \in \{1, \ldots, N\}$ locally maintains an estimate $(\mathrm{x}_i^t, \mathrm{y}_i^t, \lambda_i^t) \in \mathbb{R}^n \times \mathbb{R}^n \times \mathbb{R}^n$ about a saddle-point of L_ρ satisfying the desired constraint. Indeed, for all $t \in \mathbb{N}$, the update (33)a consists in solving the i-th optimization problem described by

$$\min_{\mathrm{x}\in\mathbb{R}^n} \sum_{j=1}^{N} g_j\left(\mathrm{x}, \mathrm{y}_i^t, \lambda_i^t\right), \tag{34}$$

where the objective function $g_j : \mathbb{R}^n \times \mathbb{R}^n \times \mathbb{R}^n \to \mathbb{R}$ reads as

$$g_j(\mathrm{x}, \mathrm{y}_i^t, \lambda_i^t) = f_j(\mathrm{x}) + \frac{1}{N}\left((\lambda_i^t)^\top(\mathrm{x} - \mathrm{y}_i^t) + \frac{\rho}{2}\left\|(\mathrm{x} - \mathrm{y}_i^t)\right\|_2^2)\right),$$

for all $j \in \{1, \ldots, N\}$. By taking advantage of the suitable structure of problem (34), we replace the step (33)a with an inner implementation of the well-known gradient tracking algorithm, i.e., a popular distributed mechanism to effectively solve consensus optimization problems (SHI et al., 2015; NEDIĆ; OLSHEVSKY; SHI, 2017; DI LORENZO; SCUTARI, 2016; XI; XIN; KHAN, 2017; SCUTARI; SUN, Y., 2019; CARNEVALE et al., 2023; CARNEVALE; NOTARSTEFANO, 2022; DANESHMAND; SCUTARI; KUNGURTSEV, 2020; CARNEVALE et al., 2022). Specifically, at each iteration $t \in \mathbb{N}$ and for each agent $i \in \{1, \ldots, N\}$, we introduce an inner iteration index $k \in \mathbb{N}$ and an estimate $\mathrm{x}_i^{k,t} \in \mathbb{R}^n$ of a solution $\mathrm{x}^*(\mathrm{y}_i^t, \lambda_i^t)$ of problem (34). Moreover, as customary in the implementation of the gradient tracking algorithm, we also introduce an auxiliary variable $\mathrm{s}_i^{k,t} \in \mathbb{R}^n$ called tracker. Each tracker $\mathrm{s}_i^{k,t}$ exploits a perturbed consensus dynamics to locally reconstruct the unavailable global gradient $\sum_{j=1}^N \nabla g_j(\mathrm{x}_i^{k,t}, \mathrm{y}_i^t, \lambda_i^t)$. In turn, such a tracker is used to update the local solution estimate $\mathrm{x}_i^{k,t}$ mimicking the well-known gradient descent method plus an averaging step, which aims to force consensus among the solution estimates $\mathrm{x}_i^{k,t}$. Hence, the whole update from the perspective of agent i reads as

$$\mathrm{x}_i^{k+1,t} = \sum_{j\in\mathcal{N}_i} a_{ij}\mathrm{x}_j^{k,t} - \gamma \mathrm{s}_i^{k,t} \tag{35a}$$

$$\mathrm{s}_i^{k+1,t} = \sum_{j\in\mathcal{N}_i} a_{ij}\mathrm{s}_j^{k,t} + \nabla g_i(\mathrm{x}_i^{k+1,t},\mathrm{y}_i^t,\lambda_i^t) - \nabla g_i(\mathrm{x}_i^{k,t},\mathrm{y}_i^t,\lambda_i^t), \tag{35b}$$

where $\gamma > 0$ is a parameter called step-size and the weights a_{ij} are the entries of the weighted adjacency matrix $\mathcal{A}$ associated to the graph $\mathcal{G}$. As a necessary condition for the algorithm effectiveness, we remark that these trackers must be initialized according to $s_i^{0,t} = \nabla g_i(\mathrm{x}_i^{0,t},\bar{\mathrm{y}}^t,\bar{\lambda}^t)$ (SHI et al., 2015; NEDIĆ; OLSHEVSKY; SHI, 2017; DI LORENZO; SCUTARI, 2016; XI; XIN; KHAN, 2017; SCUTARI; SUN, Y., 2019; CARNEVALE et al., 2023; CARNEVALE; NOTARSTEFANO, 2022; DANESHMAND; SCUTARI; KUNGURTSEV, 2020; CARNEVALE et al., 2022).

Remark 3. *We remark that if the consensus among all the variables y_i^t and λ_i^t of the network is enforced, i.e., it holds $\mathrm{y}_i^t = \bar{\mathrm{y}}^t$ and $\lambda_i^t = \bar{\lambda}^t$ for all $i \in \{1,\dots,N\}$ and some $\bar{\mathrm{y}}^t, \bar{\lambda}^t \in \mathbb{R}^n$, then problem* (34) *turns out to be the same in each agent i of the network. Specifically, it would read as*

$$\min_{\mathrm{x}\in\mathbb{R}^n} \sum_{i=1}^{N} g_i\left(\mathrm{x},\bar{\mathrm{y}}^t,\bar{\lambda}^t\right). \tag{36}$$

In turn, in (SHI et al., 2015; NEDIĆ; OLSHEVSKY; SHI, 2017; DI LORENZO; SCUTARI, 2016; XI; XIN; KHAN, 2017; SCUTARI; SUN, Y., 2019; CARNEVALE et al., 2023; CARNEVALE; NOTARSTEFANO, 2022; DANESHMAND; SCUTARI; KUNGURTSEV, 2020; CARNEVALE et al., 2022) it is guaranteed that if regularity assumptions about the local objective functions f_i (e.g., convexity) and the graph $\mathcal{G}$ (connectivity, $\mathcal{A}$ doubly stochastic) are enforced, then there exists $\bar{\gamma} > 0$ such that

$$\lim_{k\to\infty} \left\| \mathrm{x}_i^{k,t} - \mathrm{x}^*(\bar{\mathrm{y}}^t,\bar{\lambda}^t) \right\| = 0,$$

for all $i \in \{1,\dots,N\}$, where $\mathrm{x}^*(\bar{\mathrm{y}}^t,\bar{\lambda}^t) \in \mathbb{R}^n$ denotes a solution of the consensus optimization problem (36)*.*

As soon as the inner algorithm (35) converges to a steady-state $\mathrm{col}(\mathrm{x}_1^{\mathrm{eq},t+1}, \mathrm{s}_1^{\mathrm{eq},t+1}, \dots, \mathrm{x}_N^{\mathrm{eq},t+1}, \mathrm{s}_N^{\mathrm{eq},t+1}) \in \mathbb{R}^{2Nn}$ [1] of (31), we set $\mathrm{x}_i^{t+1} = \mathrm{x}_i^{\mathrm{eq},t}$ for all $i \in \{1,\dots,N\}$ and use it to update the variables y_i^t and λ_i^t as follows

$$\mathrm{y}_i^{t+1} = \arg\min_{\mathrm{y}\in\mathcal{C}} L_\rho(\mathrm{x}_i^{t+1},\mathrm{y},\lambda_i^t) \tag{37a}$$

$$\lambda_i^{t+1} = \lambda_i^t + \rho(\mathrm{x}_i^{t+1} - \mathrm{y}_i^{t+1}). \tag{37b}$$

[1] Given N vectors $\mathrm{x}_1,\dots,\mathrm{x}_N$, their vertical stack is denoted by $\mathrm{col}(\mathrm{x}_1,\dots,\mathrm{x}_N)$.

It is worth noting that (37) can be implemented in a fully-distributed fashion because it only relies on local information.

Remark 4. *In general, the steady-state configuration* $\text{col}(\mathrm{x}_1^{eq,t+1}, \dots, \mathrm{x}_N^{eq,t+1})$ *does not necessarily coincide with a consensus configuration because of the possible different pairs* $(\mathrm{y}_1^t, \lambda_1^t), \dots, (\mathrm{y}_N^t, \lambda_N^t)$ *leading to N different problems* (34)*. However, a consensus configuration* $\mathrm{y}_i^t = \bar{\mathrm{y}}$*,* $\lambda_i^t = \bar{\lambda}$ *for all* $i \in \{1, \dots, N\}$ *and some* $\bar{\mathrm{y}}^t, \bar{\lambda}^t \in \mathbb{R}^n$ *leads to N equivalent problems in the form of* (36)*. Then, as argued in Remark 3, the inner algorithm* (35) *provides, for all* $i \in \{1, \dots, N\}$*, a consensual solution* $\mathrm{x}_i^{t+1} = \mathrm{x}^*(\bar{\mathrm{y}}^t, \bar{\lambda}^t)$ *of the consensus optimization problem* (36)*. In turn, by inspecting the updates* (37)*, one may verify that consensus among the updated variables* $\mathrm{x}_i^{t+1} = \mathrm{x}^*(\bar{\mathrm{y}}^t, \bar{\lambda}^t)$ *allows to preserve consensus among the variables* y_i^{t+1} *and* λ_i^{t+1}*. Therefore, for the arguments above, we conclude that a suitable initialization* $\mathrm{y}_i^0 = \bar{\mathrm{y}}^0$*,* $\lambda_i^0 = \bar{\lambda}^0$ *for all* $i \in \{1, \dots, N\}$ *and some* $\bar{\mathrm{y}}^0, \bar{\lambda}^0 \in \mathbb{R}^n$ *allows us to maintain the consensus conditions among the local variables for the whole algorithm execution. In turn, such a suitable initialization allows the mechanisms* (35) *and* (37) *to correctly mimic the centralized algorithm* (33)*.*

Among the possibilities, we choose the consensus initialization $\mathrm{y}_i^0 = \lambda_i^0 = 0$ for all $i \in \{1, \dots, N\}$ since it can be implemented in a fully distributed manner. Moreover, by using a constant $\varepsilon > 0$, we include a mechanism to check that convergence of the inner algorithm (35) has occurred.

Algorithm 1 Basic steps of `DiSGT` Algorithm

```
1: initialization $\mathrm{y}_i^0 = 0$, $\lambda_i^0 = 0$,
2: for $t = 0, 1, \dots$ do
3:     initialization $\mathrm{x}_i^{0,t} \in \mathbb{R}^n$, $\mathrm{s}_i^{0,t} = \nabla g_i(\mathrm{x}_i^{0,t}, \mathrm{y}_i^t, \lambda_i^t)$, $k \leftarrow 0$
4:     while $\left\|\mathrm{s}_i^{k,t}\right\| > \varepsilon$ do
5:
           $$\mathrm{x}_i^{k+1,t} = \sum_{j \in \mathcal{N}_i} a_{ij} \mathrm{x}_j^{k,t} - \gamma \mathrm{s}_i^{k,t} \quad (38)a$$
           $$\mathrm{s}_i^{k+1,t} = \sum_{j \in \mathcal{N}_i} a_{ij} \mathrm{s}_j^{k,t} + \nabla g_i(\mathrm{x}_i^{k+1,t}, \mathrm{y}_i^t, \lambda_i^t) - \nabla g_i(\mathrm{x}_i^{k,t}, \mathrm{y}_i^t, \lambda_i^t), \quad (38)b$$
6:         $k \leftarrow k + 1$
7:     Set:
           $$\mathrm{x}_i^{t+1} = \mathrm{x}_i^{k+1,t} \quad (39)a$$
           $$\mathrm{y}_i^{t+1} = \arg\min_{\mathrm{y} \in \mathcal{C}} L_\rho(\mathrm{x}_i^{t+1}, \mathrm{y}, \lambda_i^t) \quad (39)b$$
           $$\lambda_i^{t+1} = \lambda_i^t + \rho(\mathrm{x}_i^{t+1} - \mathrm{y}_i^{t+1}). \quad (39)c$$
```

3.4 COMPUTATIONAL EXPERIMENTS

In this section, we perform various numerical simulations to show the effectiveness of Algorithm 1. First, we focus on a sparse classification problem with synthetic data sets. Then, we consider a real-world data set to predict the stability properties of electrical smart grids. All the numerical simulations are performed over a network of agents communicating according to a communication graph that we randomly generate setting the edge probability $p = 20\%$. We performed all the experiments on a Linux machine with an Intel Core i5 2.50 GHz processor, with four physical cores and 16 GB of RAM. Algorithm 1 is entirely implemented in the `Python 3.10` programming language and relies on `Numpy` to carry out linear algebra operations.

3.4.1 Sparse Classification with Synthetic Data Sets

Sparse classification is a central problem in machine learning as it leads to more interpretable models. Given the i-th node and local data $\{(\mathbf{x}_{(ij)}, y_{ij})\}_{j=1,\dots,m}$ with $y_{ij} \in \{-1, 1\}$ and $\mathbf{x}_{(ij)} \in \mathbb{R}^n$ distributed over a computational graph with N nodes, the goal of distributed sparse classification is to compute an estimator $(\mathbf{w}, b) \in \mathbb{R}^n \times \mathbb{R}$ which minimizes a sum of local empirical loss functions $\ell_1, \dots, \ell_N$ satisfying a cardinality constraint about the number of nonzero entries. Such a problem can be formalized as

$$\begin{aligned} &\min_{\mathbf{w}\in\mathbb{R}^n, b\in\mathbb{R}} \sum_{i=1}^{N}\sum_{j=1}^{m} \ell_i\left(y_{ij}, \mathbf{w}^\top \mathbf{x}_{(ij)} + b\right) \\ &\text{subject to } \|\mathbf{w}\|_0 \le \kappa, \end{aligned} \tag{40}$$

with $\kappa \in \mathcal{N}$. The specific choice about the loss functions ℓ_i leads to different classification models. In the next, we will see three different instances of problem (40). The Distributed Sparse Logistic Regression (DSLR) model can be obtained by solving the optimization problem defined as

$$\begin{aligned} &\min_{\mathbf{w}\in\mathbb{R}^n, b\in\mathbb{R}} \sum_{i=1}^{N}\sum_{j=1}^{m} \log\left(1 + \exp^{-y_{ij}(\mathbf{w}^\top \mathbf{x}_{(ij)} + b)}\right) \\ &\text{subject to } \|\mathbf{w}\|_0 \le \kappa. \end{aligned} \tag{41}$$

Similarly, Distributed Sparse 1-norm Support Vector Machine (DSSVM-1) is defined as

$$\begin{aligned} &\min_{\mathbf{w}\in\mathbb{R}^n, b\in\mathbb{R}} \sum_{i=1}^{N}\sum_{j=1}^{m} \max\left(0, 1 - y_{ij}(\mathbf{w}^\top \mathbf{x}_{(ij)} + b)\right) \\ &\text{subject to } \|\mathbf{w}\|_0 \le \kappa. \end{aligned} \tag{42}$$

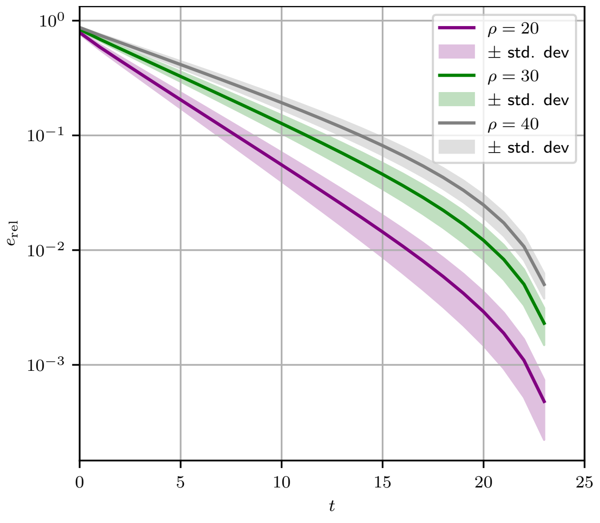


Figure 2 – Mean of the relative error and 1-standard deviation band obtained with 50 Monte Carlo trials.

Finally, Distributed Sparse 2-norm Support Vector Machine (DSSVM-2) is defined as

$$\begin{aligned} \min_{\mathbf{w}\in\mathbb{R}^n, b\in\mathbb{R}} & \sum_{i=1}^{N}\sum_{j=1}^{m_i} \max\left(0, 1 - y_{ij}(\mathbf{w}^\top \mathbf{x}_{(ij)} + b)\right)^2 \\ \text{subject to } & \|\mathbf{w}\|_0 \leq \kappa. \end{aligned} \tag{43}$$

In the next, we will provide numerical results obtained by testing Algorithm 1 over instances of DSLR problem (41). In detail, we consider a network of $N = 30$ agents and, for each agent, we generate random local data sets with zero mean and unit ℓ_2 norm for each column. The response vector y_{ij} is generated according to the logistic function as follows

$$y_{ij} = \text{round}\left(\frac{1}{1 + \exp^{-(\mathbf{w}^T \mathbf{x}_{(ij)} + b)}}\right). \tag{44}$$

In the first scenario, we evaluate the convergence performance of the proposed algorithm under different values of penalty parameter ρ. The main metric for the evaluation is the relative error e_{rel}^t in the logarithmic scale defined as

$$e_{\mathrm{rel}}^t = \log \frac{\left\|\mathbf{x}^t - \mathbf{x}^*\right\|}{\|\mathbf{x}^*\|},$$

where $\mathbf{x}^*$ is the optimal solution computed through the centralized algorithm (33). Fig. 2 depicts the mean relative error for different values of the penalty parameter ρ which is computed by performing 50 Monte Carlo trials that differ in the problem parameters and datasets. In this scenario, we set $n = 10$ and $\kappa = 3$. Fig. 2 shows that the penalty parameter considerably affects the convergence rate of the algorithm. In contrast to the convex case, where an ℓ_0 norm is approximated by ℓ_1 norm, a sufficiently large ρ is needed for the algorithm to converge. In the scenario depicted in Fig. 2, the best

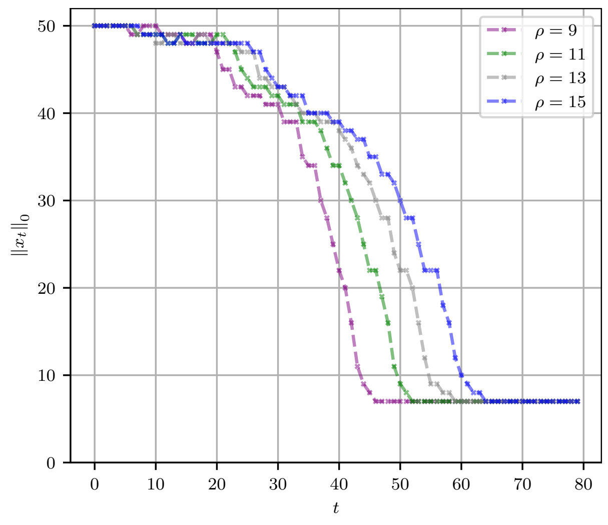


Figure 3 – Number of nonzero elements produced by Algorithm 1 with different penalty parameters.

convergence result is given by $\rho = 20$. Fig. 3 illustrates the sparsity of x^t produced by the algorithm for different values of ρ and for a problem with $n = 50$ and $\kappa = 7$. As the figure depicts, the desired convergence of $\left\|\mathrm{x}^t\right\|_0$ to $\kappa = 7$ is achieved. In this experiment, $\rho = 9$ leads to the best convergence results.

3.4.2 Smart Grid Stability Prediction

In this section, we test the proposed method in a real-world situation, namely Algorithm 1 is used to solve a DSLR problem based on a smart grid stability data set. Based on its features, the main goal is to predict whether or not an electrical smart grid is stable or unstable and, thus, such a binary classification problem can be suitably formulated into the DSLR framework.

Data is collected from the *UCI Machine Learning Repository* named as *Electrical Grid Stability Simulated Data* and consists of 10000 observation points and 13 features. The features consist of the reaction time of participants, elasticity of nominal power consumed, and so on. For more details about the data set see (ARZAMASOV; BÖHM; JOCHEM, 2018). As mentioned, the response vector consists of two classes, namely *stable* and *unstable*. Before training the DSLR model we check the class balance between stable and unstable classes. In particular, a balanced data set is a type of data set where the distribution of labels across the data set is balanced, *i.e.* the distribution is not biased or skewed. An imbalanced class may cause problems in the prediction accuracy of the DSLR model because the majority class can dominate the minority class and the classifier may only learn one concept instead of two distinct concepts. To visualize the balance class, we count and plot the number of samples in each class of the data set. The balance class is depicted in Figure 4. As the figure shows, the size of the unstable category class is about 1.7 times the size of the stable class, which

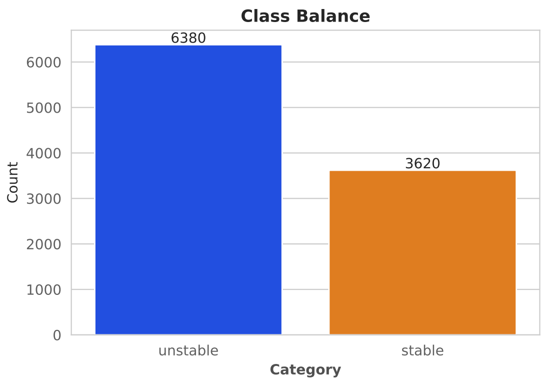


Figure 4 – Class Balance.

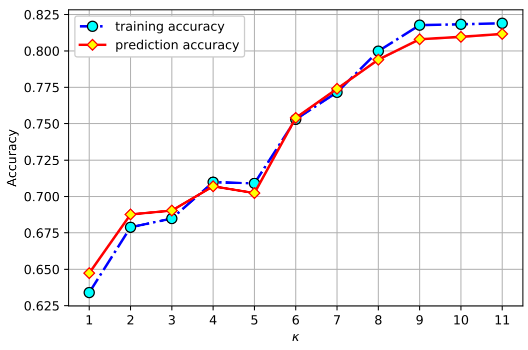


Figure 5 – Prediction vs training accuracy

demonstrates that the class balance is acceptable.

The data set is split randomly into training and test, such that 70% of the data points are considered for training the model and 30% of the samples are selected to test and evaluate the prediction accuracy of the model. We train the DSLR model for different values of κ and evaluate the prediction and training errors to select the true κ for which the model has the best performance. Here we select the best model by selecting the true number of nonzero variables, κ^*, and evaluating the training and the prediction accuracy of the DSLR model over training and test data samples. In particular, for each value of κ, we solve problem (41) and evaluate the model performance over both training and test data. Fig. 5 shows the results of accuracy for the different values of κ. According to the figure, the minimum value of κ which provides an acceptable model performance over both training and test data is $\kappa^* = 9$. In particular, there are no advantages to using more than 9 features. In other words, the combination of 9 features out of 12 features can achieve acceptable performance for the model. In order to evaluate the performance of the obtained model, we compare the DSLR model with other classifiers such as Lasso logistic regression and sparse linear Support Vector Machine (SVM). The SVM and the Lasso classifiers are tuned so that the same κ is provided. The accuracy results are provided in Table 1. Here *f-unstable* and *f-stable* are F_1 Scores for both unstable and stable categories. The *acc* is the overall accuracy of the method and *acu* is the area under the Receiver Operating Characteristic (ROC) curve. According to Table 1, the DSLR model outperforms the Lasso logistic regression and the SVM classifiers. It can be noticed that the DSLR chooses a better combination of features to improve its prediction accuracy.

Table 1 – Accuracy results

classifier	κ^*	f-unstable	f-stable	acc	auc
DSLR	9	0.88	0.74	0.83	0.80
Lasso	9	0.83	0.58	0.75	0.69
SVM	9	0.86	0.72	0.81	0.78

3.5 CONCLUSIONS

To summarize, the `DISGT` algorithm is a practical solution for the sparse optimization problem as it consists of computationally cheap steps and relies only on exchanging information between neighboring nodes. However, there are some limitations to the algorithm. One major drawback is that the convergence of the algorithm is highly dependent on the penalty parameter ρ, which must be sufficiently large for the algorithm to converge. If ρ is too small, the algorithm fails to converge, and finding an appropriate value of ρ can be challenging in many cases. Additionally, the `DISGT` algorithm cannot handle linear and/or nonlinear constraints other than sparsity constraints, and the gradient tracking algorithm employed in `DISGT` relies on projections to satisfy the constraints, which can be computationally expensive for complex sets. Given these limitations, there is a need for more stable and easy-to-tune distributed optimization algorithms that can be applied to a wide range of applications. To address these limitations, the next chapters will explore the use of mixed-integer nonlinear programming and distributed convex optimization frameworks. These approaches offer the potential to overcome the limitations of `DISGT` and enable the development of more versatile and efficient distributed optimization algorithms.

4 RELAXED-HYBRID ALTERNATING DIRECTION METHOD OF MULTIPLIERS ALGORITHM

This chapter addresses the solution of a distributed convex optimization problem with global coupling inequality constraints. The solution to this problem acts as a numerical kernel to solve the distributed CCP problem. By using the Lagrangian duality framework, the problem is transformed into a distributed consensus optimization problem and then based on the *Hybrid Alternating Direction Method of Multipliers* (H-ADMM), which merges distributed and centralized optimization concepts, a novel distributed algorithm is developed. In particular, we offer a reformulation of the original H-ADMM in an operator theoretical framework, which exploits the known relationship between ADMM and Douglas-Rachford splitting. In addition, our formulation allows us to generalize the H-ADMM by including a relaxation constant, not present in the the original design of the algorithm. Moreover, an adaptive penalty parameter selection scheme that consistently improves the practical convergence properties of the algorithm is proposed. Finally, the convergence results of the proposed algorithm are discussed and moreover, in order to present the effectiveness and the major capabilities of the proposed algorithm in off-line and on-line scenarios, Distributed Quadratic Programming (DQP), and Distributed Model Predictive Control (DMPC) problems are considered in the simulation section. This chapter is mainly written according to our previously published paper (OLAMA et al., 2019).

4.1 INTRODUCTION

The solution of distributed optimization problems is becoming increasingly central in many applications, such as control and power systems (NEDIĆ; LIU, 2018). In particular, many problems of interest in these fields can be cast as the following separable convex problem (BERTSEKAS, 2015):

$$\begin{aligned}
(\mathrm{P}) \qquad & \min_{x} \sum_{i=1}^{N} f_i(\mathrm{x}_i) \\
\text{subject to} & \sum_{i=1}^{N} c_{ik}(\mathrm{x}_i) \leq 0,\ k = 1,\ldots,L \\
& \mathrm{x}_i \in \mathcal{X}_i \subset \mathcal{R}^{n_i},\ i = 1,\ldots,N
\end{aligned}$$

where $f_i : \mathcal{R}^{n_i} \to \mathcal{R}$ and $c_{ik} : \mathcal{R}^{n_i} \to \mathcal{R}$ are convex functions and $\mathcal{X}_i$ are closed convex sets for all $i = 1,\ldots,N$. The $\mathrm{x}_i \in \mathcal{R}^{n_i}$ are the optimization variables that need to be computed and L is the number of inequality coupling constraints.

There are two different approaches to solve the problem (P): solving it directly, for example employing the Augmented Lagrangian Methods (ALM) (HE; HOU; YUAN, 2015), Alternating Direction Method of Multipliers (ADMM) (CHANG, X. et al., 2018), Interior Point (IP) (NECOARA; SUYKENS, 2009) or Proximal Point Algorithms (PPA) (BAI, J.; ZHANG, H.; LI, J., 2018); or using the Lagrangian duality framework (BERTSEKAS, 2015). Here we employ this second approach, which is briefly discussed in the following.

Let $\boldsymbol{\lambda} \in \mathcal{R}^L$ be the Lagrange multiplier associated with the inequality coupling constraint of (P). Then the Lagrangian function for problem (P) is defined as

$$\mathcal{L}(\mathrm{x},\boldsymbol{\lambda}) = \sum_{i=1}^{N} (f_i(\mathrm{x}_i) + \langle \boldsymbol{\lambda}, c_i(\mathrm{x}_i)\rangle) \tag{46}$$

where $\mathrm{x} = [\mathrm{x}_1,...,\mathrm{x}_N]^T$ and $c_i(\mathrm{x}_i) = [c_{i1}(\mathrm{x}_i),...,c_{iL}(\mathrm{x}_i)]^\top$. Accordingly, the dual problem can be expressed as follows:

$$\max_{\boldsymbol{\lambda}\geq 0} \min_{\mathrm{x}\in\mathcal{X}} \mathcal{L}(\mathrm{x},\boldsymbol{\lambda}) \equiv \max_{\boldsymbol{\lambda}\geq 0} - \max_{\mathrm{x}\in\mathcal{X}} -\mathcal{L}(\mathrm{x},\boldsymbol{\lambda}) \equiv \min_{\boldsymbol{\lambda}\geq 0} \max_{\mathrm{x}\in\mathcal{X}} -\mathcal{L}(\mathrm{x},\boldsymbol{\lambda}) \tag{47}$$

where $\mathcal{X} = \mathcal{X}_1 \times \cdots \times \mathcal{X}_N$. Therefore, the dual problem (D) is obtained as the following convex problem:

$$\min_{\boldsymbol{\lambda}\geq 0} \sum_{i=1}^{N} q_i(\boldsymbol{\lambda}).$$

where,

$$q_i(\boldsymbol{\lambda}) = \max_{\mathrm{x}_i\in\mathcal{X}_i} -(f_i(\mathrm{x}_i) + \langle \boldsymbol{\lambda}, c_i(\mathrm{x}_i)\rangle) \tag{48}$$

The problem (D) is a separable problem in which each agent has the necessary information to compute $q_i(\boldsymbol{\lambda})$. The dual problem (D) requires to minimize the sum of N convex functions in the unknown $\boldsymbol{\lambda}$, and thus in order to exploit this separable structure, this problem can be reformulated as a *consensus optimization* problem. In particular, we introduce the local copies $\boldsymbol{\lambda}_i$, $i = 1, \ldots, N$, of the variable $\boldsymbol{\lambda}$, and thus we can rewrite problem (D) as:

$$\text{(D)} \quad \min_{\boldsymbol{\lambda}_i\geq 0} \sum_{i=1}^{N} q_i(\boldsymbol{\lambda}_i)$$
$$\text{subject to} \quad \boldsymbol{\lambda}_1 = \boldsymbol{\lambda}_2 = \cdots = \boldsymbol{\lambda}_N$$

where the constraint imposes the consistency of the local $\boldsymbol{\lambda}_i$'s. In this chapter, we seek for the solution of (P) by solving the problem (D) using the proposed consensus optimization algorithm named Relaxed Hybrid Alternating Direction Method of Multipliers (RH-ADMM).

4.1.1 Literature Review

Consensus optimization problems arise in many engineering applications. Some examples include Machine Learning (BOYD, S. et al., 2011; SLAVAKIS; GIANNAKIS, Georgios B; MATEOS, 2014), Smart Grids (DALL'ANESE; SIMONETTO, 2018; BOLOGNANI; CARLI; TODESCATO, 2014; YILDIRIM et al., 2017), Wireless Sensor Networks (LEWIS, 2004; CARRON et al., 2014), Economic Power Dispatching (WANG, R. et al., 2018) and Distributed Model Predictive Control (DMPC) (WANG, Z.; ONG, C. J., 2017). During the past decades, various distributed numerical algorithms were proposed to solve the consensus optimization problem (FARINA et al., 2019; CHANG, T.-H., 2016; NOTARNICOLA et al., 2017; CHEN; YANG, Q., 2019; XIE et al., 2018). For instance, (FARINA et al., 2019) presented a fully asynchronous and distributed approach for dealing with the consensus optimization problems in which both the objective function and the constraints may be non-convex. Proximal dual consensus ADMM method proposed in (CHANG, T.-H., 2016) where the polyhedron constraints are transformed as quadratic penalty terms in the local problems, rendering the local problems efficiently solvable and consequently reducing the overall computational overhead of the agents. The ADMM, first proposed in (GLOWINSKI; MARROCO, 1975; GABAY; MERCIER, 1975), is among the most widely used and efficient algorithms for solving this class of distributed problems. As the name suggests, this algorithm is characterized by the alternating solution of two optimization sub-problems, and an update of the Lagrange multipliers. This division of the computational burden in two distinct steps makes the ADMM a computationally efficient algorithm, particularly when applied in distributed scenarios characterized by agents with reduced capabilities. For a comprehensive study of this algorithm we refer to (BOYD, S. et al., 2011) and (ECKSTEIN; YAO, 2015).

There are two classes of ADMM variants that have been proposed to solve the dual problem (D), which can be distinguished by the underlying communication network. A first category that hereafter will be referred to as *master-slave* is characterized by the presence of a *fusion center* (FC – the master) and a set of N nodes (the slaves). The nodes perform local minimization of the assigned cost functions q_i's, sending the results to the FC, which collates (averages) them, and then sends the result to all the nodes. ADMM variants based on a master-slave architecture have been proposed for example in (ZHANG, R.; KWOK, 2014; HONG, 2017). This choice of communication network ensures a faster convergence rate, at the cost, however, of introducing a single point of failure (the FC). Moreover, in many applications, it might be impractical to implement such a network.

A different architecture that can be chosen is the fully distributed one, in which the nodes collaborate only with their neighbors (*i.e.,* other nodes to which they are connected) without a global supervisor. In this framework, the nodes perform local computations and then share the results with their neighbors, which in turn send theirs, and each node then collates the information that it has received. Alongside the above mentioned (BOYD, S. et al., 2011), other implementations of a fully decentralized ADMM have been proposed for instance in (MOTA et al., 2013; WEI; OZDAGLAR, 2013; IUTZELER et al., 2016; PENG et al., 2016).

Recently, a new formulation of the ADMM has been proposed in (MA, M.; NIKOLAKOPOULOS; GIANNAKIS, Georgios B., 2018), called the *Hybrid ADMM* (H-ADMM). In particular, the algorithm introduces *local* fusion centers (LFC), that are tasked – much as the master in the master-slave architecture – with collating the local information of a subset of the agents collaborating to the solution of the problem. Differently from a fully centralized architecture, however, employing local fusion centers makes it possible to retain to a certain degree the flexibility of distributed algorithms, while at the same time accelerating the convergence rate.

Different parameter selection schemes have been proposed in the literature with the aim of accelerating, in practice, the convergence rate of the ADMM. For the centralized ADMM, parameter selections schemes have been proposed for example in (GHADIMI et al., 2015; GISELSSON; BOYD, S., 2017; XU et al., 2017a). However, these methods cannot be implemented in a distributed fashion, since information from the whole network would be required. Parameter selection algorithms specially designed for decentralized scenarios have been proposed in (SONG, C.; YOON; PAVLOVIC, 2016) for tuning the penalty parameter that characterizes the ADMM, and (XU et al., 2017b) for the step-size.

Finally, a well-known result is the equivalence of the ADMM to the *Douglas-Rachford splitting* (DRS) applied to the dual of the problem at hand (PENG et al., 2016).

4.1.2 Contributions

In this work, we will exploit that result to propose a different and more efficient formulation of the hybrid ADMM based on operator theory. For a survey on operator theory we refer the reader to (RYU; BOYD, S., 2016) and the comprehensive book (BAUSCHKE; COMBETTES, 2017), while the link between ADMM and DRS is highlighted for example in (PENG et al., 2016) and (DAVIS; YIN, 2017). The idea for using operator theory to solve convex optimization problems is to define a suitable operator or mapping T such that its fixed points, *i.e.* the points such that $\bar{x} = T\bar{x}$, are the solutions to the original problem. Therefore applying very powerful and widely studied fixed-point algorithms we can solve the optimization problem. The aim of this chapter is to propose an efficient, decentralized algorithm for solving the problem (P) based on H-ADMM. The contribution of this work is two-fold.

1. First, we use operator theory to develop a generalized version of the H-ADMM algorithm, namely the Relaxed H-ADMM algorithm, that we show to have better *practical* convergence rates than H-ADMM in a host of representative numerical problems, by suitably tuning the relaxation parameter α.
2. Second, a generalization of the adaptive selection strategy (SONG, C.; YOON;

PAVLOVIC, 2016) for the penalty parameter ρ of the centralized ADMM to work in a fully decentralized manner. In this context, each agent can compute its penalty parameter based solely on local information, and in particular on the primal and dual residuals the local problem.

4.2 BACKGROUND

4.2.1 Original H-ADMM

In this section, the basics of H-ADMM proposed in (MA, M.; NIKOLAKOPOULOS; GIANNAKIS, Georgios B., 2018) are discussed. Consider an undirected and connected graph $\mathcal{G} = (\mathcal{V},\mathcal{E})$, with $\mathcal{V} = \{1, \ldots, N\}$ being the set of nodes and $\mathcal{E}$, the set of edges. In particular, the original H-ADMM algorithm was used to solve the following convex optimization problem, which is a particular form of dual problem (D) without the non-negativity constraint on the decision variable $\boldsymbol{\gamma}$

$$\min_{\boldsymbol{\gamma}} \sum_{i=1}^{N} h_i(\boldsymbol{\gamma}) \tag{49}$$

where $h_i : \mathcal{R}^n \to \mathcal{R} \cup \{+\infty\}$ are CCP functions, each available only to the corresponding node i. In the literature two main approaches have been employed to solve this class of optimization problems. The first is the *centralized* approach[1] in which a *fusion center* (FC) collects and collates information computed by the nodes using the local information h_i. Formally, this approach modifies (49) into problem

$$\min_{\boldsymbol{\gamma}_i,\ i\in\mathcal{V}} \sum_{i=1}^{N} h_i(\boldsymbol{\gamma}_i) \tag{50}$$

$$\text{s.t. } \mathbf{z} = \boldsymbol{\gamma}_i \quad \forall i \in \mathcal{V} \tag{51}$$

where $\boldsymbol{\gamma}_i$ is a local copy of $\boldsymbol{\gamma}$ stored and updated by node i, and $\mathbf{z}$ is the variable, computed by the FC, that collates the information sent from all nodes. In particular, according to (49), each agent has enough information to construct its corresponding function h_i. However, the variable $\boldsymbol{\gamma}$ is shared between all agents which prevents the problems from being solved in a fully decentralized fashion. Therefore we introduce the auxiliary variable $\mathbf{z}$, and a local copy of $\boldsymbol{\gamma}_i$ is assigned to the i-th agent. By this reformulation, after termination of the algorithm, all $\boldsymbol{\gamma}_i$ variables will reach the same value, which is the solution of the problem (49).

The second is the *fully distributed* approach, in which the nodes solve the problem using local information and information received from the neighboring nodes. Fully distributed algorithms usually apply a consensus procedure to ensure that all nodes converge

[1] Sometimes referred to as *master-slave*.

to the same solution. In particular, the problem (49) is reformulated as a consensus optimization problem

$$\min_{\boldsymbol{\gamma}_i\ i\in\mathcal{V}} \sum_{i=1}^{N} h_i(\boldsymbol{\gamma}_i) \tag{52}$$

$$\text{s.t. } \boldsymbol{\gamma}_i = \boldsymbol{\gamma}_j \quad \forall(i,j) \in \mathcal{E}, \tag{53}$$

where only node-to-node constraints are enforced.

A third way has been very recently proposed in (MA, M.; NIKOLAKOPOULOS; GIANNAKIS, Georgios B., 2018), which merges the centralized and distributed approaches in a new framework called *hybrid distributed optimization*. The main idea is to introduce a set of M *local fusion centers* (LFC), each of which connects only a subset of the N nodes and has the task of collating the information computed by these nodes only.

In order to formalize the hybrid consensus approach, let us define the *hypergraph* $\mathcal{H} = (\mathcal{V},\bar{\mathcal{E}})$ which has the same set of nodes as $\mathcal{G}$ but instead of the set of edges $\mathcal{E}$ has a set of hyperedges $\bar{\mathcal{E}} = \{\bar{\mathcal{E}}_j\}_{j=1}^{M}$ each containing a subset of $\mathcal{V}$.

The problem (49) then becomes

$$\begin{aligned} &\min_{\boldsymbol{\gamma}_i,\ i\in\mathcal{V}} \sum_{i=1}^{N} h_i(\boldsymbol{\gamma}_i) \\ &\text{s.t. } \mathrm{y}_j = \boldsymbol{\gamma}_i, \quad \forall i \in \bar{\mathcal{E}}_j, \ \ \bar{\mathcal{E}}_j \in \bar{\mathcal{E}} \end{aligned} \tag{54}$$

where y_j is the variable updated by the j-th LFC.

Remark 5. *Clearly, the centralized approach is a particular case of hybrid consensus in which a single hyperedge containing all nodes is present, while the distributed approach is characterized by one hyperedge for each edge.*

4.3 RELAXED HYBRID ADMM ALGORITHM

By making use of operator theory, we propose the `RH-ADMM` algorithm for solving the dual problem (D). Based on the LFC framework, the dual problem (D) can be recast as

$$\begin{aligned} &\min \sum_{i=1}^{N} q_i(\boldsymbol{\lambda}_i) \\ &\text{s.t. } \boldsymbol{\lambda}_i \geq 0 \\ &\quad\ \boldsymbol{\lambda}_i = \mathrm{y}_j, \forall i \in \mathcal{E}_j, \forall \mathcal{E}_j \in \mathcal{E}. \end{aligned} \tag{55}$$

Requiring that $\boldsymbol{\lambda}_i \geq 0$ is equivalent to requiring that $\boldsymbol{\lambda}_i$ lies in the cone $\mathcal{R}^n_+$. Therefore, the problem (55) is equivalent to

$$\begin{aligned} &\min \sum_{i=1}^{N} q_i(\boldsymbol{\lambda}) + \iota_{\mathcal{R}^n_+}(\boldsymbol{\lambda}) \\ &\boldsymbol{\lambda}_i = \mathbf{y}_j, \forall i \in \mathcal{E}_j, \forall \mathcal{E}_j \in \mathcal{E} \end{aligned} \tag{56}$$

and the indicator function $\iota_{\mathcal{R}^n_+}(\boldsymbol{\lambda})$ is 0 if $\boldsymbol{\lambda} \geq 0$, and $+\infty$ otherwise. Letting T_e be the number of equality (consistency) constraints and recalling that L is the number of inequality constraints, we define $\boldsymbol{\lambda}_e = [\boldsymbol{\lambda}_1 \cdots \boldsymbol{\lambda}_N]^\top \in \mathcal{R}^{NL}$, $\mathbf{y} = [\mathbf{y}_1 \cdots \mathbf{y}_M]^\top \in \mathcal{R}^{ML}$, and $\sum_{i=1}^{N} q_i(\boldsymbol{\lambda}_i) = q(\boldsymbol{\lambda})$, and then the consensus problem (56) can be reformulated as

$$\begin{aligned} &\min_{\boldsymbol{\lambda}_e, \mathbf{y}} q(\boldsymbol{\lambda}_e) + \iota_{\mathcal{R}^n_+}(\mathbf{y}) \\ &\text{s.t. } \mathbf{A}\boldsymbol{\lambda}_e + \mathbf{B}\mathbf{y} = \mathbf{0} \end{aligned} \tag{57}$$

for suitable matrices $\mathbf{A} \in \mathcal{R}^{T_e \times NL}$ and $\mathbf{B} \in \mathcal{R}^{T_e \times ML}$. Notice that the t-th rows of $\mathbf{A}$ and $\mathbf{B}$ relative to constraint $\boldsymbol{\lambda}_i = \mathbf{y}_j$ are, respectively,

$$\begin{aligned} [\mathbf{A}]_t &= [\underbrace{0 \cdots\cdots 0}_{i-1 \text{ times}} -1 \underbrace{0 \cdots\cdots 0}_{N-i \text{ times}}] \\ [\mathbf{B}]_t &= [\ \underbrace{0 \cdots 0}_{j-1 \text{ times}}\ 1\ \underbrace{0 \cdots 0}_{M-j \text{ times}}\ ] \end{aligned} \tag{58}$$

The goal is now to employ the Krasnosel'skiĭ-Mann iteration and the DR operator to reformulate this problem as a fixed-point problem. In particular, we consider the Fenchel dual of problem (57) which is defined as

$$\min_{\mathbf{w}} d_q(\mathbf{w}) + d_g(\mathbf{w}) \tag{59}$$

where $\mathbf{w}$ is the vector of dual variables and

$$\begin{aligned} d_q(\mathbf{w}) &= q^*(\mathbf{A}^\top \mathbf{w}) \\ d_g(\mathbf{w}) &= \iota^*_{\mathcal{R}^n_+}(\mathbf{B}^\top \mathbf{w}) \end{aligned}$$

Notice that the primal problem is convex with linear constraints and hence satisfies Slater's conditions for strong duality. In order to solve the dual problem (59), let us recall the Douglas-Rachford operator, which is defined as follows.

Definition 20 (Douglas-Rachford (DR) Operator). *The DR operator is defined as*

$$T_{DR} = \text{refl}_{\rho d_q} \circ \text{refl}_{\rho d_g}, \tag{60}$$

which is non-expansive.

Clearly, the dual problem is unconstrained and depends on a single variable, therefore we can define the DR operator for problem (59) and apply the Krasnosel'skiĭ-Mann iteration to find its fixed points

$$\mathrm{z}(k+1) = (1-\alpha)\mathrm{z}(k) + \alpha\, \mathrm{refl}_{\rho d_q}(\mathrm{refl}_{\rho d_g}(\mathrm{z}(k))). \tag{61}$$

The optimum of the problem (59) can be derived from the fixed points of the DR operator through the proximal operator $\mathrm{prox}_{\rho d_q}$. By substituting the definition of reflective operator (61), we obtain,

$$\mathrm{z}(k+1) = \mathrm{z}(k) + 2\alpha(\boldsymbol{\zeta} - \boldsymbol{\psi}) \tag{62}$$

where

$$\boldsymbol{\psi} = \mathrm{prox}_{\rho d_g}(\mathrm{z}(k)) \tag{63}$$

$$\boldsymbol{\zeta} = \mathrm{prox}_{\rho d_q}(2\,\mathrm{prox}_{\rho d_g}(\mathrm{z}(k)) - \mathrm{z}(k) = \mathrm{prox}_{\rho d_q}(2\boldsymbol{\psi} - \mathrm{z}(k)) \tag{64}$$

Remark 6. *By choosing the relaxation parameter* $\alpha = 0.5$*, the fixed-point iterations* (62)-(64) *yield the well-known Douglas-Rachford (DR) splitting algorithm (PENG et al., 2016), which in turn correspond to the H-ADMM when applied to the dual of problem* (59)*. However, we show that, in practice, the possibility of choosing* α *in the interval* (0,1) *can yield better convergence rates.*

Remark 7. *The* `RH-ADMM` *algorithm can be applied to any general distributed problem in the form* (49)*, but in this chapter we are interested in solving distributed problems with coupling constraints employing the dual framework.*

The following proposition presents the derivation of the `RH-ADMM` based on the fixed-point iterations (62)-(64).

Proposition 9. *The general form of* `RH-ADMM` *algorithm based on the fixed-point iterations* (62)-(64) *is represented by the following iterations*

$$\mathrm{y}(k) = \arg\min_{\mathrm{y}} \left\{ g(\mathrm{y}) - \langle \mathrm{B}^\top \mathrm{z}(k), \mathrm{y}\rangle + \frac{\rho}{2}\|\mathrm{B}\mathrm{y}\|^2 \right\} \tag{65}$$

$$\boldsymbol{\psi}(k) = \mathrm{z}(k) - \rho \mathrm{B}\mathrm{y}(k) \tag{66}$$

$$\boldsymbol{\lambda}_e(k) = \arg\min_{\boldsymbol{\lambda}_e} \left\{ q(\boldsymbol{\lambda}_e) - \langle 2\boldsymbol{\psi}(k) - \mathrm{z}(k), \mathrm{A}\boldsymbol{\lambda}_e\rangle + \frac{\rho}{2}\|\mathrm{A}\boldsymbol{\lambda}_e\|^2 \right\} \tag{67}$$

$$\boldsymbol{\xi}(k) = 2\boldsymbol{\psi}(k) - \mathrm{z}(k) - \rho \mathrm{A}\boldsymbol{\lambda}_e(k) \tag{68}$$

$$\mathrm{z}(k+1) = \mathrm{z}(k) + 2\alpha(\boldsymbol{\xi}(k) - \boldsymbol{\psi}(k)). \tag{69}$$

Proof. Consider the equation (63), by the definition of proximal operator and of the dual function d_g, it is necessary to find the argument of the following minimization problem

$$
\begin{aligned}
&\min_{\mathrm{s}} \left\{ d_g(\mathrm{s}) + \frac{1}{2\rho} \|\mathrm{s}-\mathrm{z}\|_2^2 \right\} = \\
&\min_{\mathrm{s}} \left\{ \max_{\mathbf{u}} \left\{ \mathrm{s}^T \mathbf{B}\mathbf{u} - g(\mathbf{u}) \right\} + \frac{1}{2\rho} \|\mathrm{s}-\mathrm{z}\|_2^2 \right\} = \\
&\max_{\mathbf{u}} \left\{ \min_{\mathrm{s}} \left\{ \mathrm{s}^T \mathbf{B}\mathbf{u} + \frac{1}{2\rho} \|\mathrm{s}-\mathrm{z}\|_2^2 \right\} - g(\mathbf{u}) \right\}
\end{aligned} \tag{70}
$$

By imposing the first-order optimality condition (BERTSEKAS, 2015), the solution of the inner minimization problem is obtained as $\mathrm{s}^* = \mathrm{z} - \rho\mathbf{B}\mathbf{u}$ and hence,

$$
\min_{\mathrm{s}} \left\{ \mathrm{s}^T \mathbf{B}\mathbf{u} + \frac{1}{2\rho} \|\mathrm{s}-\mathrm{z}\|_2^2 \right\} = \mathrm{z}^T \mathbf{B}\mathbf{u} - \frac{\rho}{2} \|\mathbf{B}\mathbf{u}\|_2^2 \tag{71}
$$

Therefore,

$$
\max_{\mathbf{u}} \left\{ \min_{\mathrm{s}} \left\{ \mathrm{s}^T \mathbf{B}\mathbf{u} + \frac{1}{2\rho} \|\mathrm{s}-\mathrm{z}\|_2^2 \right\} - g(\mathbf{u}) \right\} = -\min_{\mathbf{u}} \left\{ g(\mathbf{u}) - \mathrm{z}^T \mathbf{B}\mathbf{u} + \frac{\rho}{2} \|\mathbf{B}\mathbf{u}\|_2^2 \right\} \tag{72}
$$

This problem can now be solved applying the method of multipliers (BOYD, S. et al., 2011) which gives the desired result

$$
\mathrm{y}(k) = \arg\min_{\mathbf{y}} \left\{ g(\mathrm{y}) - \langle \mathbf{B}^\top \mathrm{z}(k), \mathrm{y} \rangle + \frac{\rho}{2} \|\mathbf{B}\mathrm{y}\|^2 \right\} \tag{73}
$$

$$
\boldsymbol{\psi}(k) = \mathrm{z}(k) - \rho\mathbf{B}\mathrm{y}(k), \tag{74}
$$

and that proves (65) and (66). The same procedure can be applied for equations (67) and (68). □

We exploit now the particular structure of the problem at hand to simplify the update equations that characterize the `RH-ADMM` algorithm. First of all, using the definition of indicator function we see that equation (65) can be equivalently written as

$$
\mathrm{y}(k) = \arg\min_{\mathrm{y} \geq 0} \left\{ -\langle \mathbf{B}^\top \mathrm{z}(k), \mathrm{y} \rangle + \frac{\rho}{2} \|\mathbf{B}\mathrm{y}\|^2 \right\} \tag{75}
$$

which has the following solution

$$
\mathrm{y}(k) = \frac{1}{\rho} \max \left\{ \mathbf{E}^{-1} \mathbf{B}^\top \mathrm{z}(k), \mathbf{0} \right\} \tag{76}
$$

where the fact that

$$
\mathbf{B}^\top \mathbf{B} = \mathrm{diag}\{e_j\} =: \mathbf{E}
$$

was used, and e_j is the cardinality of j-th hyperedge. It should be noted that the max is taken component-wise. As a consequence, from (66) it holds

$$
\boldsymbol{\psi}(k) = \left(\mathbf{I}_{T_e} - \max \left\{ \mathbf{E}^{-1} \mathbf{B}^\top \mathrm{z}(k), \mathbf{0} \right\} \right) \mathrm{z}(k) \tag{77}
$$

and from (68)

$$\mathbf{x}_i(k) = \left(\mathbf{I}_{T_e} - 2\max\left\{\mathbf{E}^{-1}\mathbf{B}^\top \mathrm{z}(k), \mathbf{0}\right\}\right)\mathrm{z}(k) - \rho A\boldsymbol{\lambda}_e(k). \tag{78}$$

Moreover, it follows from (69) that

$$\mathrm{z}(k+1) = \left(\mathrm{I} - 2\alpha\max\left\{\mathrm{E}^{-1}\mathrm{B}^\top \mathrm{z}(k), \mathbf{0}\right\}\right)\mathrm{z}(k) - 2\alpha\rho\mathrm{A}\boldsymbol{\lambda}_e(k). \tag{79}$$

Notice that (79) depends only on the vectors z and $\boldsymbol{\lambda}$ at time k. We assume that the constraints are ordered according to the index of the corresponding hyperedges, that is, first those of $\mathcal{E}_1$, then $\mathcal{E}_2$ up to $\mathcal{E}_M$. In this case, the matrix B has the following structure

$$\mathrm{B} = \left[\begin{array}{cccc} 1 & 0 & \cdots & 0 \\ \vdots & \vdots & \vdots & \vdots \\ 1 & 0 & \cdots & 0 \\ \hline & \vdots & & \\ \hline 0 & 0 & \cdots & 1 \\ \vdots & \vdots & \vdots & \vdots \\ 0 & 0 & \cdots & 1 \end{array}\right]. \tag{80}$$

Therefore, it is possible to see that

$$\mathrm{B}\mathrm{E}^{-1}\mathrm{B}^\top = \mathrm{diag}\left\{\frac{1}{e_j}\mathbf{1}_{e_j} \,\middle|\, j = 1, \ldots, M\right\} \tag{81}$$

where $e_j = |\mathcal{E}_j|$. Using the definitions of **B** and **E**, it follows that

$$\boldsymbol{\psi}(k) = \mathrm{z}(k) - \begin{bmatrix} \vdots \\ \max\left\{\frac{1}{e_j}\max\left\{\sum_{l\in\mathcal{E}_j} z_{j,l}(k), 0\right\}, 0\right\} \\ \vdots \end{bmatrix}. \tag{82}$$

Using the results above we have that the row of $2\boldsymbol{\psi}(k) - \mathrm{z}(k)$ relative to the constraint between hyperedge j and node i is given by

$$\left[2\boldsymbol{\psi}(k) - \mathrm{z}(k)\right]_{j,i} = z_{j,i}(k) - \frac{2}{e_j}\max\left\{\sum_{l\in\mathcal{E}_j} z_{j,l}(k), 0\right\} \tag{83}$$

where the notation $z_{j,i}$ indicates first the hyperedge's index and then the node's index.

Consider now (67), using the definition of q we can rewrite it as

$$\boldsymbol{\lambda}_e(k) = \underset{\boldsymbol{\lambda}_e}{\arg\min}\left\{\sum_{i=1}^{N} q_i(\boldsymbol{\lambda}_i) - \langle 2\boldsymbol{\psi}(k) - \mathrm{z}(k), \mathrm{A}\boldsymbol{\lambda}_e\rangle + \frac{\rho}{2}\sum_{i=1}^{N} d_i \|\boldsymbol{\lambda}_i\|^2\right\} \tag{84}$$

where the fact that

$$\mathbf{A}^\top \mathbf{A} = \text{diag}\{d_i\} =: \mathbf{D}$$

was used. Moreover, it is clear that the element of $\mathbf{A}\boldsymbol{\lambda}_e$ corresponding to the j,i-th constraint is $-\boldsymbol{\lambda}_i$, therefore it is possible to derive

$$(2\boldsymbol{\psi}(k) - \mathrm{z}(k))^\top \mathbf{A}\boldsymbol{\lambda}_e = -\sum_{i=1}^{N} \sum_{j\in\mathcal{E} \text{ s.t. } i\in\mathcal{E}_j} \left(z_{j,i}(k) - \frac{2}{e_j} \max\left\{ \sum_{l\in\mathcal{E}_j} z_{j,l}(k), 0 \right\} \right)^\top \boldsymbol{\lambda}_i. \quad (85)$$

Then, clearly (84) is a separable minimization problem, and the primal variable of each node can be computed as follows:

$$\boldsymbol{\lambda}_i(k) = \arg\min_{\boldsymbol{\lambda}_i} \left\{ q_i(\boldsymbol{\lambda}_i) + \frac{\rho d_i}{2} \|\boldsymbol{\lambda}_i\|^2 + \right.$$
$$\left. + \sum_{j\in\mathcal{E} \text{ s.t. } i\in\mathcal{E}_j} \left(z_{j,i}(k) - \frac{2}{e_j} \max\left\{ \sum_{l\in\mathcal{E}_j} z_{j,l}(k), 0 \right\} \right)^\top \boldsymbol{\lambda}_e \right\}. \quad (86)$$

Note that update (86) depends on information that the node receives from all the hyperedges it is connected to. In particular, it needs to receive the sum of all the auxiliary variables of the hyperedge, and separately also the variable $z_{j,i}$. According to definition of q_i, the minimization problem (86) can be stated as finding the optimal point of the following problem:

$$\min_{\boldsymbol{\lambda}_i} \max_{\mathrm{x}_i\in\mathcal{X}_i} \varphi(\mathrm{x}_i,\boldsymbol{\lambda}_i) \quad (87)$$

where,

$$\varphi(\mathrm{x}_i,\boldsymbol{\lambda}_i) = \left\{ -\left(f_i(\mathrm{x}_i) + \langle \lambda, c_{ik}(\mathrm{x}_i)\rangle\right) + \frac{\rho d_i}{2} \|\boldsymbol{\lambda}_i\|^2 \right.$$
$$\left. + \sum_{j\in\mathcal{E} \text{ s.t. } i\in\mathcal{E}_j} \left(z_{j,i}(k) - \frac{2}{e_j} \max\left\{ \sum_{l\in\mathcal{E}_j} z_{j,l}(k), 0 \right\} \right)^\top \boldsymbol{\lambda}_i \right\}.$$

Since, $\varphi(\mathrm{x}_i,\boldsymbol{\lambda}_e)$ has a saddle point, (87) can be rewritten as follows:

$$\min_{\boldsymbol{\lambda}_i} \max_{\mathrm{x}_i\in\mathcal{X}_i} \varphi(\mathrm{x}_i,\boldsymbol{\lambda}_i) = \max_{\mathrm{x}_i\in\mathcal{X}_i} \min_{\boldsymbol{\lambda}_i} \varphi(\mathrm{x}_i,\boldsymbol{\lambda}_i) \quad (88)$$

The inner minimization problem is a convex unconstrained problem, whose minimum can easily be obtained as follows:

$$\boldsymbol{\lambda}_i(k+1) = \frac{1}{\rho d_i} \left(c(\mathrm{x}_i) - \sum_{j\in\mathcal{E}, \text{ s.t. } i\in\mathcal{E}_j} \left(z_{j,i}(k) - \frac{2}{e_j} \max\left\{ \sum_{l\in\mathcal{E}_j} z_{j,l}(k), 0 \right\} \right) \right). \quad (89)$$

By substituting $\boldsymbol{\lambda}_i(k+1)$ into $\varphi(\mathrm{x}_i,\boldsymbol{\lambda}_i)$, a local convex programming problem (L) with respect to x_i is obtained as

$$\text{(L)} \quad \min_{\mathrm{x}_i} \; -\varphi(\mathrm{x}_i,\boldsymbol{\lambda}_i(k+1))$$
$$\text{subject to } \mathrm{x}_i \in \mathcal{X}_i,$$

which can be solved easily using standard convex optimization algorithms. In the case of some particular applications such as MPC, usually, this problem can be stated as a boxed constrained concave Quadratic Programming (QP) problem, which can be solved explicitly using the projection methods (BERTSEKAS, 2015). Finally, the auxiliary variable relative to the j,i-th constraint can be updated using

$$z_{j,i}(k+1) = z_{j,i}(k) - \frac{2\alpha}{e_j}\max\left\{\sum_{l\in\mathcal{E}_j} z_{j,l}(k), 0\right\} + 2\alpha\rho\boldsymbol{\lambda}_i(k) \tag{90}$$

which the j-th LFC can compute with local information – the other $z_{j,l}$ variables – and the state $\boldsymbol{\lambda}$ that it receives from the i-th node. Therefore the `RH-ADMM` for consensus optimization is fully characterized by the two update equations (89) and (90). The iteration (89), is performed by each node, and (90), is performed by each LFC. Moreover, the following theorem presents the convergence result of the proposed algorithm:

Theorem 2. *The proposed `RH-ADMM` converges to the solution of problem (D) for any choice of $\alpha \in (0,1)$ and $\rho > 0$, such that*

$$\|\boldsymbol{\lambda}(k) - \boldsymbol{\lambda}^*\| \to 0$$

as $k \to \infty$ where $\boldsymbol{\lambda}^*$ is the optimal solution of the problem.

Proof. According to (61) and Theorem 1, it can be noticed that the fixed-point iteration strongly converges,to the fixed point of the DR operator in the sense of Definition 16 as it is the average of a non-expansive operator. Since q is CCP, then so is its convex conjugate (BAUSCHKE; COMBETTES, 2017); therefore the DR applied to the dual of the problem (P) converges (BAUSCHKE; COMBETTES, 2017). Since Slater's condition holds then there is strong duality and the optimal solution to the primal is reached. $\square$

Remark 8. *Theorem 2 states that the proposed `RH-ADMM` algorithm converges to the optimal point of the problem (D) for any $\rho > 0$ and $\alpha \in (0,1)$. However, the rate of convergence depends on the structure of the objective functions. For instance, in the case of strong convexity and strong smoothness of the local objective functions, the `RH-ADMM` algorithm can converge linearly (WEI; OZDAGLAR, 2013; MA, M.; NIKOLAKOPOULOS; GIANNAKIS, Georgios B., 2018). Moreover, it is worth mentioning that making these assumptions for the local objective function q_i is relatively restrictive as the dual objective functions of the local problems cannot be convex in most cases.*

4.3.1 Adaptive Penalty Parameter Selection

In the current section we present an *adaptive penalty parameter selection scheme*, based on (SONG, C.; YOON; PAVLOVIC, 2016). First of all, we define the primal and dual residuals of the H-ADMM as

$$\mathrm{r}(k) = \mathbf{A}\boldsymbol{\lambda}_e(k) + \mathbf{B}\mathrm{y}(k) \tag{91}$$

$$\mathrm{s}(k) = \rho\mathbf{A}^\top\mathbf{B}(\mathrm{y}(k) - \mathrm{y}(k-1)) \tag{92}$$

which will be used to select the penalty parameter for the next iteration. Recalling that $\mathrm{y}(k) = \mathbf{E}^{-1}\mathbf{B}^\top\mathrm{z}(k)/\rho$, it is possible to compute the i-th coordinate of s as

$$s_i(k) = -\sum_{j\in\mathcal{E},\,\text{s.t. } i\in\mathcal{E}_j}\left(\frac{1}{e_j}\sum_{l\in\mathcal{E}_j}\left(z_{j,l}(k) - z_{j,l}(k-1)\right)\right). \tag{93}$$

On the other hand $\mathrm{r} \in \mathcal{R}^{LT_e}$ and the t-th block of coordinates can be written as

$$r_t(k) = -\boldsymbol{\lambda}_i(k) + \frac{1}{\rho e_j}\max\left\{\sum_{l\in\mathcal{E}_j} z_{j,l}(k), 0\right\}.$$

Consider then all the coordinates of r that depend on $\boldsymbol{\lambda}_i$, stacking them up gives

$$\mathrm{r}_i(k) = -I_{|\mathcal{N}_i|}\boldsymbol{\lambda}_i(k) + \frac{1}{\rho}\begin{bmatrix} \vdots \\ \frac{1}{e_j}\max\left\{\sum\limits_{l\in\mathcal{E}_j} z_{j,l}(k), 0\right\} \\ \vdots \end{bmatrix} \tag{94}$$

with $\mathcal{N}_i = \{j \in \mathcal{E} \text{ s.t. } i \in \mathcal{E}_j\}$. The penalty selection is now carried out by each node with the following criterion (proposed in (SONG, C.; YOON; PAVLOVIC, 2016))

$$\rho_i(k+1) = \begin{cases} \rho_i(k)(1+\tau(k)) & \text{IF } \|\mathrm{r}_i(k)\| > \mu\,\|s_i(k)\| \\ \rho_i(k)(1+\tau(k))^{-1} & \text{IF } \|s_i(k)\| > \mu\,\|\mathrm{r}_i(k)\| \\ \rho_i(k) & \text{otherwise,} \end{cases} \tag{95}$$

with $\mu > 0$ and $\tau(k) > 0$ for each $k \in \mathbb{N}$ and $\sum_k \tau(k) < \infty$. This last condition is for instance satisfied if the penalty selection is applied only for a finite number of iterations, in which case the algorithm still converges with the usual properties after a transitory period (BOYD, S. et al., 2011).

Notice that i receives from the LFCs the quantities

$$(1/e_j)\max\left\{\sum_{l\in\mathcal{E}_j} z_{j,l}(k), 0\right\},$$

hence it has all the necessary information to compute the residuals. Moreover, the LFC need the quantity $\rho\boldsymbol{\lambda}(k)$ to update the corresponding $z_{j,i}$ variable, so it is sufficient to send $\rho_i(k)\boldsymbol{\lambda}(k)$ (instead of only $\boldsymbol{\lambda}(k)$) to each of the LFC connected to i to communicate the penalty selected as well as the updated local state. The final RH-ADMM algorithm along with the adaptive step-size scheme is provided in Algorithm 2. Clearly, the nodes

Algorithm 2 Basic steps of RH-ADMM Algorithm

```
1: Specify accepted optimality tolerances $\varepsilon_{\mathrm{p}}$ and $\varepsilon_{\mathrm{d}}$.
2: **Input:**$\alpha$, $\tau$, $\mu$, $\rho_i(0)$, $z_{i,j}(0)$, $y_i(0)$
3: **Output:** Optimal solution $\mathrm{x}_i^*$, $i$ = 1,...$N$
4: **while** $\|s_i\| \leq \varepsilon_{\mathrm{d}}$ **and** $\|r_i\| \leq \varepsilon_{\mathtt{rhadmm}}$ **do**
5:     Compute $\rho_i$ according to (95)
6:     Update $\boldsymbol{\lambda}_i$ according to (89)
7:     Update $\mathrm{x}_i$ by solving problem (L)
8:     Update $z_{i,j}$ according to (90)
   **return** $\mathrm{x}_i^*$, $i$ = 1,...,$N$
```

need only information from the LFCs they are connected to, while the LFC receive and aggregate the information from the nodes in their corresponding hyperedge only.

4.4 NUMERICAL EXPERIMENTS

In what follows, the proposed RH-ADMM is tested on the well-known Distributed Quadratic Programming (DQP) problem. In the first example, the practical convergence of the algorithm with respect to different relaxation and penalty parameters (namely α and ρ) is evaluated and, in the second example, the algorithm capabilities for solving dynamic optimization problems is investigated. The RH-ADMM algorithm has been implemented in SCOT software framework and the parallel computation is done by means of a message passage interface.

4.4.1 Distributed Quadratic Programming

The convergence rate of the proposed algorithm is evaluated in an application to Distributed Quadratic Programming (DQP) in the same form of problem (D), and then compared with the Distributed Gradient Descent Method (DGDM) (NEDIĆ; OLSHEVSKY, 2015) and the original H-ADMM (MA, M.; NIKOLAKOPOULOS; GIANNAKIS, Georgios B., 2018). In this problem, a ring graph with N = 100 nodes is considered. The local problems includes 50 decision variables. It is worth mentioning that all of the parameters of DGDM and the original H-ADMM algorithm are selected according to the indications included in the respective references. Moreover, the effectiveness of the proposed algorithm under different values of the tuning parameters, such as ρ, α and M, is discussed. The metric for the evaluation of the proposed algorithms is the

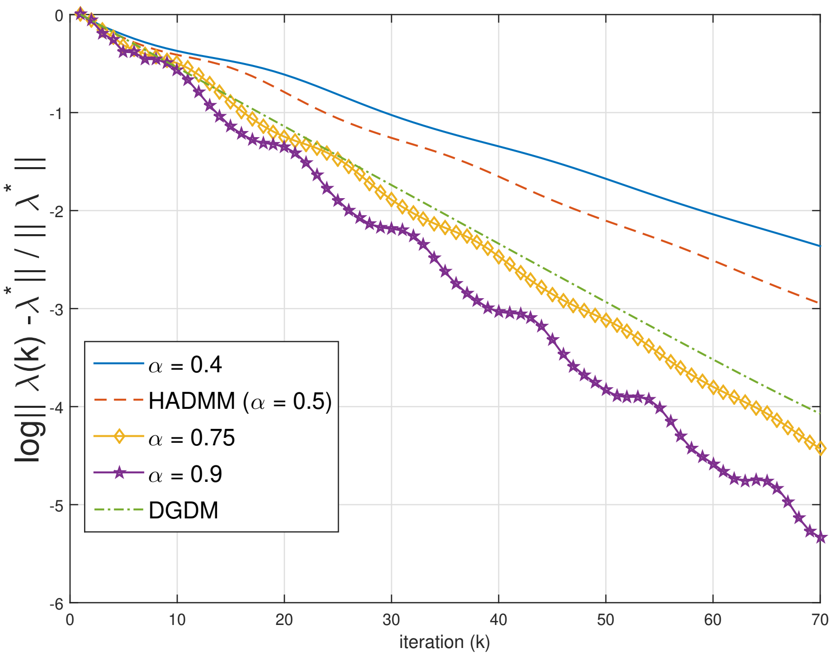


Figure 6 – Relative error results for different α (fixed ρ).

relative error. The relative error in the logarithmic scale at k-th iteration is defined as

$$\log \frac{\sum_{i=1}^{N} \|\boldsymbol{\lambda}(k) - \lambda^*\|}{\sqrt{N}\,\|\boldsymbol{\lambda}^*\|}. \tag{96}$$

The convergence results are depicted in Figures 6-9. Figures 6 and 7 depict the relative error for different values of the relaxation parameter α, with a fixed penalty and an adaptive penalty selection scheme, respectively. For the adaptive penalty, the initial value is equal to $\rho_0 = 1$. As the figures show, in both cases, tuning the parameter α can improve the speed of convergence of the algorithm.

Recalling that $\alpha = 0.5$ characterizes the H-ADMM, we can see that for values $\alpha > 0.5$ the RH-ADMM obtains better performances than the H-ADMM, hence justifying the use of the slightly more complex RH-ADMM. Moreover, the RH-ADMM with values of $\alpha > 0.5$ outperforms the DGDM, while the H-ADMM not always does. It is worth mentioning since both primal and dual residuals tend to zero after some iteration, the adaptive penalty scheme converges. Figure 8 shows the performance of the proposed algorithm for different values of the fixed penalty parameter ρ, and with relaxation parameter $\alpha = 0.9$. As can be seen, the penalty parameter can considerably affect the convergence rate of the algorithm and this fact is indeed exploited when we apply the adaptive penalty selection. As depicted in the figure, the best result is given by $\rho = 2$, which is very close to the value of 1.5 to which $\rho(k)$ in the adaptive scheme converges.

Finally, we evaluate how the number of LFCs, M, affects the convergence of the proposed algorithm. Figure 9 depicts the relative error for different values of M,

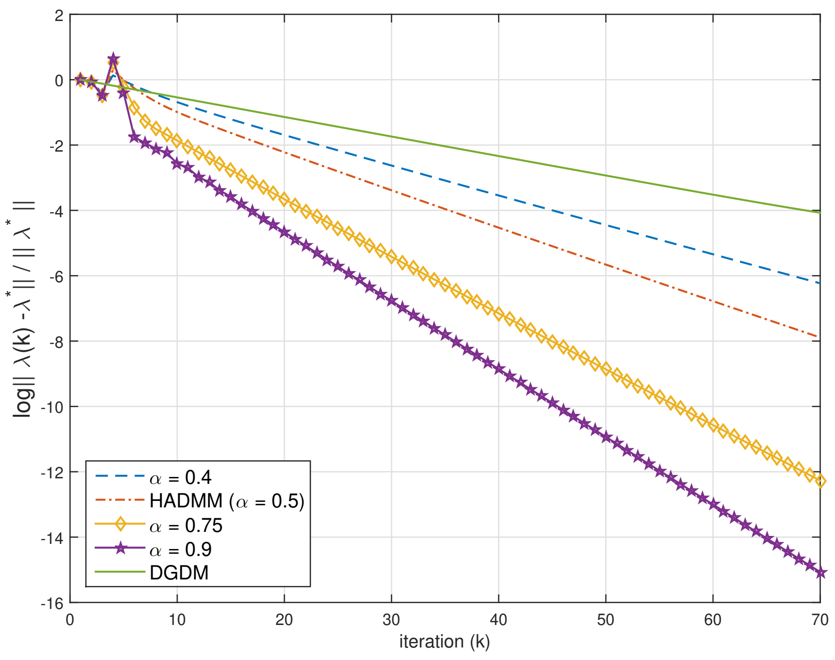


Figure 7 – Relative error results for different α (adaptive ρ).

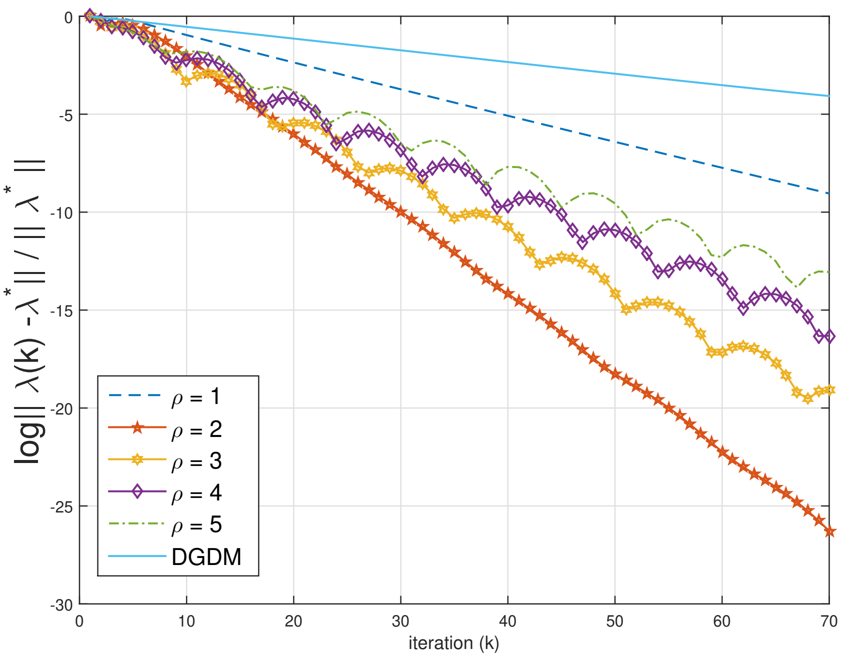


Figure 8 – Relative error results for different ρ and $\alpha = 0.9$.

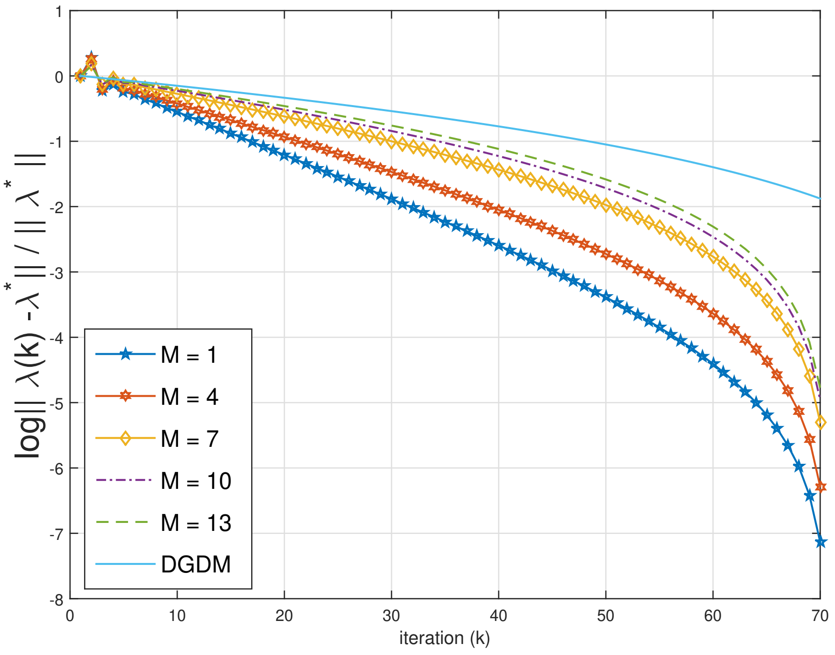


Figure 9 – Relative error results for different LFCs.

and it is clear that the best performance is obtained by a centralized architecture – as observed also in (WEI; OZDAGLAR, 2013). However, we need to keep in mind that a larger number of LFCs guarantees better resilience of the network by removing the single central point of failure. Therefore there is a trade-off between the convergence rate and the resilience of the agents' network.

4.4.2 Distributed Model Predictive Control (DMPC) With Coupling Inequality Constraints

The implementation of the RH-ADMM algorithm and its application to DMPC problems is discussed below. In the context of distributed MPC, one interesting problem is to compute the control when the agents are dynamically coupled. Most of distributed MPC approaches are developed for this type of problems. However, these distributed strategies are not suitable when the coupling arises from the constraints. To the best of our knowledge, DMPC algorithms for the distributed problems with global inequality constraints are somewhat limited (SCHERER, H. et al., 2015; CAMPONOGARA; SCHERER, H. F., 2011). Therefore, it seems to be crucial to develop efficient and reliable fully decentralized algorithms to deal with such a complex problem. Following by this explanation, in this section, the RH-ADMM algorithm is implemented and tested to solve DMPC problems with coupling inequality constraints. In order to do this, the proposed RH-ADMM algorithm was implemented for a multi-tank benchmark problem to show the effectiveness of the RH-ADMM algorithm for a dynamic implementation.

Consider the problem of controlling N discrete-time Linear Time-Invariant (LTI) systems using the Model Predictive Control (MPC). Hereafter we refer to this problem as the Distributed MPC (DMPC). The dynamic of each system is defined as the following state-space equations

$$\mathbf{x}_i(t+1) = \mathbf{A}_i\mathbf{x}_i(t) + \mathbf{B}_i\mathbf{u}_i(t) \tag{97}$$

$$\mathbf{x}_i(t) \in \mathcal{X}_i,\ \mathbf{u}_i(t) \in \mathcal{U}_i \tag{98}$$

where, $\mathbf{x}_i(t) \in \mathcal{R}_i^n$ and $\mathbf{u}_i(t) \in \mathcal{R}_i^m$ are the vectors of states and inputs, respectively. $\mathcal{X}_i$ and $\mathcal{U}_i$ are suitable polytopes which define local constraints of each system. The systems have to collectively satisfy the coupled constraint

$$\sum_{i=1}^{N} \mathbf{H}_i\mathbf{x}_i(t) + \mathbf{G}_i\mathbf{u}_i(t) \leq \mathbf{b} \tag{99}$$

with matrices $\mathbf{H}_i$ and $\mathbf{G}_i$ of suitable dimensions. Accordingly, the DMPC problem is defined as the following separable convex optimization problem.

$$\min_{\mathbf{U}_i} \sum_{i=1}^{N} J_i(\mathbf{x}_i(t), \mathbf{U}_i) \tag{100}$$

$$\text{subject to} \sum_{i=1}^{N} \mathbf{H}_i\mathbf{x}_i^j + \mathbf{G}_i\mathbf{u}_i^j \leq \mathbf{b},\ \forall j = 0,1\ldots,T-1 \tag{101}$$

$$\mathbf{U}_i \in \mathcal{K}_i. \tag{102}$$

$J_i(\mathbf{x}_i(t), \mathbf{U}_i)$ are the local cost functions of each agent and defined as

$$J_i(\mathbf{x}_i(t), \mathbf{U}_i) = \sum_{j=1}^{T} \left\|\mathbf{x}_i^j - \bar{\mathbf{x}}_i\right\|_{\mathbf{Q}_i}^2 + \left\|\mathbf{u}_i^j - \bar{\mathbf{u}}_i\right\|_{\mathbf{R}_i}^2 \tag{103}$$

and $\mathbf{x}_i^j$ and $\mathbf{u}_i^j$ are the states and inputs of agent i predicted j steps from t, respectively. $\bar{\mathbf{x}}_i$ and $\bar{\mathbf{u}}_i$ are the desired values of states and inputs of the i-th agent. The parameter T is the prediction horizon, and $\mathbf{Q}_i$ and $\mathbf{R}_i$ are suitable positive definite matrices, which define the local cost function. $\mathbf{U}_i$ is the vector of predicted inputs along the prediction horizon. Finally, the convex set $\mathcal{K}_i$ includes the local constraints, and is defined as

$$\mathcal{K}_i = \left\{\mathbf{U}_i \,|\, \mathbf{x}_i^{j+1} = \mathbf{A}_i\mathbf{x}_i^j + \mathbf{B}_i\mathbf{u}_i^j \mathbf{x}_i^j \in \mathcal{X}_i,\ \mathbf{u}_i^j \in \mathcal{U}_i\right\}. \tag{104}$$

The defined DMPC problem can be considered as a particular form of problem (P) with a linear coupling constraint and quadratic objective function. In particular, we are interested in controlling the multi-tank system depicted in Figure 10. As depicted in Figure 10, q^i is the input flow and h_1^i and h_2^i are the water levels for system i, which are

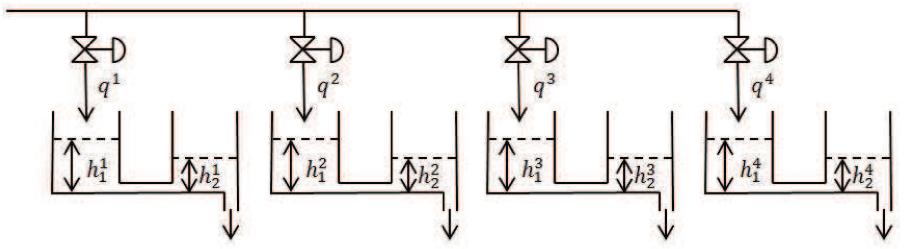


Figure 10 – The multi-tank system (WANG, Z.; ONG, C.-J., 2018).

considered as the states of the system. Accordingly, the system dynamics are obtained as follows:

$$\mathrm{x}_i(t+1) = \begin{pmatrix} 0.8750 & 0.1250 \\ 0.1250 & 0.8047 \end{pmatrix} \mathrm{x}_i(t) + \begin{pmatrix} 0.3 \\ 0 \end{pmatrix} u^i(t) \tag{105}$$

$$\mathcal{X}_i = \left\{ \mathrm{x}_i \in \mathcal{R}^2 | 0 \le \mathrm{x}_i^1 \le 2,\ 0 \le \mathrm{x}_i^2 \le 1.28 \right\} \tag{106}$$

$$\mathcal{U}_i = \left\{ u_i \in \mathcal{R}^1 | 0 \le u_i \le 0.6 \right\} \tag{107}$$

The control objective is to regulate the liquid of each tank to the global time-varying desired given reference while satisfying the global coupling constraint on the total input flow rate, which is defined as $\sum_{i=1}^{N} q^i \le 8.5$. The system was run for 500 seconds and the corresponding optimization problem was solved with the RH-ADMM. In order to assess the RH-ADMM algorithm on the dynamic and real-time implementation, the worst-execution time is measured with respect to prediction horizon T, for different values of the relaxation parameter α and applying the adaptive penalty parameter selection strategy. Moreover, the results were compared with the original H-ADMM.

The results are depicted in Figure 11. As it can be seen in the figure and explained in the last section, increasing the relaxation parameter α to its maximum value will improve the convergence time of the RH-ADMM algorithm. It is worth mentioning that it will be relatively simple to tune the RH-ADMM algorithm for any problem at hand. The simulation results of the closed-loop system have been provided in Figures 12-13. Fig. 12 depicts the tanks levels. As it can be seen in the figure, the levels of all tanks are successfully regulated on the desired value. The input flows of all tanks and the total input flow are shown in Figure 13. Notice that the constraint on the total input flow is satisfied by the solution that the RH-ADMM computes.

4.4.3 Economic Power Dispatching Problem

The IEEE 118 Bus System has 54 generator units and total power demand equal to 4600 MW (XIA et al., 2018). Each generator has a cost function, which is denoted by $g_i(p_i)$, where p_i is the output power of i-th generator unit. It is assumed that the cost functions are convex quadratic and they have the following structure:

$$g_i(p_i) = a_i p_i^2 + b_i p_i + c_{ik} \tag{108}$$

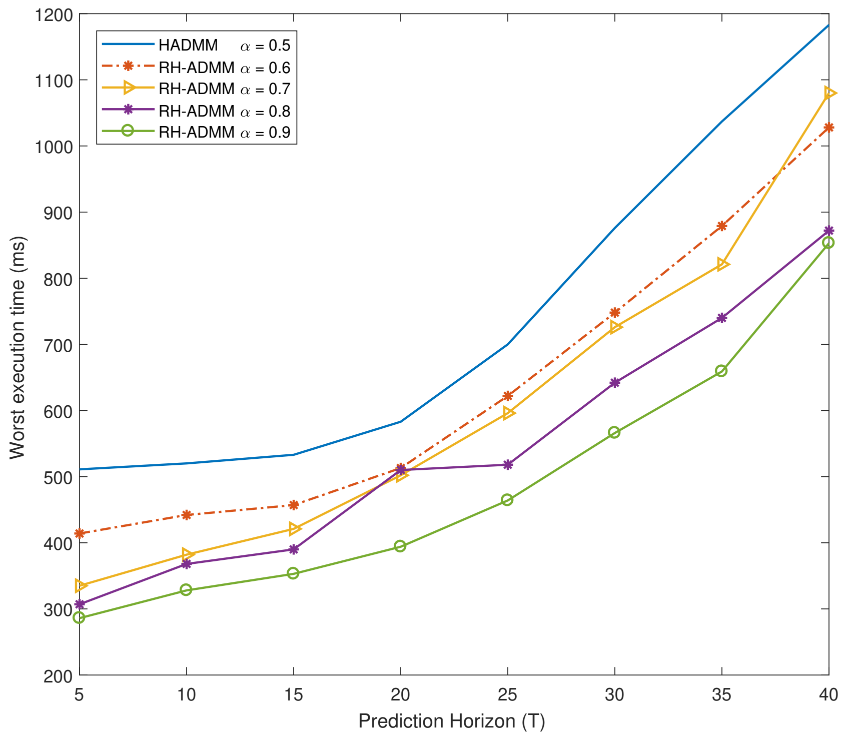


Figure 11 – Execution time of RH-ADMM for the DMPC application

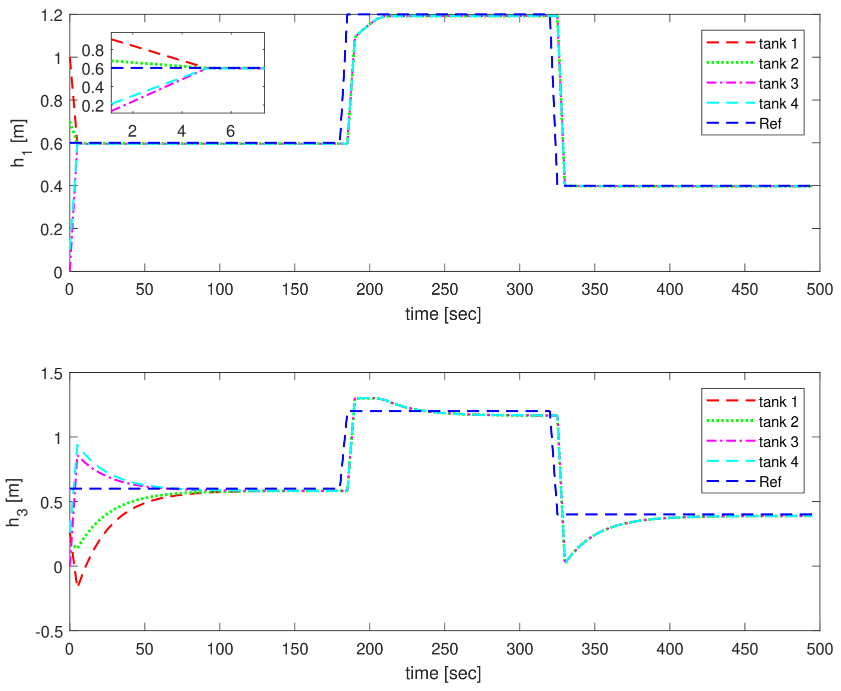


Figure 12 – Level of the tanks.

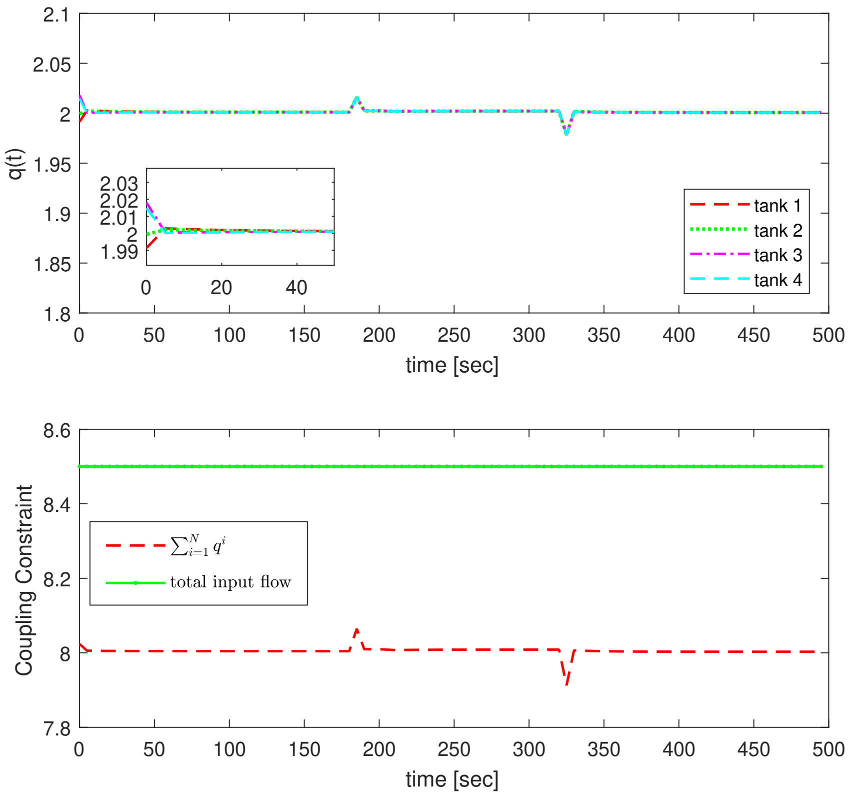


Figure 13 – Input flow of each agent and total input flow.

where a_i, b_i, and g_i are cost function parameters and they are constant. Thus the EPD problem with demand and production capacity constraints can be expressed as follows:

$$\min_{p_i} \quad \sum_{i=1}^{N} g_i(p_i) \tag{109}$$

$$\text{s.t} \quad \sum_{i=1}^{N} p_i = \sum_{i=1}^{N} pl_i = D \tag{110}$$

$$p_i \in \mathcal{P}_i \tag{111}$$

where D is total demand request, pl_i is the load of bus i and the convex set $\mathcal{P}_i = \left\{p_i : p^i_{min} \leq p_i \leq p^i_{max}\right\}$ indicates the power generation capacity of each unit in which p^i_{min} and p^i_{max} are the minimum and maximum production capacity of the generator i, respectively. Moreover, It is assumed that D satisfies $\sum_{i=1}^{N} p^i_{min} \leq D \leq \sum_{i=1}^{N} p^i_{max}$ constraint. In order to solve the EPD problem (109)-(111) in a fully decentralized fashion, we employ the duality framework that is explained above.

In order to verify the effectiveness of the proposed algorithm, the simulation is performed under three different scenarios. In the first scenario, it is assumed that the power demand is constant and no failure happens during the power grid operation. The second scenario includes time-varying demand and in the last case, the failure of some

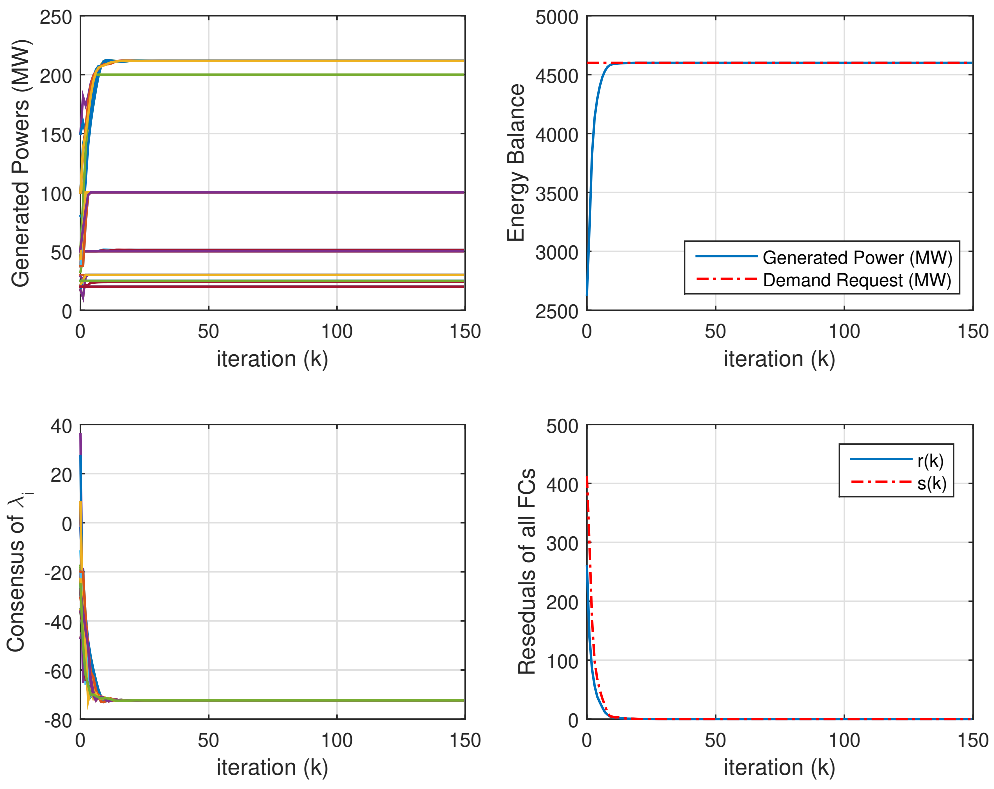


Figure 14 – Simulation results for the first scenario.

generation units is investigated.

Scenario I:

In this case the Algorithm 2 along with adaptive penalty parameter scheme (95) is considered. The relaxation parameter has been selected equal to $\alpha = 0.75$. As in the other scenarios, the number of LFCs is considered equal to 10 and the hypergraphs are generated randomly. Moreover, the adaptive penalty parameters τ and μ are selected equal to 2.5 and 10, respectively. The simulation results of this scenario are shown in Fig. 14. As it is clear from the figure, the algorithm converges after 18 iterations and the generated powers converge to their optimal values. Moreover, the dual variables $\boldsymbol{\lambda}$s successfully reach the consensus.

Scenario II:

In most situations, the demand request is not constant during the power grid operation. For this reason, in this scenario, Algorithm 2 is tested under time-varying demand. It is assumed that at iteration 50, the demand request increases to 6000 MW and at iteration 100 decreases to 3000 MW. Fig. 15, depicts the simulation results in this scenario. It is clear in the figure that the proposed algorithm can handle the time-varying

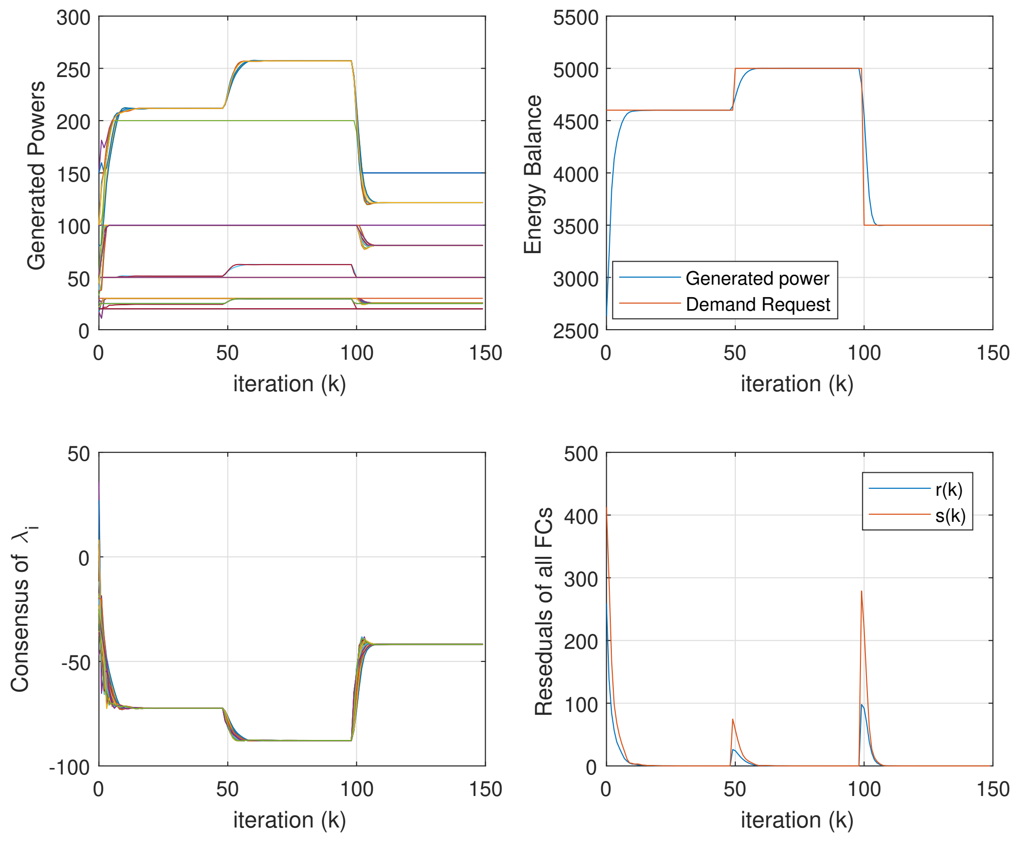


Figure 15 – Simulation results for the second scenario.

demand properly and without any modification.

4.4.3.1 Scenario III: Time-varying power demand and failure in generation units

in this case, the algorithm is tested when the failure of some power generation units occurs. Moreover, the demand request is still assumed to be time-varying. In this scenario, it is assumed that the power generators 1, 52 and 23 and then 9, 33 and 46 shut down at iteration 25 and 80, respectively. The simulation results of this scenario are provided in Fig. 16. According to Fig. 16, in the case of time-varying demand and failure of some power generators, the proposed method can provide a suitable solution.

To conclude, the simulation results presented in this section show that the proposed RH-ADMM is an efficient algorithm for solving distributed problems with coupling constraints and that it outperforms both the distributed gradient descent and the original hybrid ADMM. Moreover, we showed that by applying the proposed penalty selection scheme we are able to effectively speed up the convergence rate of the algorithm. Finally, although a centralized architecture obtains the best performance, we observed that a small number of local fusion centers yields good performances while at the same time increasing the resilience of the network to agents' failures.

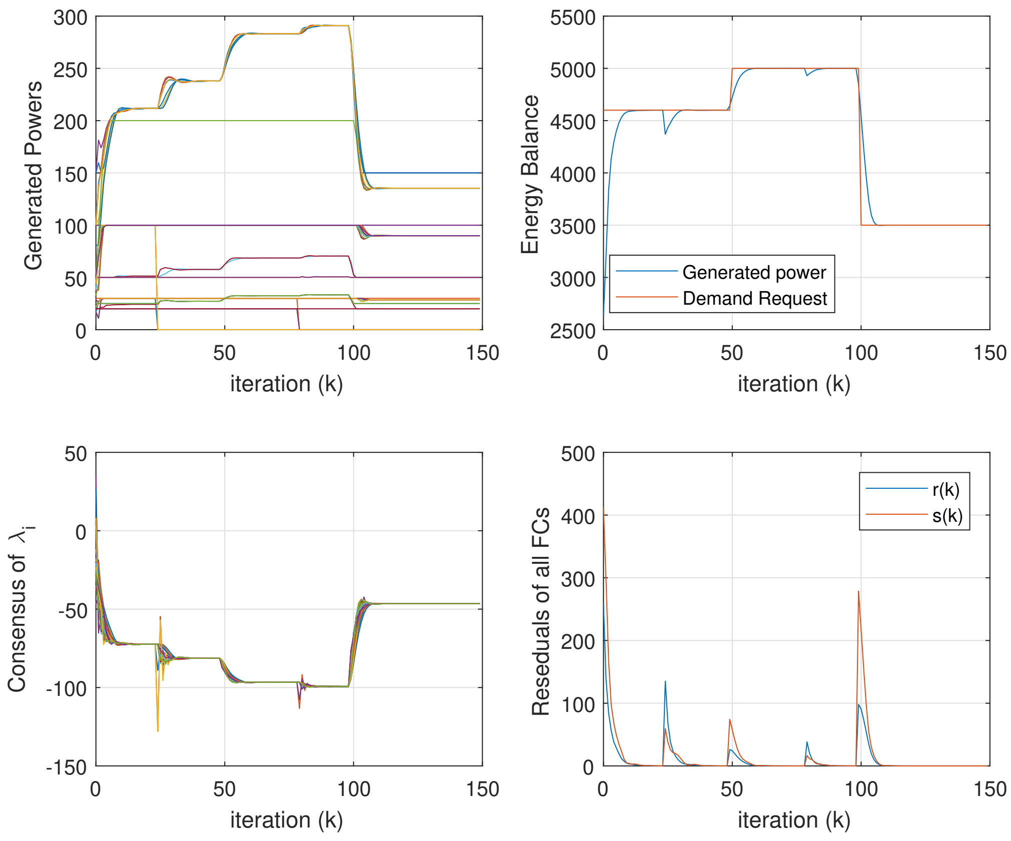


Figure 16 – Simulation results for the third scenario.

4.5 CONCLUSIONS

In this chapter, we proposed the RH-ADMM algorithm for solving distributed convex problems with global coupling inequality constraints. By using the advantages of the Krasnosel'skiĭ-Mann iteration along with the DR operator, a generalized version of the H-ADMM algorithm proposed in (MA, M.; NIKOLAKOPOULOS; GIANNAKIS, Georgios B., 2018) was developed. According to the results obtained in simulated experiments, the RH-ADMM algorithm leads to better practical convergence properties depending on the relaxation parameter α. Moreover, in order to further improve the convergence of the algorithm and to obtain a unified framework for choosing the penalty parameter ρ, we proposed an adaptive parameter selection scheme that can be implemented in a fully decentralized fashion.

Finally, we tested the proposed algorithm by applying it to two problems: Quadratic Programming and the control of a set of networked LTI systems with global coupling constraints using the distributed MPC law.

Future directions of our research will focus on improving the convergence rate, for example through Nesterov's acceleration scheme and preconditioning, and on extending the convergence guarantees in case of asynchronous updates and communication losses. Moreover, developing optimal frameworks to achieve the optimal α and ρ with guaranteeing the linear convergence may be considered.

5 DISTRIBUTED OUTER APPROXIMATION FOR SPARSE OPTIMIZATION

5.1 INTRODUCTION

This chapter introduces two distributed algorithms, namely the Distributed Primal Outer Approximation (`DiPOA`) (OLAMA; CAMPONOGARA; MENDES, 2021) and the Distributed Hybrid Outer Approximation (`DiHOA`) (OLAMA; CAMPONOGARA; KRONQVIST, 2022), which are designed to tackle separable structured SCO problems. The development of the `DiPOA` algorithm involves the integration of the proposed `RH-ADMM` algorithm (as described in Chapter 4) into the Outer Approximation (OA) algorithm. This integration is carried out in a manner that allows for distributed handling of the NLP subproblems. Essentially, the `RH-ADMM` algorithm serves as a distributed numerical engine within the `DiPOA` algorithm, thereby enabling it to leverage the multi-core architecture of modern processors for faster numerical computations. One of the primary motivations for solving the NLP problem in a distributed manner is that, in practical applications, a significant portion of the solution time is often consumed by solving these subproblems. For instance, in Table 1 of (KRONQVIST; BERNAL; GROSSMANN, Ignacio E., 2020), it is evident that more than 150 seconds are required by OA to solve NLP problems when solving a convex MINLP of moderate size. Additionally, for inherently distributed problems where the data is dispersed over a large computational network, solving a single NLP in a traditional centralized manner can be challenging or even impossible. In such scenarios, a distributed algorithm can provide significant computational benefits.

Although `DiPOA` solves the SCO problem in a distributed manner, the key components of the algorithm are built upon a multiple-tree OA algorithm, which involves constructing a BnB tree from scratch at every iteration. Consequently, a significant portion of the total solution time of `DiPOA` is typically consumed by solving MIP subproblems. To address these limitations, the `DiHOA` algorithm is developed. Unlike `DiPOA`, `DiHOA` gradually constructs a single BnB tree to avoid the need for constructing and solving multiple similar MILP problems from scratch. The `DiHOA` algorithm aims to solve a given SCO problem by dynamically updating the MIP subproblem based on the concept of *lazy constraints*. This approach is inspired by the LP/NLP-BnB method proposed by (QUESADA; GROSSMANN, Ignacio E, 1992). At the beginning of the algorithm, `DiHOA` employs the multiple-tree search strategy. When an event is triggered, `DiHOA` switches to a single-tree search strategy that builds a BnB tree by introducing multiple cuts initially to the root node to improve the initial formulation. The single BnB tree then tightens the integer relaxations by dynamically introducing more linear approximations (cuts) to the MIP problem. The multiple-tree search strategy is only applied in the initial iterations since the MIP problems are usually easier to solve, and the nonlinear constraints are only roughly represented through a few constraints. Furthermore, starting the BnB search with a tighter approximation of the nonlinear constraints results in a smaller BnB tree since a smaller infeasible region of the continuously relaxed search space is explored. Without these initial cuts, the nonlinear constraints would be completely ignored until an integer solution is found, and the nonlinear constraints would be poorly approximated until a few integer solutions are explored in the BnB tree.

Additionally, we introduce two algorithmic enhancements for `DiPOA` and `DiHOA` in the form of a specialized feasibility pump and an event-triggered second-order cut generation methods. These methods are designed to control both the quality and the number of cutting planes that approximate nonlinear functions. The main contributions of this chapter are summarized as follows:

- Equivalent distributed MINLP models for large-scale SCO problems. This is achieved by utilizing the hybrid architecture for the CN architecture and consensus optimization concepts.
- Algorithms `DiPOA` and `DiHOA` designed to solve SCO problems using a multiple-tree and a single-tree MINLP approach, respectively.
- A specialized distributed feasibility pump method to accelerate the convergence of the algorithms.
- A practical distributed method for the `RH-ADMM` algorithm to detect problem infeasibility before performing the main computational steps.
- A distributed second-order cut and an event-triggered cut generation scheme to improve performance and computational efficiency.
- A performance analysis of `DiPOA` and `DiHOA` regarding their applications to solve Distributed Sparse Logistic Regression (DSLogR), Distributed Sparse Linear Regression (DSLinR), and Distributed Sparse Quadratically Constrained Quadratic Programming (DSQCQP) problems over a computational network.

The organization of this chapter is as follows: In Section 5.2, we present the formulation of the SCO problem and modeling techniques to reformulate it into a suitable MINLP formulation. Section 5.3 introduces the two main subproblems used by both the `DiPOA` and `DiHOA` algorithms. Sections 5.4 and 5.5 present the `DiPOA` and `DiHOA` algorithms, respectively. In Section 5.6, we present the special feasibility pump method used to warm-start the algorithms. Finally, Section 5.7 introduces the practical infeasibility detection, and Section 5.8 provides the numerical experiments.

5.2 DISTRIBUTED SPARSE CONVEX OPTIMIZATION

In this chapter, we consider the SCO problem as a mathematical programming problem that consists of finding the κ-sparse optimal solution of a convex optimization

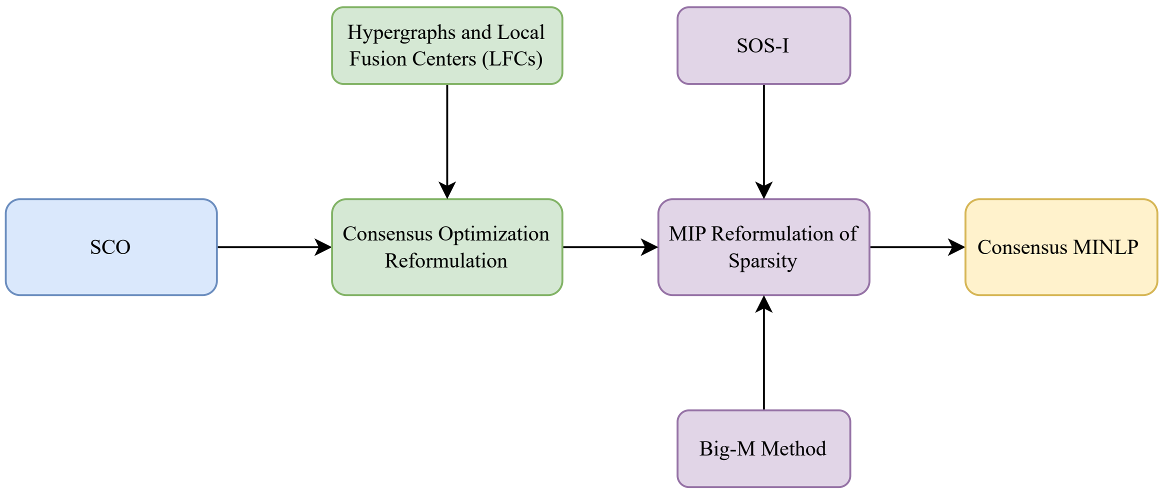

Hypergraphs and Local Fusion Centers (LFCs)
SOS-I
SCO
Consensus Optimization Reformulation
MIP Reformulation of Sparsity
Consensus MINLP
Big-M Method

Figure 17 – Problem reformulations performed by SCOT

problem with a separable structure defined as,

$$\begin{aligned} z_{\texttt{SCO}} = \min_{\mathbf{x}\in\mathbb{R}^n} & \sum_{i=1}^{N} f_i(\mathbf{x}) \\ \text{subject to} \quad & \mathbf{x} \in \mathcal{L} \cap \mathcal{N}_{\mathrm{g}} \cap \mathcal{C}_{\kappa} \end{aligned} \tag{112}$$

where

$$\mathcal{L} = \{x \in \mathbb{R}^n : \mathbf{A}\mathbf{x} - \mathbf{b} \leq 0\} \tag{113}$$

$$\mathcal{N}_{\mathrm{g}} = \{x \in \mathbb{R}^n : g_h(x) \leq 0, \forall h = 1, \ldots, m_2\} \tag{114}$$

$$\mathcal{C}_{\kappa} = \{\mathbf{x} \in \mathbb{R}^n : \|\mathbf{x}\|_0 \leq \kappa\}. \tag{115}$$

Here N is the number of nodes of the computation network, $\mathbf{x} \in \mathbb{R}^n$ is the vector of decision variables, and $f_i : \mathbb{R}^n \to \mathbb{R}$ is a convex function assumed to be continuously differentiable and only known by node i, for all $i \in \{1, \ldots, N\}$. The set $\mathcal{L}$ and $\mathcal{N}_{\mathrm{g}}$ are closed convex sets with no empty interior which represents the general linear and nonlinear constraints, respectively in which $g_h : \mathbb{R}^n \to \mathbb{R}$ is a continuously differentiable convex function and the matrix $\mathbf{A} \in \mathbb{R}^{m_1 \times n}$ and the vector $\mathbf{b} \in \mathbb{R}^{m_1}$ define the set of given linear constraints assumed to be known by all nodes. Various modeling techniques are used by SCOT to find an equivalent problem of (112) for which distributed algorithms can be developed. The techques used by SCOT are depicted in Figure 17 which will be discussed below.

First, SCOT transforms problem (112) into a consensus optimization problem and then multiple modeling techniques are used to handle the sparsity constraint.

5.2.1 Consensus Optimization Modeling

In problem (112), the decision variables are shared among the nodes, with each node having access only to information needed to construct its own objective function. Local problem data is kept private from other nodes. The distributed setting in

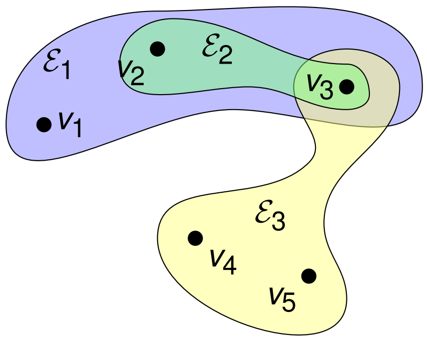


Figure 18 – Schematic of a hypergraph with 5 nodes and 3 hyperedges

(112) is a common approach used in many important learning and control applications. For example, in distributed machine learning, a practical solution to handling large volumes of data involves distributing the data across a network (NOTARSTEFANO; NOTARNICOLA; CAMISA, et al., 2019; NEDIĆ; LIU, 2018). This approach enables significant reductions in memory requirements for computation while maintaining the same unknown model parameters. To decompose (112), we adhere to the concept of hypergraphs introduced in Chapter 4, which is a generalization of a regular graph in which an edge can join an arbitrary number of vertices (see figure 18). We consider a hypergraph $\mathcal{H} = (\mathcal{V},\mathcal{E})$ defined as follows: $\mathcal{V} = \{1,2,...,N\}$ is the set of nodes such that node i decides upon the values of vector variable $\mathbf{x}_i$; $\mathcal{E} = \{\mathcal{E}_k \subset \mathcal{V} : k = 1,\dots,K\}$ is the set of hyperedges, where a hyperedge $\mathcal{E}_k$ connects all nodes $i \in \mathcal{E}_k$ and K is the number of hyperedges. Now we introduce the concept of path in a hypergraph: a path $p(i,j) = \langle \mathcal{E}'_1,...,\mathcal{E}'_k \rangle$ connects nodes i and j if $i \in \mathcal{E}'_1$, $j \in \mathcal{E}'_k$, and $\mathcal{E}'_l \cap \mathcal{E}'_{l+1} \neq \emptyset$ for $l = 1,\dots,k-1$, and $\mathcal{E}'_l \in \mathcal{E}$ for all l. Put another way, through the hyperedges, a path $p(i,j)$ establishes a communication channel between nodes i and j.

We assume that the hypergraph $\mathcal{H}$ is connected, meaning that for all $(i,j) \in \mathcal{V}$ there exists a path $p(i,j)$ connecting i and j. By using the hypergraph structure, the equivalent formulation for (112) is obtained as follows,

$$\begin{aligned} z_{\text{SCO}} = \min_{\substack{\mathbf{x}_1,\dots,\mathbf{x}_N \\ \mathbf{y}_1,\dots,\mathbf{y}_K}} \; & \sum_{i=1}^{N} f_i(\mathbf{x}_i) \\ \text{subject to } & \mathbf{x}_i = \mathbf{y}_j,\ \forall i \in \mathcal{E}_j,\ \forall \mathcal{E}_j \in \mathcal{E} \\ & \mathbf{x}_i \in \mathcal{L} \cap \mathcal{N}_{\text{g}} \\ & \mathbf{y}_j \in \mathcal{C}_K,\ \forall j = 1,\dots,K \end{aligned} \tag{116}$$

where $\mathbf{x}_i \in \mathbb{R}^n$ is the vectors of decision variables associated with the i-th node and $\mathbf{y}_j \in \mathbb{R}^n$ is the vector of auxiliary decision variables associated with the j-th LFC, which are represented by the hyperedges.

5.2.2 MIP Reformulation of $\mathcal{C}_\kappa$

In this section, we introduce three methods utilized by `SCOT` to represent the non-convex set $\mathcal{C}_\kappa$ in problem (112): the *Big-M* method, the *Specially Ordered Set of Type I* (SOS-1) method, and a *hybrid* approach.

5.2.2.1 Big-M Method

The Big-M method is arguably the simplest technique for modeling the sparsity constraint. This method incorporates a binary variable and an estimated upper bound into the model for each continuous variable appearing in the sparsity constraint. By using the Big-M method, we can represent the sparsity constraints by the following set of mixed-integer linear inequalities,

$$\begin{aligned} &-\mathtt{M}_j\delta_{jk} \leq y_{jk} \leq \mathtt{M}_j\delta_{jk},\ \forall k = 1,\ldots,n,\ \forall j = 1,\ldots,K \\ &\sum_{k=1}^{n} \delta_{jk} \leq \kappa,\ \forall j = 1,\ldots,K \\ &\delta_j \in \{0,1\}^n,\ \forall j = 1,\ldots,K \end{aligned} \tag{117}$$

where y_{jk} is the k-th element of $\mathbf{y}_j$, δ_j, $\forall j = 1,\ldots,K$, is a vector of binary variables whose k-th element is denoted by δ_{jk}, and $\mathtt{M}_j$ is a constant assumed to be a valid upper bound for $\left\|\mathbf{y}_j\right\|_\infty$. In this case, if $\delta_{jk} = 0$ then $y_{jk} = 0$ and $y_{jk} = 1$ otherwise. Thus, inequalities (117) impose the maximum κ number of nonzero variables in $\mathbf{y}_j$. Therefore for any $\mathbf{y}_j$ and δ_j, the set $\mathcal{C}_\kappa$ can be replaced by the following mixed-integer set

$$\mathcal{C}_\kappa^{\mathtt{MIP}} = \{(\mathbf{y},\delta) \in \mathbb{R}^n \times \mathbb{B}^n : \mathbf{A}_1\mathbf{y} + \mathbf{A}_2\delta \leq \mathbf{b}_1\}. \tag{118}$$

where $\mathbf{A}_1$ and $\mathbf{A}_2$ are appropriate matrices and $\mathbb{B} \subset \mathbb{Z}$ is the set of binary numbers.

5.2.2.2 Specially Ordered Set of Type I (SOS-1) Method

This section discusses the sparsity constraint reformulation using the SOS-1 constraint. Any feasible solution to problem (116) satisfies the following complementary constraints,

$$(1-\delta_{jk})y_{jk} = 0,\ \forall j = 1,\ldots,K,\ \forall k = 1,\ldots,n \tag{119}$$

which is equivalent to the first constraint in (117). In order for constraint (119) to be satisfied, either $(1–\delta_{jk})$ or y_{jk} must be zero. Such constraints can be modeled via integer optimization software using Specially Ordered Sets of Type I (SOS-1) (BERTSIMAS; WEISMANTEL, 2005) as follows,

$$\left(y_{jk}, 1-\delta_{jk}\right)\ :\ \text{SOS-1},\ \forall j = 1,\ldots,K,\ \forall k = 1,\ldots,n. \tag{120}$$

Therefore, the sparsity constraint can be modeled by the following constraints

$$\begin{aligned}
&\left(y_{jk}, 1-\delta_{jk}\right) \ : \text{SOS-1}, \ \forall j = 1,\ldots,K, \ \forall k = 1,\ldots,n.\\
&\sum_{k=1}^{n} \delta_{jk} \leq \kappa, \ \forall j = 1,\ldots,K \\
&\delta_{jk} \in \{0,1\}, \ \forall j = 1,\ldots,K, \ \forall k = 1,\ldots,n
\end{aligned} \tag{121}$$

In summary, selecting a suitable value for $\mathtt{M}_j$ is crucial in the big-M formulation, as a value that is too small can lead to the exclusion of valid solutions, whereas an excessively large value can result in numerical difficulties. Hence, a good choice of $\mathtt{M}_j$ affects the strength of the formulation, being critical for MIP algorithms to obtain high-quality bounds. However, in machine learning, power systems, and control applications, the big-M value $\mathtt{M}_j$ can typically be computed from data in statistical learning tasks (BERTSIMAS; PAUPHILET; VAN PARYS, 2021) and from the physical bounds in control applications (AGUILERA et al., 2017). Alternatively, SOS-1 constraints do not depend on a problem-specific constant and thus avoid these issues. In general, when a relatively small $\mathtt{M}_j$ is required, big-M modeling may be appropriate, while the SOS-1 approach may be preferable in other situations. The `SCOT` preprocessing module provides the functionality to determine the most suitable method based on the application and problem data.

5.2.3 Mixed Integer Nonlinear Optimization Reformulation

This section covers the main problem formulation addressed by the `DiPOA` and `DiHOA` algorithms. By utilizing Big-M and SOS-1 constraints to model sparsity constraints, the primary optimization problem that `SCOT` strives to solve can be expressed as follows:

$$\begin{aligned}
z_{\mathtt{SCO}} = \min_{\substack{\gamma \\ \mathbf{x}_1,\ldots,\mathbf{x}_N \\ \mathbf{y}_1,\ldots,\mathbf{y}_K}} \ & \gamma \\
\text{subject to } & \mathbf{x}_i \in \mathcal{L} \cap \mathcal{N}, \ \forall i = 1,\ldots,N \\
& \mathbf{x}_i = \mathbf{y}_j, \ \forall i \in \mathcal{E}_j, \ \forall \mathcal{E}_j \in \mathcal{E} \\
& (\mathbf{y}_j, \delta_j) \in \mathcal{C}_{\kappa}^{\mathtt{MIP}}, \ \forall j = 1,\ldots,K \\
& \left(\mathbf{y}_j, 1-\delta_j\right) \ : \text{SOS-1}, \ \forall j = 1,\ldots,K
\end{aligned} \tag{122}$$

where the equivalent epigraph reformulation is used and $\mathcal{N} = \mathcal{N}_{\mathtt{g}} \cap \mathcal{N}_{\mathtt{f}}$ with $\mathcal{N}_{\mathtt{f}} = \{\sum_{i=1}^{N} f_i(\mathbf{x}_i) - \gamma \leq 0\}$, is the set of all nonlinear constraints. The advantage of using both SOS-1 and Big-M constraints is that it might be possible to determine better $\mathtt{M}_j$ coefficients than the MIP solver is able to derive. In that way, `SCOT` provides more information to the MIP solver by including these constraint. Problem (122) is an MINLP problem with a separable structure, where the objective function is linear and $\gamma \in \mathbb{R}$ is an auxiliary variable. In the following sections, we introduce a distributed formulation and algorithm that SCOT implements to solve problem (122).

5.3 PRIMAL AND DUAL PROBLEMS

In this section, we will introduce the primal and dual problems of SCOT and discuss two algorithms that SCOT utilizes. SCOT is a decomposition-based MINLP solver, which breaks down problem (122) into two main sub-problems: the *primal* problem and the *dual* problem. As described in (KRONQVIST; BERNAL; GROSSMANN, Ignacio E., 2020; LUNDELL; KRONQVIST, 2019), we refer to the optimal solution and objective value of the primal problem as the *primal solution* and *primal bound*, respectively. Likewise, we use the terms *dual solution* and *dual bound* for the dual problem.

The primal solution of problem (122) is expected to satisfy all linear, nonlinear, and consensus constraints up to a given tolerance. The best-known primal solution obtained by either DiPOA or DiHOA algorithms is referred to as the *incumbent solution*, and its objective value represents the *best primal bound*. The incumbent solution is updated when SCOT algorithms discover a primal solution with a lower objective value. Although the primal problem in the standard OA algorithm is a convex NLP problem, the primal problem in SCOT is a distributed convex NLP problem due to the distributed nature of problem (122). We will discuss this distributed primal problem later.

A dual solution is a solution point whose objective value serves as a valid lower bound for the optimal solution of problem (112), but does not necessarily satisfy all nonlinear constraints. Like the standard OA algorithm, SCOT obtains dual solutions by solving relaxed problems that approximate the nonlinear constraints with outer approximations. Depending on the type of outer approximations used, the dual problem can be a MILP, MIQP, or MIQCL(Q)P problem. The dual bound represents the best possible objective value of the dual problem. The primal and dual sub-problems are then iteratively solved by proper algorithms. The SCOT algorithms are distinguished depending on how the sub-problems are constructed, solved, and coordinated. Regardless of the solution algorithms adopted by SCOT, the dual and primal sub-problems are two primary components of DiPOA and DiHOA.

5.3.1 Lifted Formulation

By default, SCOT focuses on solving problem (122), which enforces both Big-M and SOS-1 constraints and utilizes epigraph reformulation. The separability of the nonlinear functions in $\mathcal{N}_{\mathtt{f}}$ in problem (122) enables SCOT to use an alternative formulation known as the *lifted formulation* (KRONQVIST; LUNDELL; WESTERLUND, 2018). SCOT employs the lifted formulation for f_i resulting in tighter outer approximations when approximating nonlinear functions (KRONQVIST; LUNDELL; WESTERLUND, 2018;

HIJAZI; BONAMI; OUOROU, 2014; TAWARMALANI; SAHINIDIS, 2005). In this case, the problem (122) is rewritten as,

$$\begin{aligned} z_{\mathtt{SCO}} = \min_{\substack{\mathbf{x}_1,\dots,\mathbf{x}_N\\ \mathbf{y}_1,\dots,\mathbf{y}_K\\ \gamma_1,\dots,\gamma_N}} &\sum_{i=1}^{N} \gamma_i \\ \text{subject to } & \mathbf{x}_i \in \mathcal{L} \cap \mathcal{N}_i,\ \forall i = 1,\dots,N \\ & \mathbf{x}_i = \mathbf{y}_j,\ \forall i \in \mathcal{E}_j,\ \forall \mathcal{E}_j \in \mathcal{E} \\ & (\mathbf{y}_j, \delta_j) \in \mathcal{C}_K^{\mathtt{MIP}},\ \forall j = 1,\dots,K \\ & \left(\mathbf{y}_j, 1-\delta_j\right) \ :\ \text{SOS-1},\ \forall j = 1,\dots,K \end{aligned} \tag{123}$$

where $\mathcal{N}_i = \mathcal{N}_{\mathtt{g}} \cap \mathcal{N}_{\mathtt{f_i}}$ with $\mathcal{N}_{\mathtt{f_i}} = \{\mathbf{x} \in \mathbb{R}^n : f_i(\mathbf{x}) - \gamma_i \leq 0\}$ and $\gamma_i \in \mathbb{R}$, $i = \{1,\dots,N\}$, are new auxiliary decision variables.

5.3.2 Dual Problem

In this section, we derive the dual subproblem of the MINLP problem (123). Given a set of k feasible points of problem (123) from the perspective of the i-th agent $\mathcal{S}_i^k = \{\mathbf{x}_i^q : \forall q = 1,\dots,k\}$, an outer approximation of the set $\mathcal{N}_i$ can be constructed as,

$$\widetilde{\mathcal{N}}_i^k = \{\mathbf{x}_i \in \mathbb{R}^n : \widetilde{g}_{\mathbf{x}_i^q}(\mathbf{x}_i) \leq 0, \widetilde{f}_{i\mathbf{x}_i^q}(\mathbf{x}_i) - \gamma_i \leq 0, q = 1,\dots,k\} \tag{124}$$

where $\widetilde{g}_{\mathbf{x}_i^q}(\mathbf{x}_i)$ and $\widetilde{f}_{i\mathbf{x}_i^q}(\mathbf{x}_i)$ are outer approximations of $g(\mathbf{x}_i)$ and $f_i(\mathbf{x}_i^q)$ around $\mathbf{x}_i^q$, respectively. Therefore, for the i-th agent we observe that

$$\mathcal{N}_i \subseteq \widetilde{\mathcal{N}_i}^k \subseteq \widetilde{\mathcal{N}_i}^{k-1} \subseteq \cdots \subseteq \widetilde{\mathcal{N}_i}^1 \tag{125}$$

which follows from the convexity of $g(\mathbf{x}_i)$ and $f_i(\mathbf{x}_i^q)$. Considering the outer approximation set $\widetilde{\mathcal{N}}_i^k$ for each agent, we define the dual problem of (123), at iteration k, as the following mixed-integer optimization problem

$$\begin{aligned} z_{\mathtt{MIP}}^k = \min_{\substack{\mathbf{x}_1,\dots,\mathbf{x}_N\\ \mathbf{y}_1,\dots,\mathbf{y}_K\\ \gamma_1,\dots,\gamma_N}} &\sum_{i=1}^{N} \gamma_i \\ \text{subject to } & \mathbf{x}_i \in \mathcal{L} \cap \widetilde{\mathcal{N}}_i^k,\ \forall i = 1,\dots,N \\ & \mathbf{x}_i = \mathbf{y}_j,\ \forall i \in \mathcal{E}_j,\ \mathcal{E}_j \in \mathcal{E} \\ & (\mathbf{y}_j, \delta_j) \in \mathcal{C}_K^{\mathtt{MIP}},\ \forall j = 1,\dots,K \\ & \left(\mathbf{y}_j, 1-\delta_j\right) \ :\ \text{SOS-1},\ \forall j = 1,\dots,K \end{aligned} \tag{126}$$

Hence,

$$z_{\mathtt{SOC}} \geq z_{\mathtt{MIP^k}} \geq z_{\mathtt{MIP^{k-1}}} \cdots \geq z_{\mathtt{MIP^1}}. \tag{127}$$

The quality of outer approximations generated by the set $\widetilde{\mathcal{N}}_i^k$ directly impacts the convergence of the `DiPOA` and `DiHOA` algorithms. Hence, `SCOT` provides first- and second-order outer approximations and an event-triggered scheme that controls the effectiveness of the approximations. According to first-order Taylor series and convexity of $f_i(\mathrm{x}_i)$ and $g(\mathrm{x}_i)$ functions, we can express $\widetilde{g}_{\mathrm{x}_i^q}(\mathrm{x}_i)$ and $\widetilde{f}_{i\mathrm{x}_i^q}(\mathrm{x}_i)$ as,

$$\begin{aligned}\widetilde{f}_{i\mathrm{x}_i^q}(\mathrm{x}_i) &= f_i(\mathrm{x}_i^q) + \nabla f_i(\mathrm{x}_i^q)^T(\mathrm{x}_i - \mathrm{x}_i^q)\\ \widetilde{g}_{\mathrm{x}_i^q}(\mathrm{x}_i) &= g(\mathrm{x}_i^q) + \nabla g(\mathrm{x}_i^q)^T(\mathrm{x}_i - \mathrm{x}_i^q)\end{aligned} \tag{128}$$

which are polyhedral approximations. In case the nonlinear functions, $f_i(\mathrm{x}_i)$ and $g(\mathrm{x}_i)$, are *strongly* convex functions, `SCOT` utilizes the following second-order outer approximation functions,

$$\begin{aligned}\widetilde{f}_{i\mathrm{x}_i^q}(\mathrm{x}_i) &= f_i(\mathrm{x}_i^q) + \nabla f_i(\mathrm{x}_i^q)^T(\mathrm{x}_i - \mathrm{x}_i^q) + \frac{m_i^f}{2}\left\|(\mathrm{x}_i - \mathrm{x}_i^q)\right\|_2^2\\ \widetilde{g}_{\mathrm{x}_i^q}(\mathrm{x}_i) &= g(\mathrm{x}_i^q) + \nabla g(\mathrm{x}_i^q)^T(\mathrm{x}_i - \mathrm{x}_i^q) + \frac{m^g}{2}\left\|(\mathrm{x}_i - \mathrm{x}_i^q)\right\|_2^2\end{aligned} \tag{129}$$

where $m_i^f > 0$ and $m^g > 0$ are strong convexity constants. With strong convexity constants, it is clear that the cut given by (129) is stronger than the cut given by (128). However, the second-order approximations (129) tend to result in more challenging sub-problems in BnB. Therefore, there can still be a computational advantage of the linear cuts.

Remark 9. *For general strongly convex functions, m_i^q is not obtained easily. However, in some practical problems found in statistical learning and control, the computation of m_i^q is feasible. For example, the objective function in sparse Model Predictive Control (s-MPC) problems (which is a subclass of the SCO problem) is typically a convex quadratic function. For convex quadratic functions, m_i^q is the smallest Eigenvalue of the Hessian matrix. In machine learning problems the objective function usually consists of a convex function and a strongly convex regularization term. In this case, m_i^q can be computed from the regularization term.*

5.3.3 Primal Problem

A crucial step to forming the outer approximation set is the computation of the set $\mathcal{S}_i^k$. To compute $\mathcal{S}_i^k$ similar to standard OA algorithm, we fix the local binary decision variables of problem (123), $\delta_j = \delta_j^q$, and solve the resulting distributed convex optimization problem. In this case, a distributed convex NLP problem is solved, and its optimal solution provides a primal solution to the original MINLP problem (123). Given the set of fixed feasible binary variables, then $\mathcal{S}_i^k$ can be computed by solving the following set of problems for $q = 1, \dots, k$

$$\begin{aligned} z^q_{\mathtt{Feasb}} = \min_{\substack{\mathrm{x}_1,\dots,\mathrm{x}_N \\ \mathrm{y}_1,\dots,\mathrm{y}_K \\ \gamma_1,\dots,\gamma_N}} \; & \sum_{i=1}^{N} \gamma_i \\ \text{subject to } & \mathrm{x}_i \in \mathcal{L} \cap \mathcal{N}, \; \forall i = 1,\dots,N \\ & \mathrm{x}_i = \mathrm{y}_j, \; \forall i \in \mathcal{E}_j, \; \forall \mathcal{E}_j \in \mathcal{E} \\ & (\mathrm{y}_j, \delta^q_j) \in \mathcal{C}^{\mathtt{MIP}}_K, \; \forall j = 1, \dots, K \end{aligned} \tag{130}$$

Since the solution of (130) is feasible and not necessarily optimal, $z^q_{\mathtt{Feasb}}$ provides an upper bound on $z_{\mathtt{SCO}}$ which reads as,

$$z^q_{\mathtt{Feasb}} \geq z_{\mathtt{SOC}} \geq z_{\mathtt{MIP^k}} \geq z_{\mathtt{MIP^{k-1}}} \cdots \geq z_{\mathtt{MIP^1}}. \tag{131}$$

Therefore, one can observe that

$$\mathrm{P}^k_{\mathrm{b}} \geq z_{\mathtt{SCO}} \geq \mathrm{D}^k_{\mathrm{b}} \tag{132}$$

where

$$\mathrm{P}^k_{\mathrm{b}} = \min_{q \in \{1,\dots k\}} z^q_{\mathtt{Feasb}} \tag{133}$$

$$\mathrm{D}^k_{\mathrm{b}} = z_{\mathtt{MIP^k}} \tag{134}$$

are the tightest primal bound and dual bound on $z_{\mathtt{SCO}}$ at step k. Problem (130) is a distributed convex NLP problem and its solution has the advantage of generating linearizations about points closer to the feasible region. Therefore, primal solutions and primal bounds are obtained by iteratively solving problem (130).

In the case that the primal problem is a centralized NLP, a feasible point that satisfies all linear and nonlinear constraints is considered to be the primal solution candidate. However, when the primal problem has to be solved distributedly, as in problem (130), some numerical considerations have to be taken into account. Particularly, in this case, in addition to all linear and nonlinear constraints, the consensus constraints have to be satisfied which is more challenging to deal with since all computational nodes have to agree on a consensus solution. In case the primal solution does not satisfy the consensus constraints within an acceptable numerical tolerance, poor outer approximations are generated and added to the dual problem. Therefore a larger number of iterations are required by the distributed NLP solver, especially when a large computational network is considered. Another challenge in solving the primal problem distributedly is the communication burden between the nodes of the network and the LFCs.

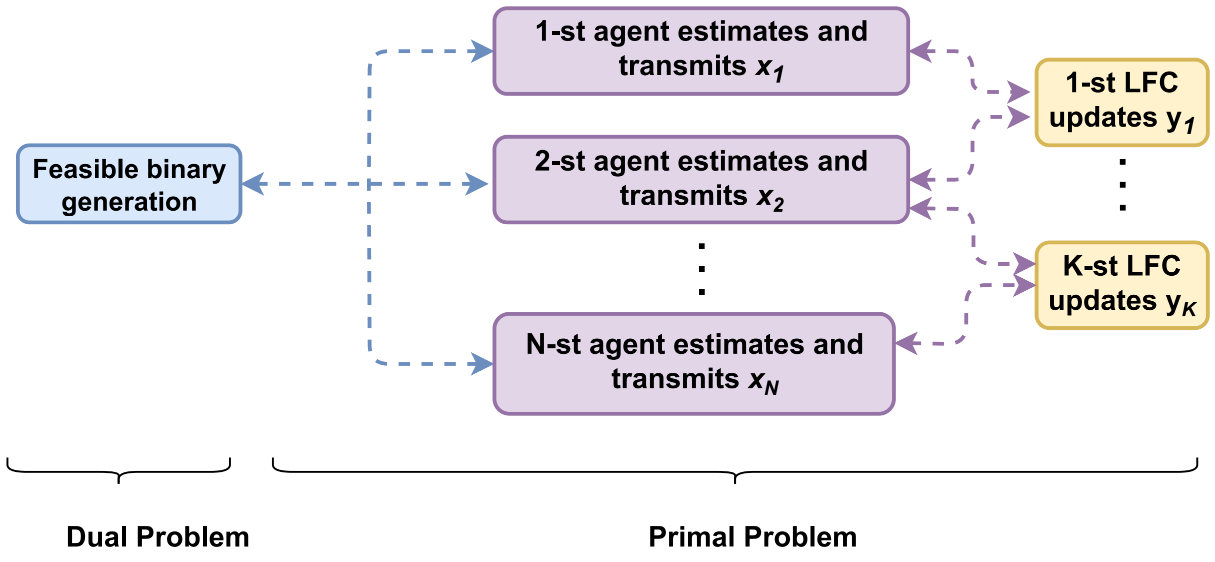

1-st agent estimates and transmits x_1
2-st agent estimates and transmits x_2
N-st agent estimates and transmits x_N
Feasible binary generation
1-st LFC updates y_1
K-st LFC updates y_K
Dual Problem
Primal Problem

Figure 19 – DiPOA distributed architecture

5.4 DISTRIBUTED PRIMAL OUTER APPROXIMATION

In this section, we present the `DiPOA` algorithm for solving problem (123), which is essentially the equivalent of the SCO problem (112). Moreover, various techniques and heuristics are introduced to enhance the efficiency of the algorithm. The `DiPOA` algorithm consists of multiple stages and follows a standard approach for implementing outer-approximation-based algorithms, which involves solving several individual instances of the dual (126) and primal (130) problems at each iteration. This approach requires solving a distributed convex optimization problem and a mixed-integer programming (MIP) problem at each iteration of the `DiPOA` algorithm. This implementation strategy offers a high degree of flexibility as it does not require extensive interaction with the MIP solver.

A basic implementation of `DiPOA` can simply construct the MIP dual problem and pass it to the MIP solver for an optimal solution. The obtained solution can then be read back into the main loop of the `DiPOA` algorithm, where outer approximations based on supporting hyperplanes or second-order approximations are generated and added to the dual problem before it is solved again. This process is repeated until a certain termination tolerance is reached, indicating that the final solution to problem (112) has been found. The distributed architecture of the `DiPOA` is depicted in Figure 19.

This approach is referred to as a *multi-tree* outer approximation because a new BnB tree is created at each invocation of the MIP solver. In the subsequent sections, we provide a detailed discussion of each step involved in the `DiPOA` algorithm.

5.4.1 The Primal Step

The primal step in DiPOA is primarily responsible for generating feasible points x_i^q around which the local nonlinear functions $f_i(\mathrm{x}_i)$ and $g(\mathrm{x}_i)$ are approximated using either the (128) or (129) equations. This step specifically handles the NLP portion of problem (123) and is composed of two steps, each of which is responsible for a specific computational task. To solve the primal problem (130), similar to what we proposed in Chapter 4, we decompose the main computational parts into two steps that distributedly communicate over the network. In the first step, each agent of the network solves a *local* convex NLP in parallel, in a fully decentralized fashion. In the next step, the local agents then *synchronously* communicate local solutions to LFCs where solutions of a series of unconstrained NLP problems are computed. In some practical cases, LFC level computations are related to computing the projection operator, which guarantees the problem feasibility (MA, M.; NIKOLAKOPOULOS; GIANNAKIS, Georgios B., 2018; OLAMA et al., 2019). As before, the computations in the LFC level are performed in parallel. In many real-world applications, due to the structure of the problem to be solved, the computations at the LFC level are translated into numerical linear algebra operations. In synthesis, the primal step aims to obtain the solutions to different nonlinear optimization problems[1] for which the computations are performed in parallel, using a fully decentralized approach.

5.4.2 The Dual Step

The dual step of the DiPOA algorithm corresponds to solving the dual mixed-integer programming (MIP) approximation problem (126). This problem is based on the outer approximation generated around x_i^q, $q \in \{1, \ldots, k\}$, which is provided by the primal step of the algorithm. Therefore at each iteration k the dual step iteratively enhances the approximation of nonlinear terms by tightening the linear (or quadratic) approximation of the nonlinear functions of problem (123). Thus, as the number of outer approximations increases, the dual bound generated by the dual problem (126) is improved as it can be seen in (127). This iterative improvement process allows the DiPOA algorithm to converge to the optimal solution of the original MINLP when a sufficiently large number of outer approximations are added to the dual problem.

Example 1. To demonstrate how DiPOA constructs the set of outer approximations $\widetilde{\mathcal{N}}$,

[1] The problems are *different* in the sense that the problem data are not necessarily the same.

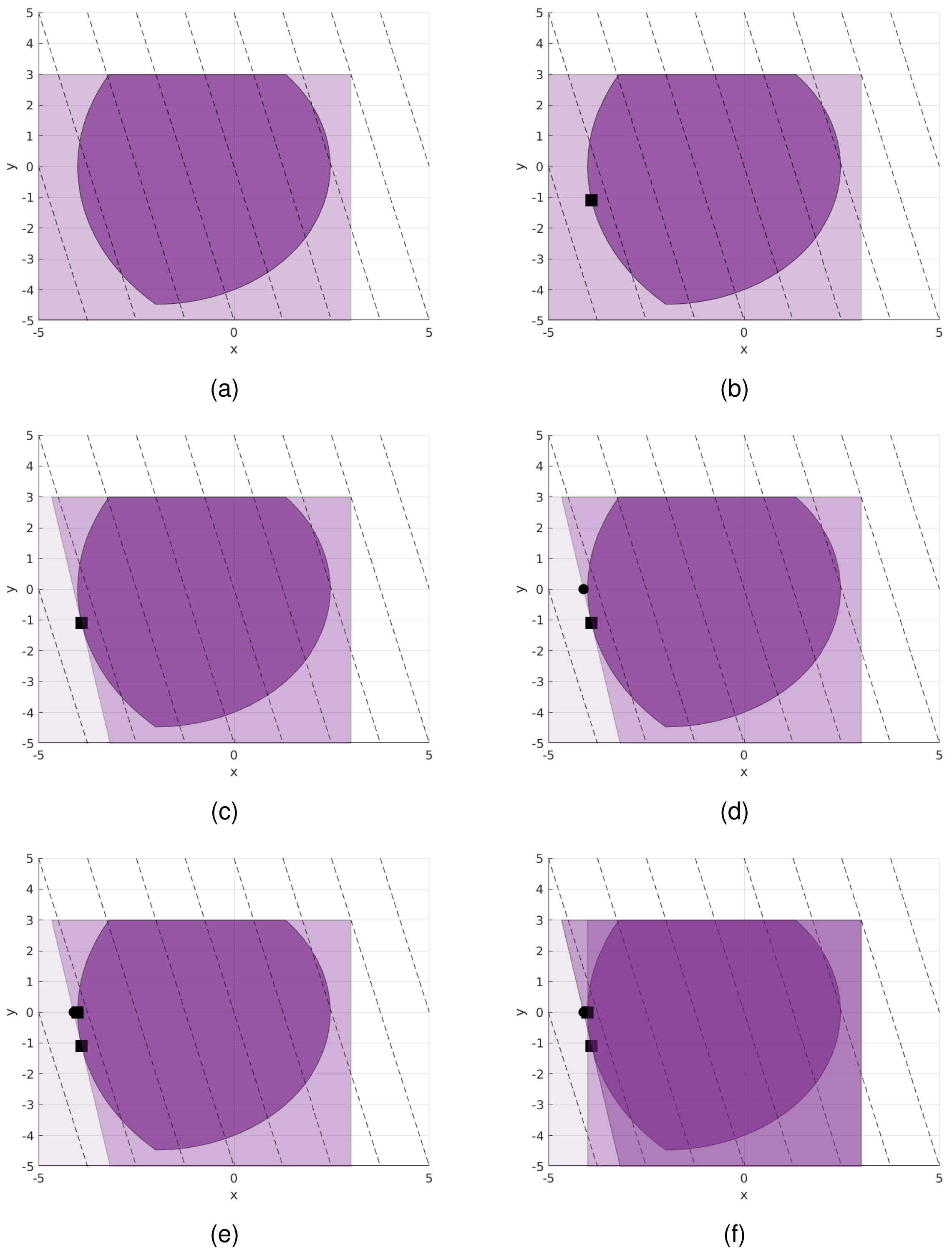


Figure 20 – First iterations of `DiPOA` Algorithms; the dark purple region represents the feasible region defined by the nonlinear constraints, the light purple areas represent the outer approximations, and the dashed lines show contours of the objective functions. The circular dots represent the solution of the dual problem, wand the squared dots represent the solution of the primal problem. the objective contours.

we present a numerical example using the following SOC problem:

$$\begin{aligned}\min_{\mathbf{x}} \quad & 8x+2y \\ \text{subject to} \quad & (x+2)^2+y^2-20 \leq 0 \\ & x^2+(-1-y)^2-36 \leq 0 \\ & (2-x)^2+y^2-36 \leq 0 \\ & -5 \leq x \leq 3 \\ & -5 \leq y \leq 3 \\ & \|(x,y)\|_0 \leq 1\end{aligned} \tag{135}$$

Figure 20 displays the initial iterations of the algorithm. To start, the dual problem (126) is constructed by disregarding the nonlinear constraints and solving the remaining mixed-integer linear programming (MILP) relaxation of the problem, illustrated in Figure 20a. Note that in this figure, we have included the nonlinear feasible region to provide a visual representation of how outer approximations are used to approximate this region. The binary variables obtained from the MILP relaxation are then utilized to construct and solve the first primal problem, as depicted in Figure 20b. The solution of the primal problem provides vital information for generating the first outer approximation to approximate nonlinear functions, only for active constraints, as illustrated in Figure 20c. Next, the dual MILP problem is solved, and its solution provides the next binary variables to be utilized in the primal problem. The solution of the dual problem is shown in Figure 20d. This iterative process continues until an acceptable linear approximation of the nonlinear functions around the optimal point is obtained.

5.4.3 Event-Triggered and Multi-Cut Generation

This section presents an event-triggered technique utilized by the `DiPOA` algorithm to improve the quality of the outer approximations while also controlling the complexity of the resulting dual problem (126). At the outset of every iteration of the `DiPOA` algorithm, a linear outer approximation (128) is generated and incorporated into the dual problem resulting in an MILP approximation of the original MINLP problem. By enforcing the linear approximations at the subsequent iterations and resolving the MILP dual problem, a tighter representation of nonlinear terms is achieved as the algorithm iterates.

Nonetheless, in certain scenarios (e.g., when highly nonlinear functions are integrated into the optimization problem), the linear approximations may prove to be ineffective, leading to insufficient improvement in the lower bound. In such instances, the lower bound that is computed from solving the MILP dual problem may be improved gradually, necessitating a significant number of iterations. The `DiPOA` algorithm implements event-triggered and multi-cut generation techniques to enhance the quality of the outer approximation and corresponding lower bounds produced by the dual problem.

5.4.3.1 Event-Triggered Second-Order Cut Generation

In various practical applications of the SCO problem, such as machine learning and control, the nonlinear functions are often both convex and strongly convex. By leveraging the strong convexity assumption, we can specialize the method proposed in (SU et al., 2018) to achieve a distributed global underestimator without requiring online tuning. This approach enables us to obtain the tightest possible global quadratic approximation of the local nonlinear functions. To further enhance the quality of the outer approximations and corresponding lower bounds generated by the dual problem, we introduce an event-triggered scheme called the Event-Triggered Second-Order Cut (ET-SoCut) method. This technique involves generating and adding second-order information provided in (129) to the dual problem at some specific iterations, which can lead to significant improvements in the lower bounds.

In order to demonstrate the effectiveness of utilizing second-order information in the approximation process, we first present the following illustrative example.

Example 2. Consider a convex function $f : \mathbb{R}^2 \to \mathbb{R}$ defined as,

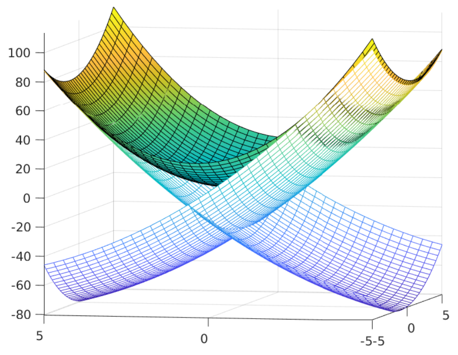
$$f(x) = x^2 + 3y^2 + 0.5xy + 1. \tag{136}$$

By computing the Hessian matrix we have:

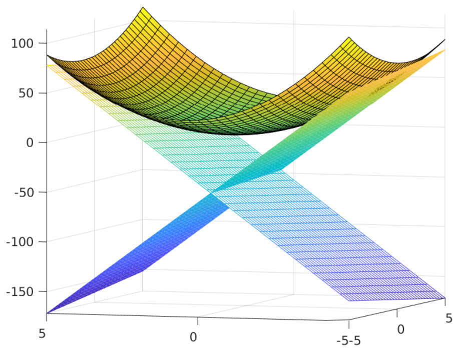
$$\nabla^2 f(x,y) = \begin{bmatrix} 1 & 0.25 \\ 0.25 & 3 \end{bmatrix}.$$

Since f is quadratic and the Hessian is constant, the strong convexity parameter, m, can be easily obtained. In this case m is the minimum Eigenvalue of $\nabla^2 f(x,y)$, which is computed as $m = 0.969$. Figure 21 shows the obtained approximations for both linear and quadratic underestimators. The approximations are performed around two points $(x_1,y_1) = (-2,4)$ and $(x_2,y_2) = (2,-4)$. As depicted in the Figure 21a, the quadratic underestimators provide a better lower bound for the function f. The lower bound provided by the quadratic approximation is $\mathrm{D_b} = -22.552$ while the linear underestimators provide a lower bound around $\mathrm{D_b} = -47.88$. Although quadratic underestimators provide a better lower bound for strongly convex functions, they increase the complexity of the DIPOA dual problem.

For general strongly convex functions, it is more challenging to compute m efficiently. However, in practical problems of learning and control, the computation of m can be straightforward. For example, in many instances of sparse Model Predictive Control (s-MPC) problems (which is a subclass of the SCO problem) the objective function



(a)



(b)

Figure 21 – Comparison between linear and quadratic underestimators. a) quadratic underestimators. b) linear underestimators

is a convex quadratic function. In s-MPC, the goal is to control a process employing a reduced number of inputs, which can improve the operation of the control system (AGUILERA et al., 2017). For convex quadratic functions, m is the smallest Eigenvalue of the Hessian matrix. In machine learning problems the objective function usually consists of a convex function and a strongly convex *regularization* term. In this case, m can be computed from the regularization term.

More details about implementing SoCuts are discussed later. In summary, based on (129), we generate quadratic cuts for the nonlinear convex functions of the SCO problem (112) to accelerate the convergence of the `DiPOA` algorithm. Traditionally, solving MIQCP problems is more challenging and time-consuming than MILPs due to their complexity. However, modern MIP solvers like CPLEX and GUROBI have made significant progress in solving MIQCPs. In particular, for convex MIQCPs with dense Hessian matrices, MIP solvers can handle problems with constraints in the form of (129) that approximate the Hessian with a quadratic diagonal matrix. These constraints use the minimum Eigenvalue of the Hessian as their elements. This feature is advantageous for SoCuts (129) since they can simplify the MIQCP dual problem, making it easier to solve.

However, despite the effectiveness of modern solvers, it is still crucial to keep the dual problem simple and easily solvable to ensure the efficiency of the `DiPOA` algorithm. To keep the dual problem as simple as possible, and also benefit from better approximations (SoCuts), we propose the ET-SoCut generation strategy in which the SoCuts are generated whenever necessary. To do so, we define an event that is based on the relative optimality gap, $\mathrm{G}_{\mathtt{rel}}$ of the `DiPOA` algorithm. The relative optimality gap at each `DiPOA` iteration k is defined as,

$$\mathrm{G}^{k}_{\mathtt{rel}} = \frac{\mathrm{P}^{k}_{\mathtt{b}} - \mathrm{D}^{k}_{\mathtt{b}}}{\max\{\mathrm{P}^{k}_{\mathtt{b}}, 10^{-10}\}} \times 100 \tag{137}$$

Based on the relative gap (137), at each iteration k, we define

$$e^k_{\mathtt{soc}} = \frac{\mathrm{G}^{k-1}_{\mathtt{rel}} - \mathrm{G}^{k}_{\mathtt{rel}}}{\mathrm{G}^{k-1}_{\mathtt{rel}}} \tag{138}$$

as an event for the dual problem. At each iteration k, $e^k_{\mathtt{soc}}$ measures the difference between two consecutive relative gaps. This event is triggered, whenever,

$$e^k_{\mathtt{soc}} \leq \varepsilon_{\mathtt{soc}} \tag{139}$$

where $\varepsilon_{\mathtt{soc}} > 0$ is a small number which determines the iteration from which SOC cuts are generated. Therefore, using the ET-SoCut strategy, at each iteration of the `DiPOA` algorithm, the relative increment error is computed and if it is smaller than a threshold, a SoCut is generated for the master's problem. In other words, to avoid generating the SoCuts at each iteration, we just add SoCuts whenever the relative gap starts to flatten out. Otherwise, the linear approximations are generated as usual. By using ET-SoCut, a small number of SoCuts are generated to help the MIP solver for solving the problem efficiently. More details about the ET-SoCut strategy are provided in the implementation section.

5.4.3.2 Multi-Cut Generation

The multi-cut generation techniqe of `DiPOA` is based on the lifted problem formulation (123). The lifted formulation (123) provides several benefits, one of which is particularly relevant to this discussion: it enables each agent in the network to locally generate outer approximations for local nonlinear functions, as demonstrated in Equations (128) and (129). This approach facilitates parallel cut generation in `DiPOA`, allowing multiple outer approximations to be generated simultaneously at each iteration. By leveraging this technique, the quality of the lower bounds generated by the dual problem is improved, as more approximations are introduced into the problem.

5.4.4 Algorithmic details and pseudo-code

This section presents implementation details and the pseudo-code of the `DiPOA` algorithm which is summarized in Algorithm 3. The algorithm starts by specifying the accepted optimality tolerances, denoted as $\varepsilon_{\mathtt{abs}}$, $\varepsilon_{\mathtt{rel}}$, and $\varepsilon_{\mathtt{soc}}$. These tolerances determine the accuracy of the optimal solution and generated second order event. Afterwards, several parameters are initialized by `DiPOA`. These include the iteration index k, as well as the relative and absolute optimality gaps denoted by $\mathrm{G}_{\mathtt{rel}}$ and $\mathrm{G}_{\mathtt{abs}}$, respectively. Furthermore, the algorithm sets up the primal and dual bounds, denoted by $\mathrm{P}_{\mathtt{b}}$ and $\mathrm{D}_{\mathtt{b}}$, respectively. `DiPOA` also initializes $\mathrm{P}_{\mathtt{cut}}$, which is a set of outer approximations, and $\mathrm{E}_{\mathtt{SOC}}$, which indicates whether a second-order cut event has been emitted or not. Note that some technical implementation details including the computation of $\mathrm{G}_{\mathtt{rel}}$

Algorithm 3 Basic steps of DiPOA Algorithm

```
1: Specify accepted optimality tolerances $\varepsilon_{\mathtt{abs}}$, $\varepsilon_{\mathtt{rel}}$, and $\varepsilon_{\mathtt{soc}}$.
2: Initialize: $k \leftarrow 0$, $G^0_{\mathtt{rel}} \leftarrow \infty$, $G^0_{\mathtt{abs}} \leftarrow \infty$, $P^0_{\mathtt{b}} \leftarrow \infty$, $D^0_{\mathtt{b}} \leftarrow -\infty$, $P^0_{\mathtt{cut}} \leftarrow \emptyset$,
   $E_{\mathtt{SOC}} \leftarrow 0$.
3: $(\mathtt{x_r}, \mathtt{z_r}) \leftarrow$ NLPRelaxation()        ▷ NLP Rexalation of Problem (123)
4: if IsFeasible($\mathtt{x_r}$) then        ▷ $\mathtt{x_r} \in \mathcal{L} \cap \mathcal{N}_i$, $\forall i = 1,...,N$
5:     return $\mathtt{x}^* \leftarrow \mathtt{x_r}$, $z^*_{\mathtt{SOC}} \leftarrow \mathtt{z_r}$
6: $P\mathtt{cut} \leftarrow \widetilde{\mathcal{N}}^0_i$.
7: while $G^k_{\mathtt{rel}} > \varepsilon_{\mathtt{rel}}$ or $G^k_{\mathtt{abs}} > \varepsilon_{\mathtt{abs}}$ do
8:     $k \leftarrow k + 1$
9:     $(\delta_k, D^k_{\mathtt{b}}) \leftarrow$ MILPRelaxation($P^{k-1}_{\mathtt{cut}}$)        ▷ Dual Problem (126)
10:    $(\mathtt{x_k}, P_{\mathtt{k}}) \leftarrow$ DistNLP($\delta_k$)        ▷ Primal Problem (130), Distributed Update
11:    $(\mathtt{x_P}, P^k_{\mathtt{b}}) \leftarrow$ CurrentIncumbent($\mathtt{x_k}$, $P_{\mathtt{k}}$)
12:    $E_{\mathtt{SOC}} \leftarrow$ SOCEvent($D^k_{\mathtt{b}}$, $D^{k-1}_{\mathtt{b}}$)        ▷ Eq. (138)
13:    for $i = 1,\dots,N$ do        ▷ Parallel Update
14:        if $E_{\mathtt{SOC}}$ then
15:            $(\widetilde{\mathcal{N}}^k_i, P^k_{\mathtt{cut}}) \leftarrow$ GenerateQuadraticOA($\mathtt{x_k}$)        ▷ Eq. (129)
16:        else
17:            $(\widetilde{\mathcal{N}}^k_i, P^k_{\mathtt{cut}}) \leftarrow$ GenerateLinearOA($\mathtt{x_k}$)        ▷ Eq. (128)
18:    $(G^k_{\mathtt{rel}}, G^k_{\mathtt{abs}}) \leftarrow$ CalculateOptimalityGap($P^k_{\mathtt{b}}$, $D^k_{\mathtt{b}}$)
19: return $\mathtt{x}^* \leftarrow \mathtt{x_P}$, $z^*_{\mathtt{SOC}} \leftarrow P^k_{\mathtt{b}}$
```

and G$_{\mathtt{abs}}$, distributed and parallel updates and communications between agent will be explained in detail in the next chapter.

In step (3), DiPOA attempts to solve the NLP relaxation of the MINLP problem (123) using the procedure NLPRelaxation. This step is performed by RH-ADMM algorithm since it is a distributed convex optimization problem whose structure is suitable for RH-ADMM. In case, the solution of the NLP relaxation, $\mathtt{x_r}$ is feasible with respect to the orignina1 MINLP problem, DiPOA returns it as the optimal solution. Otherwise, the algorithm generates the initial set of outer approximations for the nonlinear constraints using $\widetilde{\mathcal{N}}^0_i$ and construct the dual problem (126).

Afterwards, the main loop of the algorithm begins, and it continues until the relative and absolute optimality gaps converge to specified tolerances. In each iteration, the algorithm solves the dual problem using the procedure MILPRelaxation and obtains the optimal solution. The algorithm then solves the primal problem using the procedure DistNLP, which is a distributed optimization problem, and generates the incumbent solution using the procedure CurrentIncumbent.

Next, the algorithm checks whether a second-order cut generation event is triggered, which is determined by the procedure SOCEvent. If there the event is triggered, the algorithm generates quadratic outer approximations using the procedure GenerateQuadraticOA, and if there is no event, the algorithm generates linear outer approximations using the function GenerateLinearOA.

After generating the outer approximations, the `DiPOA` calculates the optimality gaps using the procedure `CalculateOptimalityGap` and updates the iteration index. The algorithm continues with the next iteration until either the optimality gaps are less than or equal to the specified tolerances.

Finally, the algorithm returns the optimal solution and the optimal value of the SOC, denoted as $\mathbf{x}^*$ and $z_{\texttt{SOC}}$, respectively. We present a numerical example to demonstrate the application of `DiPOA` to a given problem.

Example 3. We consider the DSLinR problem (DSLinR) with $N = 2$ agents, 10 samples for each agent, 3 features, and 2 allowed non-zero elements as follows,

$$\begin{aligned} \min_{\boldsymbol{\theta}} \sum_{i=1}^{2} \|\mathbf{X}_i\boldsymbol{\theta} - \mathbf{b}_i\|_2^2 + \frac{\lambda}{2}\|\boldsymbol{\theta}\|_2^2 \\ \text{subject to} \quad \|\boldsymbol{\theta}\|_0 \leq \kappa \end{aligned} \tag{DSLinR}$$

where,

$$\mathbf{X}_1 = \begin{bmatrix} 0.26 & -1.38 & 0.18 & 0.37 & -1.18 & -0.21 & -0.01 & -0.29 & -0.37 & -0.51 \\ 0.37 & -0.25 & -0.78 & -0.17 & -0.11 & 0.58 & -0.15 & -0.28 & 0.94 & -0.18 \\ 0.21 & 0.58 & 1.91 & 0.80 & 0.81 & 1.16 & 0.45 & 0.47 & -1.81 & 0.57 \end{bmatrix}^T,$$

$$\mathbf{b}_1 = \begin{bmatrix} 1.50 & -0.03 & -1.50 & 0.01 & -1.87 & -1.48 & -1.22 & -0.48 & 0.35 & 0.25 \end{bmatrix}^T,$$

$$\mathbf{X}_2 = \begin{bmatrix} 0.32 & 0.90 & 1.18 & 0.85 & -0.59 & 1.16 & 0.04 & 0.89 & -0.86 & 1.23 \\ -0.89 & -1.01 & -0.16 & 0.82 & -0.88 & -2.41 & 1.16 & 0.40 & 1.18 & 1.97 \\ -1.23 & 0.73 & 0.62 & -0.92 & 2.06 & 0.22 & -0.60 & 0.77 & 0.59 & 0.44 \end{bmatrix}^T,$$

$$\mathbf{b}_2 = \begin{bmatrix} 0.47 & 0.45 & 1.48 & -1.15 & 0.64 & 2.13 & 0.25 & 0.99 & 0.99 & 1.27 \end{bmatrix}^T.$$

Algorithm 3 utilizes the `RH-ADMM` algorithm to solve the NLP relaxation of the original problem in a distributed manner. The solution of the NLP relaxation problem yields $\mathbf{x}_r$ and δ_r, with values of $\begin{bmatrix} 0.628 & -0.075 & -0.073 \end{bmatrix}^T$ and $\begin{bmatrix} 0.704 & 0.722 & 0.281 \end{bmatrix}^T$, respectively.

After approximating nonlinear objective functions using (128), we construct the first dual MILP problem with the lifted-formulation (123). Solving this problem yields the first set of feasible binary variables, $\delta^1 = \begin{bmatrix} 0 & 1 & 1 \end{bmatrix}^T$, which we use to solve the primal problem (130). As a result of the first iteration of `DiPOA`, we obtain the next point around which outer approximations are generated: $\mathbf{x}_1 = \begin{bmatrix} 0 & -0.123 & -0.119 \end{bmatrix}^T$. The primal and dual bounds generated by the algorithm at this iteration are $\text{P}\mathtt{b}^1 = 12.068$ and $\text{D}^1\mathtt{b} = 9.7562$, respectively.

In the second iteration, a new set of outer approximations is generated by each agent and a new dual problem is constructed and solved. The binary combination obtained in this iteration is $\delta^2 = \begin{bmatrix}1 & 0 & 1\end{bmatrix}^T$, which yields a feasible solution $\mathtt{x}^2 = \begin{bmatrix}0.63 & 0 & -0.05\end{bmatrix}^T$. The primal and dual bounds are then updated as $\mathrm{P}\mathtt{b}^2 = 9.8044$ and $\mathrm{D}^2\mathtt{b} = 9.7563$, respectively. This is the best primal bound found by the algorithm so far, as $\mathrm{P}\mathtt{b}^2 < \mathrm{P}\mathtt{b}^1$, and therefore the incumbent solution is updated as $x_{\mathtt{incumbent}} = \mathtt{x}^2$.

The process is repeated until a sufficiently large dual bound is obtained, which occurs at iteration $k = 4$. At this point, the incumbent solution is not updated since the primal bound is not decreasing. The algorithm is then terminated by reporting $\mathtt{x}^* = \mathtt{x}_2$ as the optimal solution and $f^* = 9.8044$ as the optimal objective value. Table 2 presents a summary of the first 4 iterations of the `DiPOA` algorithm. The table shows the contributions of the first and second agents, denoted by f_i, $i = 1{,}2$, to the computation of primal and dual bounds, denoted by $\mathrm{P}\mathtt{b}$ and $\mathrm{D}\mathtt{b}$, respectively. Additionally, the table lists the number of outer approximations introduced at each iteration, denoted by N_{c}.

Table 2 – Four iterations of `DiPOA`

k	f_1	f_2	γ_1	γ_2	$\mathrm{P}_{\mathtt{b}}$	$\mathrm{D}_{\mathtt{b}}$	$\mathrm{G}_{\mathtt{rel}}$	$\mathrm{G}_{\mathtt{abs}}$	N_{c}
1	5.4546	6.6134	0.9289	8.8272	12.068	9.7562	0.2305	2.9238	2
2	4.8863	4.9181	–0.6099	10.3662	9.8044	9.7563	0.0049	0.0481	4
3	5.3122	4.4941	6.8105	2.9461	9.8062	9.7566	0.0051	0.0496	6
4	4.8863	4.9181	4.4673	5.337	9.8044	9.8043	$1e^{-5}$	0.0001	8

5.5 DISTRIBUTED HYBRID OUTER APPROXIMATION

In this section, we introduce the Distributed Hybrid Outer Approximation (`DiHOA`) algorithm that computes a solution to problem (122). The `DiHOA` algorithm is built on top of the LP/NLP-based BnB algorithm (also known as single-tree outer approximation algorithm), which was originally proposed in (QUESADA; GROSSMANN, Ignacio E, 1992). Unlike the multiple-tree algorithms, which construct a new BnB tree at each iteration, the original LP/NLP-based BnB algorithm creates a single BnB tree. In this method, outer approximations of nonlinear functions are added *lazily* through callbacks to eliminate integer-feasible solutions that violate the nonlinear constraints in the original MINLP problem. Callbacks are methods provided by the MIP solver that are activated at specific points during the solution process, such as when a new integer feasible solution is found or a new node is created in the branching tree. This approach allows the user to implement customized strategies that affect the behavior of the MIP solver in a way that is not possible through a normal solver call. It is also possible to manipulate node generation in the MIP solver using callbacks. Overall, this approach offers greater flexibility to the user to customize and optimize the solution process.

Algorithm 4 Basic steps of `DiPHOA` Algorithm

```
1:  Specify accepted optimality tolerances $\varepsilon_{\mathtt{abs}}$, $\varepsilon_{\mathtt{rel}}$, and $\varepsilon_{\mathtt{soc}}$.
2:  Initialization
3:  $k \leftarrow 0$, $\mathrm{G}^0_{\mathtt{rel}} \leftarrow \infty$, $\mathrm{G}^0_{\mathrm{abs}} \leftarrow \infty$, $\mathrm{P}^0_{\mathrm{b}} \leftarrow \infty$, $\mathrm{D}^0_{\mathrm{b}} \leftarrow -\infty$, $\mathrm{P}^0_{\mathtt{cut}} \leftarrow \emptyset$, $\mathrm{E}_{\mathtt{SOC}} \leftarrow 0$. $\mathcal{P} \leftarrow \emptyset$
4:  $(\mathtt{x_r}, \mathtt{z_r}) \leftarrow$ NLPRelaxation()          ▷ NLP Relaxation of Problem (123)
5:  if IsFeasible($\mathtt{x_r}$) then          ▷ $\mathtt{x_r} \in \mathcal{L} \cap \mathcal{N}_i$, $\forall i = 1,...,N$
6:      return $\mathtt{x}^* \leftarrow \mathtt{x_r}$, $z^*_{\mathtt{SOC}} \leftarrow \mathtt{z_r}$
7:  $\mathrm{P}\mathtt{cut} \leftarrow \widetilde{\mathcal{N}}^0_i$.          ▷ Cunstruct dual problem (126)
8:  MultipleTreeSearch
9:  while not SOCEvent($\mathrm{D}^k_{\mathrm{b}}, \mathrm{D}^{k-1}_{\mathrm{b}}$) do
10:     $k \leftarrow k + 1$
11:     $(\delta_k, \mathrm{D}^k_{\mathrm{b}}) \leftarrow$ MILPRelaxation($\mathrm{P}^{k-1}_{\mathtt{cut}}$)          ▷ Dual Problem (126)
12:     $(\mathtt{x_k}, \mathrm{P_k}) \leftarrow$ DistNLP($\delta_k$)          ▷ Primal Problem (130), Distributed Update
13:     $(\mathtt{x_P}, \mathrm{P}^k_{\mathrm{b}}) \leftarrow$ CurrentIncumbent($\mathtt{x_k}, \mathrm{P_k}$)
14:     $\mathrm{E}_{\mathtt{SOC}} \leftarrow$ SOCEvent($\mathrm{D}^k_{\mathrm{b}}, \mathrm{D}^{k-1}_{\mathrm{b}}$)          ▷ Eq. (138)
15:     for $i = 1, \ldots, N$ do          ▷ Parallel Update
16:         $(\widetilde{\mathcal{N}}^k_i, \mathrm{P}^k_{\mathtt{cut}}) \leftarrow$ GenerateQuadraticOA($\mathtt{x_k}$)
    SingleTreeSearch
17: $\mathcal{P}^0 \leftarrow$ BnBRootInit($\mathrm{P}^k_{\mathtt{cut}}$)          ▷ initialize BnB Root with SOC cuts
18: $\mathcal{P} \leftarrow \mathcal{P} \cup \mathcal{P}^0$
19: while $\mathrm{G}^k_{\mathtt{rel}} > \varepsilon_{\mathtt{rel}}$ or $\mathrm{G}^k_{\mathrm{abs}} > \varepsilon_{\mathtt{sbs}}$ do
20:     $\mathcal{P}^k \leftarrow$ SelectNode($\mathcal{P}$)          ▷ Select and Remove Node
21:     $(\mathtt{x_k}, \delta_k) \leftarrow$ QCLPRelaxation($\mathcal{P}^k$)          ▷ Process Node
22:     if IsIntegerFeasible($\delta_k$) then
23:         $\mathrm{D}^k_{\mathrm{b}} \leftarrow$ UpdateDualBound()
24:         $(\mathtt{x_k}, \mathrm{P_k}) \leftarrow$ DistNLP($\delta_k$)          ▷ Primal Problem (130), Distributed Update
25:         LazyCutGeneration($\mathtt{x_k}$)          ▷ Dual Problem Lazy Update
26:         NodesUpdate($\mathtt{x_k}, \mathcal{P}$)          ▷ Update All Open Nodes
27:         $(\mathtt{x_k}, \delta_k) \leftarrow$ ResolveQCLPRelaxation($\mathtt{N_k}$)          ▷ Resolve QCLP relaxation
28:         $(\mathtt{x_P}, \mathrm{P}^k_{\mathrm{b}}) \leftarrow$ CurrentIncumbent($\mathtt{x_k}, \mathrm{P_k}$)
29:         $(\mathrm{G}^k_{\mathtt{rel}}, \mathrm{G}^k_{\mathrm{abs}}) \leftarrow$ CalculateOptimalityGap($\mathrm{P}^k_{\mathrm{b}}, \mathrm{D}^k_{\mathrm{b}}$)
30:     else
31:         $(\mathcal{P}^k_1, \mathcal{P}^k_2) \leftarrow$ Branch($\mathcal{P}^k$)
32:         $\mathcal{P} \leftarrow \mathcal{P} \cup \mathcal{P}^k_1 \cup \mathcal{P}^k_2$
33:     $k \leftarrow k + 1$
34: return $\mathtt{x}^* \leftarrow \mathtt{x_P}$, $z^*_{\mathtt{SOC}} \leftarrow \mathrm{P}^k_{\mathrm{b}}$
```

Based on DiPOA and LP/NLP-based BnB, we develop the DiHOA algorithm to avoid constructing many similar MIP BnB trees. The main idea of DiHOA is to iteratively build a single BnB tree whereby the primal problem is solved distributedly, while dynamically updating the dual problem (126) without reconstructing the BnB tree. However, the single-tree OA algorithm may lead to a large BnB tree and weaker approximations. To avoid that, DiHOA introduces several second-order outer approximations of the nonlinear functions to the root of the BnB tree through an event-triggered scheme, which leads to a tighter problem representation and a BnB tree with a fewer number of nodes. This procedure constructs the first MIP dual problem and initiates the BnB algorithm. During the BnB search, as soon as a new integer-feasible solution is found, the primal problem (130) is distributedly solved to determine whether a *lazy constraint* removing this integer-feasible point should be generated. By lazy constraints, we mean cutting planes that are lazily added to the MIP model whenever an integer feasible solution is found. At this point, the RH-ADMM algorithm is applied, and the new primal information is distributed to the computational nodes of the network. Then the generated lazy constraint is added to the current node, and all open nodes of the BnB tree and the search continues. Therefore, it is not required to reconstruct the BnB tree as in multiple-tree algorithms and the same BnB tree can be used after adding new linearizations as lazy constraints. Finally, the algorithm is terminated until the MIP integer relaxation results in a feasible integer solution to the MINLP problem (122).

A detailed description of the DiHOA algorithm is summarized in Algorithm 4. It can be observed in Algorithm 4 that DiHOA consists of three primary computational phases, namely, Initialization, MultipleTreeSearch, and SingleTreeSearch steps. In a manner similar to the approach taken by DiPOA, the feasibility of the NLP relaxation of problem (123) is the first aspect that DiHOA considers. In case the solution of the NLP relaxation problem is integer-feasible with respect to the original MINLP problem, DiPOA terminates. After the algorithm is successfully initialized, DiHOA starts a multiple-tree strategy with a second-order outer approximation cut generation strategy and continues the computations until either the solution is found or poor lower bound improvement is achieved.

In the former case, the algorithm is terminated and the optimal solution is returned. In the latter case, however, DiHOA accumulates all the outer approximations obtained until iteration k in the root of the latest BnB tree and, then, it starts a single-tree search strategy whereby approximations are added dynamically. As the name of the algorithm suggests, DiHOA is a *hybrid* algorithm that combines both single-tree and multiple-tree strategies by introducing an event-triggered scheme that determines the switching iteration, k_{switch}, at which a single-tree search strategy is started. In the following, we describe each computational step in detail.

5.5.1 Initialization Step

To begin the initialization step, the first task is to solve the NLP relaxation of the original problem (123). This step involves disregarding the integrality constraint from the binary variables, resulting in a distributed nonlinear programming problem with both linear and nonlinear constraints. This problem can be solved using the `RH-ADMM` algorith through the `NLPRelaxation` method. If the relaxation step obtains a feasible solution with respect to problem (122), then the optimization process is terminated with the optimal solution achieved.

On the other hand, if the relaxation step does not obtain an integer-feasible solution, then we proceed to generate N first-order outer approximations using the local information available in each agent of the network. Based on these approximations, we construct the first MIP dual problem as per (126). During this process, the `IsFeasible` method is used to check whether the solution obtained from the `NLPRelaxation` method is feasible with respect to the original MINLP problem. This step ensures that the solution obtained is valid and satisfies all the constraints of the original problem.

5.5.2 Multiple Tree Search Step

The multiple-tree search step of the `DiHOA` algorithm is a crucial step that directly affects the `DiHOA` performance. In the cases that k_{switch} is a large number, a pure multiple-tree algorithm with second-order outer approximations is obtained. Otherwise, the resulting algorithm becomes a pure single-tree algorithm. Therefore, k_{switch} should be determined in such a way that maximizes the `DiHOA` performance. To do so, we introduce an event-triggered scheme that selects k_{switch} based on the difference between two consecutive lower bounds. In particular, during the solution procedure, `DiHOA` checks if the generated lower bounds by the dual problem start to flatten out within a given tolerance $\varepsilon_{\mathtt{SIC}} > 0$ and triggers a switching event, $E_{\mathtt{SOC}}$, if $\mathrm{D}_{\mathrm{b}}^{k} - \mathrm{D}_{\mathrm{b}}^{k-1} \leq \varepsilon$. As soon as $E_{\mathtt{SOC}}$ is triggered, `DiHOA` switches to the single-tree strategy by performing the BnB algorithm on the latest dual problem, which was obtained during the multiple-tree strategy.

5.5.3 Single Tree Search Step

The single-tree search step is activated for $k > k_{switch}$ after the $E_{\mathtt{SOC}}$ event is triggered. k_{switch} is the iteration step at which the single-tree search is activated. In this step, `DiHOA` accumulates all the outer approximations obtained during the `MultipleTreeSearch` phase in the root of the latest BnB tree and then starts the single-tree BnB procedure. The BnB search is initialized by solving an integer relaxation of the dual problem (126). In each node of the BnB tree, a (Quadratically Constrained Linear Programming) QCLP relaxation is solved and the search is stopped once an integer solution is obtained in one of the nodes. The integer solution is then used to solve the primal problem (130) with integer variables fixed. The primal solution provides a valid upper bound and new approximations can be generated. The new approximations are then added to all open nodes in the BnB tree and the QCLP relaxation is resolved for the node which resulted in the integer combination. The BB procedure continues from the existing search tree with the improved polyhedral outer approximation.

As in the standard BnB, nodes can be pruned off in case the optimum of the QCLP relaxation exceeds the upper bound. However, the search cannot be stopped once an integer solution is obtained at a node, which must continue until the QCLP relaxation results in a feasible integer solution for problem (123) or until the node can be pruned off.

Finally, since all variables in problem (112) are bounded, then the assumptions A1–A3 in (FLETCHER; LEY, 1996) and assumptions in (KRONQVIST; BERNAL; GROSSMANN, Ignacio E., 2020) are valid and Slater's constraint qualification holds for problem (122) when the binary variables are fixed. Hence, the convergence of Algorithm 4 follows the converge proof of the centralized case.

Example 4. In this example, we solve the problem provided in Example 3 by applying the `DiHOA` algorithm. The BnB tree with respect to this problem is depicted in Figure 22 where $\mathrm{C}_{\mathrm{P}_i}$ is the callback of the node P_i. To simplify the process, we omit second-order outer approximations in the root of the BnB and use Breath-First Search (BFS) to traverse the BnB tree. The `DiHOA` algorithm, similar to `DiPOA`, constructs the first dual problem (126) by solving the NLP relaxation of the original problem. Once the dual problem is constructed, the BnB tree is initialized, and an LP relaxation for the dual problem is solved at node P_1. Solving the first LP relaxation problem in the root of the BnB tree yields an integer-feasible solution of $\delta^1 = \begin{bmatrix} 0 & 1 & 1 \end{bmatrix}^T$ with $\mathrm{Db} = 9.7562$. Since δ^1 is integer-feasible with respect to the MILP dual problem, we can introduce a lazy outer approximation for each agent. To do this, we solve the primal problem (130) in a distributed manner using the callback introduced in node 1 (*i.e.* $\mathrm{C}_{\mathrm{P}_1}$). In this case, we obtain $\mathbf{x}^1 = \begin{bmatrix} 0 & -0.1239 & -0.1196 \end{bmatrix}^T$, around which local objective functions are linearized. The point $\mathbf{x}^1$ is feasible with respect to the original problem and serves as the first incumbent solution, denoted by $\mathtt{xincumbent} = \mathbf{x}^1$, with primal bound $\mathrm{Pb}^1 = 12.0680$.

In the following step, lazy outer approximations are introduced to the node P_1 and the LP relaxation is resolved (denoted by $\mathrm{P}_{1,r}$), resulting in a fractional solution $\delta = \begin{bmatrix} 0.3723 & 0.6277 & 1 \end{bmatrix}^T$. Thus, $\mathrm{P}_{1,r}$ is divided into two sub-problems, P_2 and P_3, based on one of the fractional variables. In this case, we choose δ_2 to split the problem.

Upon solving the LP relaxation of P2, an integer-feasible solution is obtained as $\delta^2 = \begin{bmatrix} 1 & 0 & 1 \end{bmatrix}^T$, with $\mathrm{Db} = 9.7563$. Next, the next callback is called, and we solve an instance of the primal problem with new fixed binary variables. By solving the primal problem, we obtain a new feasible solution $\mathrm{x}^2 = \begin{bmatrix} 0.6370 & 0 & -0.0545 \end{bmatrix}^T$ with $\mathrm{P}_\mathrm{b}^2 = 9.8044$. Therefore, $\mathrm{x}_{\mathtt{incumbent}} = \mathrm{x}^2$, and P_b^2 is the best primal bound found by the algorithm so far.

In the next step, we generate and introduce lazy outer approximations to P_2 and P_3 and resolve the LP relaxation of $\mathrm{P}_{2,r}$. However, upon solving the LP relaxation of P_2, we obtain the same integer-feasible solution, indicating that further exploration of this node will not improve the solution. Hence, we can prune $\mathrm{P}_{2,r}$.

The algorithm proceeds to consider the node P_3, where the solution to the LP relaxation is fractional, specifically $\delta = \begin{bmatrix} 0.3481 & 1 & 0.6519 \end{bmatrix}^T$. In this case, the branching is based on δ_1, and P_3 is divided into two sub-problems, P_4 and P_5.

When solving P4, a feasible integer solution is obtained with $\mathrm{D}_\mathrm{b} = 12.067$. As $\mathrm{D}_\mathrm{b} > \mathrm{P}_\mathrm{b}^2$, P_4 is fathomed.

Finally, the LP relaxation of the dual problem is solved for P5, resulting in an integer-feasible solution $\delta^2 = \begin{bmatrix} 1 & 1 & 0 \end{bmatrix}^T$ and $\mathrm{L}_\mathrm{b}^2 = 9.7566$. Using callbacks, the primal problem is solved, and a feasible solution $\mathrm{x}^3 = \begin{bmatrix} 0.6371 & -0.0552 & 0 \end{bmatrix}^T$ with $\mathrm{P}_\mathrm{b} = 9.8062$ is obtained. However, since $\mathrm{P}_\mathrm{b}^3 > \mathrm{P}_\mathrm{b}^2$, the incumbent solution remains unchanged. The same set of binary variables is obtained by generating new outer approximations and resolving P_5, so P_5 is fathomed.

Therefore, the optimal solution is provided by $\mathrm{P}_{2,r}$. Tables 3 and 4 show the details of each iteration of the DiHOA.

Table 3 – Branch and Bound Nodes Information

node	γ_1	γ_2	D_b	N_c	status	branch
P_1	0.9289	8.8273	9.7561	2	integral	no
$\mathrm{P}_{1,r}$	0.355	9.4012	9.7562	4	fractional	yes
P_2	–0.6099	10.366	9.7563	4	integral	no
$\mathrm{P}_{2,r}$	4.4672	5.3370	9.8043	6	pruned	no
P_3	–0.6099	10.366	9.7563	6	fractional	yes
P_4	13.7451	–1.6773	12.0670	6	pruned	no
P_5	6.810	2.9461	9.7566	6	integer	no
$\mathrm{P}_{5,r}$	5.6984	4.1076	9.8061	8	pruned	no

Table 4 – Lazy Callbacks Information

callback	f_1	f_2	P_b	$\mathrm{G}_\mathtt{rel}$	$\mathrm{G}_\mathtt{abs}$	status
$\mathrm{C}_{\mathrm{P}_1}$	5.4546	6.6133	12.0680	0.1915	2.3118	feasible
$\mathrm{C}_{\mathrm{P}_2}$	04.8862	4.9180	9.8044	0.0049	0.0481	feasible
$\mathrm{C}_{\mathrm{P}_5}$	5.3121	4.4940	9.8062	0.0050	0.0496	feasible

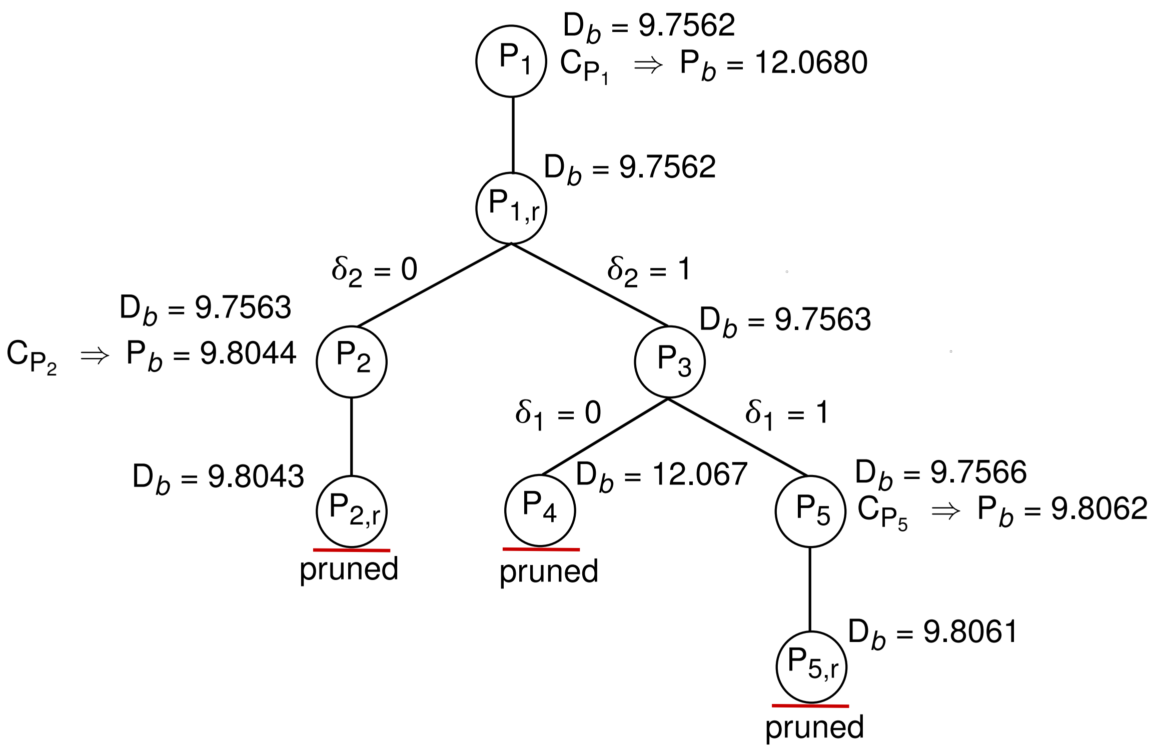


Figure 22 – Branch and bound tree

In summary, both DiPOA and DiHOA algorithms provide a distributed solution for SCO problems in which different strategies are used. Table 5 reports the main differences between these two algorithms.

	DiPOA	DiHOA
BnB strategy	multiple-tree	multiple-tree-single-tree
Cut generation	Problem reconstruction	Lazy constraints
Event scheme	FOC cuts to SOC cuts	multiple-tree to single-tree
Subproblems	Distributed NLP – MI(QC)LP	Distributed NLP / (QC)LP

Table 5 – Key differences of DiPOA and DiHOA algorithms.

5.6 SPECIALIZED FEASIBILITY PUMP

Feasibility Pump (FP) methods refer to a set of algorithms and techniques designed to efficiently find initial feasible solutions of MIP problems (see (BERTHOLD; LODI; SALVAGNIN, 2019) and references therein). The FP algorithms are fundamentally based on the idea of generating two sequences of points that aim to converge toward a feasible solution for a given MIP problem. One sequence consists of the points that are feasible for continuous relaxation but possibly integer infeasible. The other sequence consists of points that are integral but might violate some of the imposed constraints. The next point of one sequence is always generated by minimizing the distance to the last point of the other sequence, using different distance measures in either case (e.g., the ℓ_1 or the ℓ_2 norm).

In this section, we propose a Specialized Feasibility Pump (SFP) method tailored for SCO problem (112) which aims to provide a good feasible sparse solution. The SFP method consists of three main steps, namely, *relaxation*, *sparse projection*, and *alternating projection* steps which are discussed in the following subsections.

5.6.1 Relaxation Step

The main goal of the relaxation step is to find an initial point $\mathrm{x}_{\mathrm{R}}^0$ such that $\mathrm{x}_{\mathrm{R}}^0 \in \mathcal{L} \cap \mathcal{N}$ however the sparsity constraint $\mathrm{x}_{\mathrm{R}}^0 \in \mathcal{C}_\kappa$ might not be satisfied. To compute $\mathrm{x}_{\mathrm{R}}^0$, the simplest method is to solve a convex *feasibility* problem that disregards the sparsity constraint and the objective function in (112) which we refer to as the feasibility-based SFP method. However, this approach may produce a poor a feasible solution whose upper bound is far from the optimal value. To address this, we incorporate the objective function into the optimization and use the ℓ_1 norm as an approximation for the ℓ_0 norm. This method is referred to as the optimality-based SFP method and results in a distributed convex optimization problem, defined as follows:

$$\begin{aligned} z_{\mathrm{R}} = \min_{\mathrm{x}\in\mathbb{R}^n} \ & \sum_{i=1}^{N} f_i(\mathrm{x}) \\ \text{subject to} \ \ & \mathrm{x} \in \mathcal{L} \cap \mathcal{N}_{\mathrm{g}} \cap \widetilde{\mathcal{C}}_\kappa \end{aligned} \tag{140}$$

where $\widetilde{\mathcal{C}}_\kappa = \{\mathrm{x} \in \mathbb{R}^n : \|\mathrm{x}\|_1 \le \kappa\}$. Therefore it is easy to observe that $z_{\mathrm{R}} \le z_{\mathrm{SOC}}$.

5.6.2 Sparsity Projection Step

The point $\mathrm{x}_{\mathrm{R}}^0$ computed from (140) does not necessarily satisfy the original sparsity constraint (*i.e.*, $\left\|\mathrm{x}_{\mathrm{R}}^0\right\|_0 \ge \kappa$). The sparsity projection step is responsible to find an initial sparse solution $\mathrm{x}_{\mathrm{S}}^0$ that satisfies the sparsity constraint which can be done by projecting $\mathrm{x}_{\mathrm{R}}^0$ to the set of ℓ_0 norm, $\mathcal{C}_\kappa$. Before discussing the sparsity projection step, we define the projection on a set $\mathcal{A}$.

Definition 21. *The projection of a point $\mathrm{y} \in \mathcal{R}^n$ onto a closed set $\mathcal{A} \subseteq \mathcal{R}^n$ is denoted by y^* and defined as:*

$$\mathrm{y}^* = \Pi_{\mathcal{A}}(\mathrm{y}) = \arg\min_{\mathrm{y}\in\mathcal{A}} \|\mathrm{y}_0 - \mathrm{y}\| \tag{141}$$

In some special cases, the projection of a point on a set is unique. For instance, if $\mathcal{A}$ is a closed and convex set and the norm is strongly convex, then the projection is unique (BOYD, S.; BOYD, S. P.; VANDENBERGHE, 2004). However, for non-convex sets, the projection is not unique and can be difficult to achieve. However, for some specific cases, it is not too difficult to compute the projection on a non-convex set. In the case of the set $\mathcal{C}_\kappa$ in which the ℓ_0 norm is incorporated, we can define the sparsity projection as follows:

Definition 22. *The projection $\Pi_{\mathcal{C}_\kappa}(\mathrm{y})$ of a point $\mathrm{y} \in \mathcal{R}^n$ onto the set $\mathcal{C}_\kappa$ keeps the κ largest (in absolute value) elements and zeros out the remaining, breaking ties in the lexicographic order.*

Therefore we can compute $\mathrm{x}_\mathrm{S}^0 = \Pi_{\mathcal{C}_\kappa}(\mathrm{x}_\mathrm{R}^0)$ which satisfies the sparsity constraints. In case x_S^0 also satisfies the linear and nonlinear constraints the SFP is terminated and a valid upped bound z_S on z_SCO is obtained. Therefore it is easy to write,

$$z_\mathrm{R} \leq z_\mathrm{SOC} \leq z_\mathrm{S} \tag{142}$$

where z_S is the objective value of (112) computed at x_S^0.

Example 5. Consider $\mathrm{y} \in \mathcal{R}^5$ to be defined as $\mathrm{y} = \begin{bmatrix} 5.7 & 1.4 & -3.2 & -2.3 & 2.3 \end{bmatrix}^T$ and $\kappa = 3$. The sparsity projection is obtained as follows:

$$\mathrm{y}^* = \Pi_{\mathcal{C}_\kappa}(\mathrm{y}) = \begin{bmatrix} 5.7 & 0 & -3.2 & -2.3 & 0 \end{bmatrix}^T$$

Based on y^* we can generate a binary vector $\mathrm{z} \in \{0,1\}^5$, such as $\mathrm{z} = \begin{bmatrix} 1 & 0 & 1 & 1 & 0 \end{bmatrix}^T$ in the example, that can be used to initialize the `DiPOA` and `DiHOA` algorithms. The tie between –2.3 and 2.3 is resolved by selecting the element that comes first in the lexicographic order.

As in standard FP methods, point $\mathrm{x}_\mathrm{S}^0 \in \mathcal{C}_\kappa$ might violate some of linear and nonlinear constraints. In such a scenario, we perform the alternating projection step which is discussed in the next section.

5.6.3 Alternating Projection Step

In the context of convex optimization, alternating projections is a simple algorithm for computing a point in the intersection of some convex sets, using a a sequence of projections onto the sets (BOYD, S. et al., 2011). The alternation projection step in SFP consists of generating two sequences of points that hopefully converge to the intersection of sparsity, linear and nonlinear constraints. The algorithm starts with $\mathrm{x}_\mathrm{S}^0 \in \mathcal{C}_\kappa$ obtained from the sparsity projection step and then alternatively projects onto $\mathcal{N}_\mathrm{g}$ and $\mathcal{C}_\kappa$:

$$\mathrm{x}_\mathrm{R}^{k+1} = \Pi_{\mathcal{N}\cap\mathcal{L}}(\mathrm{x}_\mathrm{S}^k), \qquad \mathrm{x}_\mathrm{S}^{k+1} = \Pi_{\mathcal{C}_\kappa}(\mathrm{x}_\mathrm{R}^{k+1})\ , k = 0,1,\ldots. \tag{143}$$

This generates a sequence of points $\mathrm{x}_\mathrm{R}^k \in \mathcal{N} \cap \mathcal{L}$ and $\mathrm{x}_\mathrm{S}^k \in \mathcal{C}_\kappa$. While the alternating projection algorithm is guaranteed to converge when the sets involved are convex, convergence is not guaranteed in the case where a non-convex set is involved. However, even though the convergence of the method is not guaranteed when one of the sets involves ℓ_0 norm constraints, it can still be effective and useful in practice. This is especially true when the method is started with a good initial point. A high-level overview of SFP utilized in SCOT is presented in Algorithm 5.

Algorithm 5 Basic steps of SFP Algorithm

```
 1: Specify accepted optimality tolerance $\varepsilon_{\mathtt{sfp}}$.
 2: $x_R^0 \leftarrow$ NLPRelaxation()                       ▷ NLP Problem (140)
 3: $x_S^0 \leftarrow$ SparsityProject($x_R^0$)              ▷ Sparsity Projection
 4: if IsFeasible($x_S^0$) then          ▷ $x_r \in \mathcal{L} \cap \mathcal{N}_i,\ \forall i = 1,...,N$
 5:     return $x_{\mathtt{feasb}} \leftarrow x_S^0$
 6: $k \leftarrow 0$
 7: while $\left\| x_S^k - x_R^k \right\| \geq \varepsilon_{\mathtt{sfp}}$ do
 8:     $x_R^{k+1}$ = ConvexProject($x_S^k$).
 9:     $x_S^{k+1}$ = SparsityProject($x_R^{k+1}$).
10:     $k \leftarrow k + 1$
    return $x_{\mathtt{feasb}} \leftarrow x_S^k$
```

Example 6. This example demonstrates the effectiveness of the SFP method for solving a Distributed, Sparse Quadratic Programming (DSQP) problem using randomized data. In this example, only 1 variable can be non-zero and both the feasibility and optimality relaxation steps are used. The feasible region of the problem and the iterations of the algorithm are depicted in Figure 23. Figure 23a illustrates the performance of Algorithm 5 with a feasibility-based relaxation step. The algorithm converges in a single iteration, where each agent in the network solves a centralized convex feasibility problem. The SFP method generates a feasible point $x_{\mathtt{feasb}} = [0, 0.08]^T$, while the optimal point is $x_{\mathtt{opt}} = [0, -0.026]^T$ with $f^* = -0.0820$. In this case, the SFP algorithm produces a feasible solution with logarithmic relative error of –12.47, with the objective value value of 0.2775. Figure 23b illustrates the behaviour of the SFP algorithm when incorporating an optimality-based relaxation step. In this case, all agents within the network collaboratively obtain the initial solution x_R, which requires the application of RH-ADMM. Although the computational complexity associated with the optimality-based relaxation step is higher than that of the feasibility-based method, the degree of suboptimality of the generated feasible point is smaller. Specifically, the SFP algorithm terminates in 16 iterations, producing a feasible solution $x_{\mathtt{feasb}} = [0, -0.029]^T$ with a logarithmic relative error of –43.12 and an objective value of –0.0529. These results demonstrate that the SFP algorithm, leveraging the information provided by each agent, can generate a good starting point that is suitable for use in both DiHOA and DiPOA algorithms.

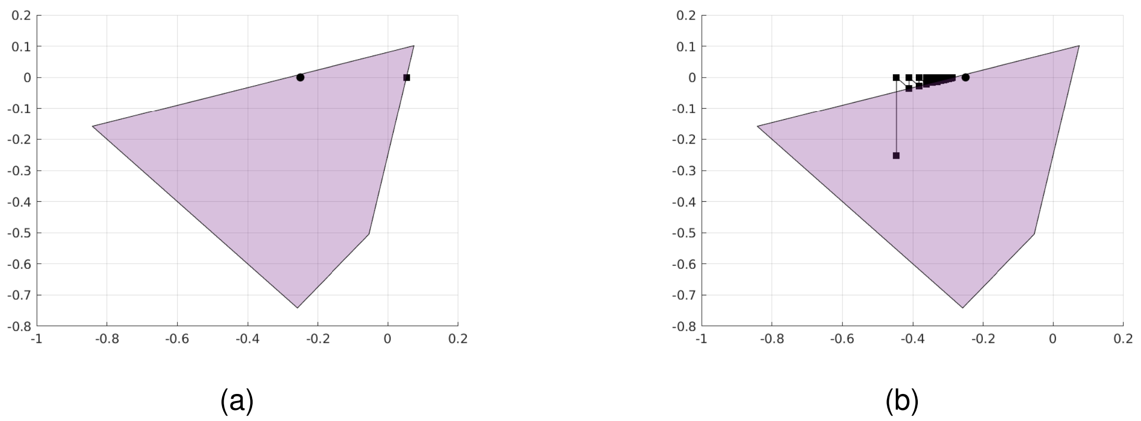


Figure 23 – Behavior of Algorithm 5 for a random DSQP problem; the circular point illustrates the optimal solution of the problem and the square points illustrates the sequence of points generated by the SFP method. a) Feasibility based SFP. b) Optimality based SFP

5.7 INFEASIBILITY DETECTION AND FEASIBILITY CUTS

This section presents an algorithm designed to detect the nonlinear infeasibility of the primal problem (130) before attempting to solve it. Although problem (130) is capable of generating reliable feasible points for generating linear and quadratic approximations, certain combinations of fixed binary variables have the potential to render the nonlinear constraint $\mathcal{N}$ infeasible. Consequently, it is essential to identify an infeasible point prior to attempting to solve the problem.

The practical infeasibility detection technique provides an effective means of detecting such infeasibilities and is a tool for improving the the efficiency of the solution process. In such a situation, one approach is to introduce a constraint, commonly referred to as a *feasibility cut*, that excludes the infeasible point from the feasible region (FLETCHER; LEY, 1996). Feasibility cuts are typically obtained by minimizing the ℓ_1 or ℓ_∞ norm of the infeasible nonlinear constraints. Therefore, it is crucial to distinguish between feasible and infeasible nonlinear constraints, a responsibility that is left to the NLP solver by the original OA algorithm. However, the `DiPOA` and `DiHOA` algorithms do not inherently support infeasibility detection. To address this issue and detect infeasibilities in nonlinear constraints, we formulate the following optimization problem:

$$
\begin{aligned}
\min_{\substack{\mathbf{x}_1,\dots,\mathbf{x}_N \\ \mathbf{y}_1,\dots,\mathbf{y}_K}} & \sum_{i=1}^{N}\sum_{h=1}^{m} g_h(\mathbf{x}_i) \\
s.t.\ & \mathbf{x}_i = \mathbf{y}_j,\ \forall i \in \mathcal{E}_j,\ \ \mathcal{E}_j \in \mathcal{E} \\
& \mathbf{x}_i \in \mathcal{L},\ \forall i = 1,\dots,N \\
& -M_j\delta_j^k \le \mathbf{y}_j \le M_j\delta_j^k,\ \forall j = 1,\dots,K
\end{aligned} \tag{144}
$$

This problem which incorporates the nonlinear constraints of the original problem into the objective function is a consensus convex optimization problem with linear constraints which can be solved using RH-ADMM. Moreover, since $\mathcal{L}$ is a polytope and the Big-M constraints cannot be empty, problem (144) is always feasible. If the optimal solution of (144) yields a positive objective function, then this solution serves a certificate that the corresponding distributed NLP problem (130) is infeasible. In this case, the next step is to detect the indices of the infeasible nonlinear constraints which is detected to be infeasible. Considering $\bar{\mathrm{x}} = \bar{\mathrm{x}}_i$, $i = 1,..., N$, as the optimal solution of (144), we can easily detect the nonlinear constraints by computing $g_h(\bar{\mathrm{x}})$, $h = 1,...,m$. Finally, we generate and introduce outer approximations either in the first-order (128) or the second-order (129) form around the point $\bar{\mathrm{x}}$. These outer approximations are then used to update the dual problem represented by (126). According to (FLETCHER; LEY, 1996), if the problem (130) is infeasible, given δ_j^k and $\bar{\mathrm{x}}$, then the outer approximations (128) or (129) effectively remove δ_j^k from the feasible region.

5.8 COMPUTATIONAL EXPERIMENTS AND ALGORITHM EVALUATION

In this section, the accuracy and performance of two algorithms, DiPOA and DiHOA, in solving three problem instances of the SCO problem, specifically the DSLogR, DSLinR, and DSQCQP problems, are evaluated. To validate the optimality of the algorithms, the results of a comparison with state-of-the-art MINLP solvers are presented. We considered SHOT and BONMIN as decomposition-based and KNITRO as nonlinear BnB solvers. The comparison is first performed for DiPOA, with a detailed analysis provided against other MINLP solvers. Subsequently, the comparison is extended to include DiHOA, and solution profiles are presented to assess the overall performance of the algorithms. Both DiPOA and DiHOA algorithms are implemented within the SCOT framework which will be discussed in the next chapter.

5.8.1 Implementation Details and Set-up

All the experiments were performed on a Linux machine with an Intel Core i5 2.50 GHz processor, with four physical cores and 16 GB of RAM. To perform linear algebra operations required by the distributed NLP solver, SCOT uses Eigen 3.4 library. Moreover, the MIP solver employed in SCOT is GUROBI 9.5.2 with an academic license. As for the comparison with other MINLP solvers, GAMS 36.2 was selected as the optimization platform. It should also be noted that to achieve a meaningful comparison, GUROBI is selected as the primary MIP solver for all MINLP solvers considered in the benchmarks. The absolute and relative gap for all algorithms and solvers are chosen to be $\varepsilon = 1 \times 10^{-5}$. For the MINLP solvers included in GAMS, we chose optCA and optCR to have the same value as ε.

5.8.2 Numerical Evaluation of DiPOA

To evaluate `DiPOA` for the DSLogr problem, we generate N random local datasets which are standardized to have zero mean and unit ℓ_2 norm. The response vector Γ is generated according to the logistic function as follows,

$$\Gamma_{i,\ell} = \text{round}\left(\frac{1}{1+\exp(-\theta^T \mathrm{x}_{i,\ell})}\right), \ \forall \ell = 0, \ldots, p,$$

We evaluate the `DiPOA` based on five main scenarios with different problem settings and parameters. Each scenario consists of solving multiple instances of the DSLogR problem. In the first scenario, **SC-I**, we consider a different number of total sample points and evaluate `DiPOA` for the case where only SFP is available and also for the case where both SFP and SoCut features are activated. In the second scenario, **SC-II**, we fix the total number of sample points and change the number of decision variables. Finally, in the last three scenarios, **SC-III** and **SC-V**, we compare fully-featured `DiPOA` with MINLP solvers for relatively small, medium, and large problem instances. Each scenario consists of different settings which are provided in Table 6. In this table, `total-samples` refers to the number of data points, `num-var` and `num-nodes` denote the number of variables and nodes, respectively.

Table 6 – Scenario settings

`scenario`	`total-samples`	`num-var`	`sparsity` [%]	`num-nodes`
SC-I	[$2k$ .. $50k$]	20	25	10
SC-II	$50k$	40 – 200	25	10
SC-III	$10k$	20	[10 .. 90]	10
SC-IV	$100k$	200	[10 .. 90]	10
SC-V	$300k$	300	[10 .. 90]	10

The numerical results from **SC-I** appear in Table 7. In this case, `DiPOA` is evaluated according to two different settings. In the first setting (`dipoa-sfp` column), `DiPOA` only generates linear cuts, and only SFP is activated, whereas, in the second setting (`dipoa-soc` column), SoCuts can be generated. It can be seen in Table 7 that second-order cuts can drastically improve the algorithm convergence in terms of wall-clock time and the number of cuts. Moreover, we observe that as the number of all samples increases, the number of FOC cuts and therefore the wall-clock time decreases. In other words, a large number of sample points often lead to a smaller number of iterations, however, with more computational complexity per iteration. Considering the centralized architecture, although the number of iterations can be small, the computational burden is larger than the distributed case since the entire dataset is used in the evaluation of the objective function. This behavior can be seen in Figure 24 where the scalability of the `DiPOA` is evaluated. This figure compares the wall-clock time between `DiPOA` and

Table 7 – Numerical Results for SC-I (Varying Number of Samples)

	dipoa-sfp				dipoa-soc			
`m`	`time`	`nfoc`	`mip`	`nlp`	`time`	`nsoc`	`mip`	`nlp`
$2k$	600	1510	439.39	160.60	0.947	20	0.79	0.14
$6k$	600	780	439.39	160.60	1.326	10	1.10	0.21
$10k$	336.141	430	246.16	89.97	1.591	10	1.32	0.26
$14k$	268.325	361	196.49	71.82	1.585	10	1.32	0.26
$18k$	186.343	270	136.464	49.87	1.681	10	1.40	0.27
$22k$	152.284	200	111.52	40.761	1.775	10	1.48	0.29
$26k$	147.808	180	108.23	39.56	1.814	10	1.51	0.29
$30k$	122.878	170	88.75	34.12	1.918	10	1.60	0.31
$34k$	120.662	160	87.15	33.50	1.886	10	1.57	0.31
$38k$	118.914	150	85.88	33.02	2.046	10	1.70	0.33
$42k$	86.67	110	62.60	24.06	1.874	10	1.56	0.30
$46k$	11.041	30	11.04	3.06	2.286	10	1.90	0.37
$50k$	11.413	30	8.24	3.16	2.598	10	2.17	0.42

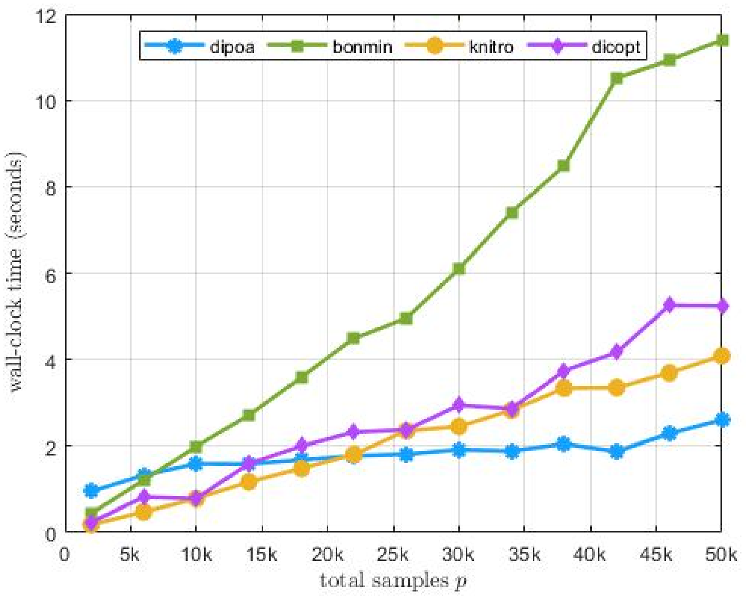


Figure 24 – Comparison between DiPOA, BONMIN, KNITRO, and DICOPT

BONMIN, KNITRO, and DICOPT which are centralized MINLP solvers. Compared to the centralized solvers, for the problem instances with a relatively small number of data points ($p \leq 6k$), `DiPOA` needs more time to provide the solution. In contrast, as we increase the size of the dataset, `DiPOA` behaves more robustly and needs a smaller execution time. Hence, `DiPOA` can scale well for problem instances with much larger data sets. We also note that SHOT was able to successfully solve only the first instances ($2k$, $6k$, and $10k$) of problem **SC-I**, while for larger instances and other scenarios it reaches the cut-off time of 600 seconds. For the first and second instances (where numbers of total samples are $2k$ and $6k$) the solution time is 57.86 and 286.63 seconds. For the third case with 10*k* as the number of total samples, SHOT converged in 536.10 seconds. Since SHOT failed to solve the larger problem instances within the considered cut-off time, we omit its results in the figures and tables.

Table 8 provides the numerical results for **SC-II**. In this case, the number of data points is fixed to be 50*k* and `DiPOA` is evaluated concerning a varying number of variables, which runs with both SFP and SoCuts activated. These results show that `DiPOA` is robust to the increase of variables.

The numerical results of **SC-III** are presented in Table 9. The experiment aims to analyze the sensitivity of DiPOA for the number of non-zero elements in the solution (bound κ). Since the problem size is small, the centralized solvers outperform DiPOA in terms of execution time.

Finally, Table 10 presents the numerical results for medium and large-scale scenarios (*i.e.*, **SC-IV and SC-V**). In these scenarios, the centralized solvers were not capable of providing a solution for the problem instances. The properties of the problem instances are defined in Table 6 for each scenario.

Additionally, we investigate the impact of SoCuts and their event-triggered scheme on the solution of the master's problem. Figure 25 illustrates the impact on the upper and the lower bounds in the context of the DSLogR problem. In particular, we allow

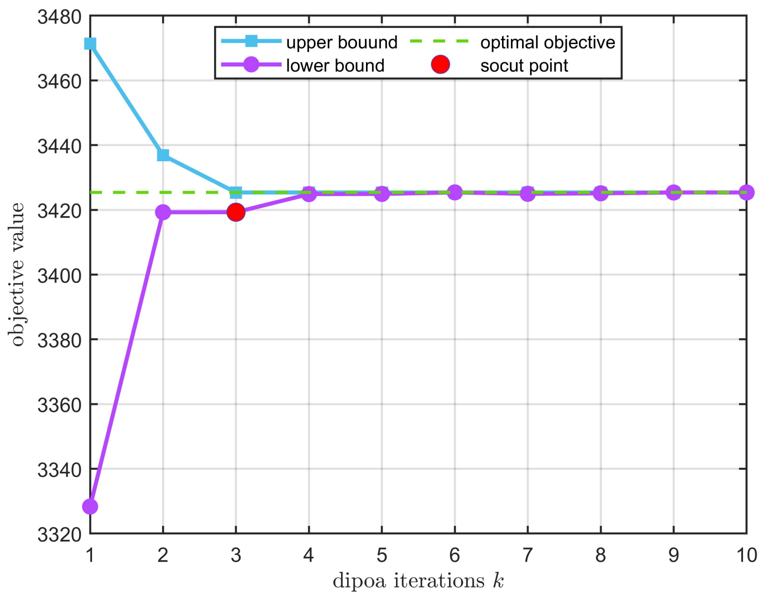


Figure 25 – The impact of the SoCuts and the event-triggered scheme.

DiPOA to perform 10 iterations for a small instance of the DSLinR problem with $\kappa = 5$, $\theta = 10$, and $p = 2000$. In Figure 25, purple circles indicate the lower bound and the black squares show the upper bound generated by DiPOA. The red circle is the point at which quadratic cuts are generated. As the figure depicts, the SoCuts improve the convergence of the algorithm by inducing a higher lower bound. Moreover, the event-triggered scheme prevents the algorithm from generating a high number of SoCuts. For

Table 8 – Numerical Results for SC-II (Varying Number of Variables)

	dipoa		bonmin		knitro		dicopt	
`num-var`	`time[sec]`	`rel-gap[%]`	`time[sec]`	`rel-gap[%]`	`time[sec]`	`rel-gap[%]`	`time[sec]`	`rel-gap[%]`
40	2.446	0.06	25.561	0.12	11.637	0.012	17.188	0.22
60	2.651	0.051	59.001	0.09	20.335	0.013	22.900	0.087
80	13.872	0.003	97.167	0.13	30.336	0.063	32.403	0.31
100	14.321	0.005	136.693	0.18	40.33	0.036	55.151	0.02
120	15.005	0.005	203.814	0.02	49.354	0.063	60.003	0.03
140	15.537	0.009	272.458	0.19	56.332	0.055	114.103	0.06
160	16.315	0.006	336.524	0.15	67.356	0.041	117.705	0.08
180	17.189	0.003	390.053	0.36	76.387	0.032	164.234	0.03
200	18.002	0.007	463.526	0.45	85.569	0.045	228.631	0.08

example, in this problem instance, only one SoCut is generated, preventing the master's problem from becoming overly complex with the quadratic constraints. Although this

Table 9 – Numerical Results for SC-III (Varying Number of Non-Zeros)

	dipoa		bonmin		knitro		dicopt	
κ	time[sec]	rel-gap	time[sec]	rel-gap	time [sec]	rel-gap	time [sec]	rel-gap
2	2.927	0.075	2.115	0.15	0.861	0.12	1.095	0.12
4	2.746	0.120	1.970	0.061	0.860	0.204	1.119	0.20
6	2.862	0.160	1.950	0.102	0.866	0.21	1.192	0.16
8	3.164	0.150	2.021	0.23	0.783	0.13	1.365	0.12
12	2.816	0.249	2.024	0.12	0.673	0.132	1.157	0.18
14	2.936	0.200	2.040	0.10	0.715	0.153	1.490	0.16
16	2.729	0.250	2.034	0.17	0.656	0.106	1.021	0.13
18	2.459	0.173	2.094	0.09	0.686	0.151	1.156	0.15

procedure provides a better lower bound while controlling the complexity of the master's problem, it remains an unknown function of the problem data.

Finally, numerical results comparing DiPOA to the Centralized Outer Approximation (COA) of (BERTSIMAS; KING, 2017b) are provided in Table 11. The same settings from (BERTSIMAS; KING, 2017b) are adopted. As the table shows, the DiPOA algorithm outperforms the centralized architecture and solves the problem instances in significantly less computational time.

Table 10 – Numerical Results for SC-IV and SC-V (Medium and Large Scenarios)

(a) **SC-IV**

κ	`time[sec]`	`gap[%]`
20	6.608	0.11
40	6.787	0.16
60	6.945	0.14
80	6.550	0.17
100	7.078	0.20
120	6.595	0.10
160	6.601	0.11
180	5.586	0.22

(b) **SC-V**

κ	`time[sec]`	`gap[%]`
20	18.699	0.03
60	18.424	0.05
100	18.598	0.06
140	18.402	0.11
180	17.618	0.092
220	18.785	0.097
240	18.582	0.095
260	18.201	0.093

Table 11 – Numerical Results Comparing DiPOA with Centralized OA for $\kappa = 5$.

`total-samples`	`num-var`	`num-nodes`	`DiPOA-time` [sec]	`COA-time` [sec]
100	10	4	0.077	< 1
1*k*	100	4	2.581	15
2*k*	200	4	3.298	16

5.8.2.1 Sparse Quadratically Constrained Quadratic Programming Problem

In this section, we provide numerical evaluation results for the SQCQP problem, which is a SCO problem with quadratic objective and constraint functions, namely:

$$\min_{\mathrm{x}\in\mathcal{R}^n} \sum_{i=1}^{N} \left(\frac{1}{2}\mathrm{x}^T Q_i \mathrm{x} + q_i^T \mathrm{x} + d_i\right) \tag{145)a}$$

$$s.t.\ \frac{1}{2}\mathrm{x}^T P_h \mathrm{x} + c_h^T \mathrm{x} + r_h \leq 0,\ \forall h = 1,\dots,m \tag{145)b}$$

$$\mathrm{x} \in \Omega \tag{145)c}$$

$$\|\mathrm{x}\|_0 \leq \kappa \tag{145)d}$$

In this experiment, two main scenarios with different problem sizes are considered. The first scenario has $n = 100$ variables while the second has $n = 200$. In both scenarios, we evaluate DiPOA according to a different number of nonzero elements κ. The numerical results of the first scenario appear in Table 12. It can be noticed that, by decreasing the degree of the solution sparsity, the optimal objective value also decreases. This behavior is expected since, by reducing the sparsity of the solution, more variables can appear in the solution and therefore the optimal objective approaches the value of the dens problem (*i.e.*, problem (112) without sparsity constraint). The number of cuts and hence the execution time also changes depending on the sparsity of the solution. In particular, as the number of nonzero variables κ increases, more iterations are needed by the DiPOA algorithm to converge. However, this phenomenon occurs until $\kappa = \kappa_{\max}$, where $\kappa_{\max}$ is the largest number of nonzero variables for which the

Table 12 – Numerical Results for SQCP with $n = 100$ Variables

κ	`objective-value`	`rel-gap[$\%$]`	`total-cuts`	`time[sec]`
5	10.994	0.131	30	4.64
7	8.601	0.143	80	14.96
8	7.457	0.142	180	45.388
9	6.378	0.146	200	57.519
10	5.141	0.146	270	94.68
20	–5.759	0.139	250	81.962
30	–15.411	0.144	90	21.986
40	–23.576	0.127	50	6.604
50	–29.733	0.144	50	9.554
60	–34.529	0.101	40	9.462
70	–37.885	0.122	30	8.921
80	–39.208	0.060	30	8.712
90	–39.595	0.054	30	8.699

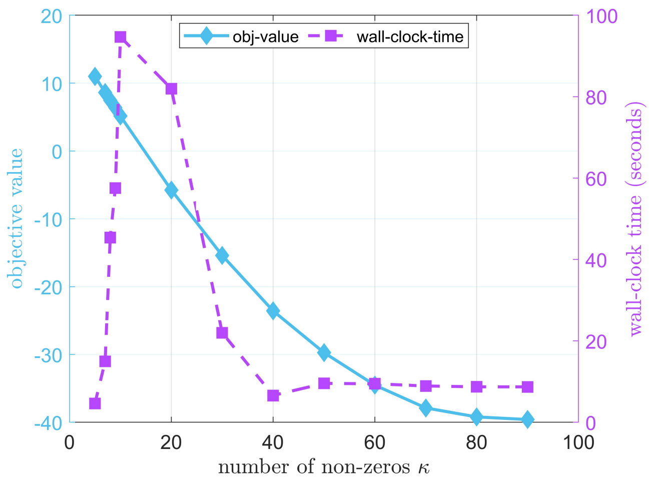


Figure 26 – Objective value and wall-clock time for different values of κ for $n = 100$; the left axis shows the behavior of the optimal objective value and the right axis represents the execution time.

highest number of iterations is needed. For $\kappa > \kappa_{\max}$ the complexity of the problem decreases. The behavior of the objective value and also the solution time is illustrated in Figure 26, according to the sparsity of the solution. For this problem instance, the figure shows that $\kappa_{\max} = 10$, at which sparsity-level the wall-clock time is 94.68 s and the total number of cuts is 270. Finally, the numerical results of the second scenario are provided in Table 13. As mentioned before, in this case, we scale the problem and set $n = 200$. The behavior of the objective value and the wall-clock time is illustrated in Figure 27. As expected, a similar behavior emerges from the application of DiPOA, however, a large computational time is needed in some instances.

We also compared DiPOA against Gurobi for solving the convex SQCQP. In most problem instances, Gurobi (GUROBI OPTIMIZATION, 2020) achieved better per-

Table 13 – Numerical Results for SQCQP with $n = 200$

κ	`objective-value`	`rel-gap[$\%$]`	`total-cuts`	`time[sec]`
5	11.001	0.14	240	104.768
10	4.816	1.34	810	600
20	–7.427	1.27	880	600
30	–19.397	0.11	740	543.268
40	–30.802	0.14	80	27.133
60	–50.985	0.14	20	8.021
100	–81.291	0.12	40	12.498
140	–97.308	0.05	50	26.952

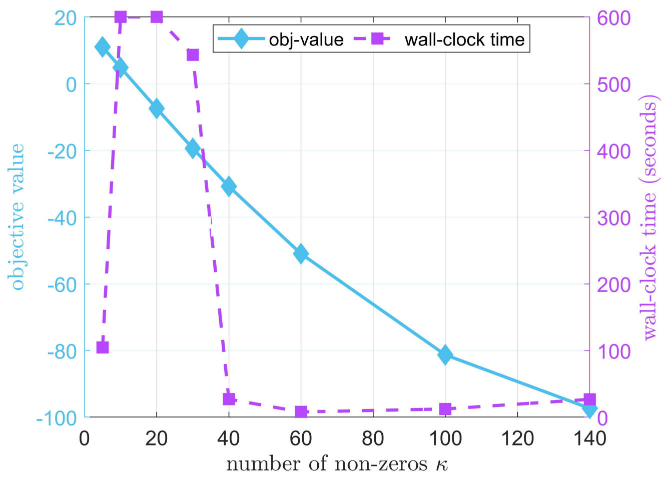


Figure 27 – Objective value and wall-clock time for different values of κ for $n = 200$; the left axis shows the behavior of the optimal objective value and the right axis represents the execution time.

formance for the centralized case in terms of wall clock time, which is expected considering the communication and computing overhead of DiPOA. Regarding the distributed setup, Gurobi like other centralized solvers cannot be directly applied because the objective-defining data is private to each node and spread over the CN.

In this section, we evaluated the performance of DiPOA for DSLinR and SQCQP problems. We considered 70 problem instances under different scenarios and settings. According to the numerical experiments, DiPOA performance and efficiency were acceptable for both problems in most situations. The numerical results and comparison with state-of-the-art MINLP solvers also showed that, for the conditions when distributed computations are inevitable, DiPOA provides a reliable and optimal solution.

As a general conclusion, the separability of the optimization problem can be used by distributed algorithms to speed up the computation by splitting the computational burden among several processors. Essentially, each processor would be responsible to solve a portion of the problem independent of other processors and in parallel. For small-scale problems, however, a distributed algorithm might not necessarily outperform

Table 14 – MINLP Solver Settings

	Settings			
solver	algorithm	name	MIP solver	NLP solver
SCOT	DiPOA	SCOT-MT	GUROBI	RHADMM
SCOT	DiPOA	SCOT-ST	GUROBI	RHADMM
SHOT	ESH multiple-tree	SHOT-MT	GUROBI	IPOPT
SHOT	ESH single-tree	SHOT-ST	GUROBI	IPOPT
BONMIN	B-OA	BONMIN-MT	GUROBI	IPOPT
BONMIN	B-QG	BONMIN-ST	GUROBI	IPOPT
KNITRO	BnB	KNITRO	–	KNITRO

centralized algorithms taking into account process synchronization and inter-process communication. A distributed algorithm can be a suitable choice in scenarios where either the problem is inherently distributed or large-scale. The former case refers to the problems where the data is spread over a large network and data privacy matters. In this case, because of the data privacy and the size of the network, it is not usually practical to process the problem data in a central node. In the latter case, due to hardware limitations, it is difficult (if not impossible) to process the entire problem data in a single computational node and a distributed algorithm over multiple nodes can be useful.

5.8.3 Numerical Evaluation of DiHOA

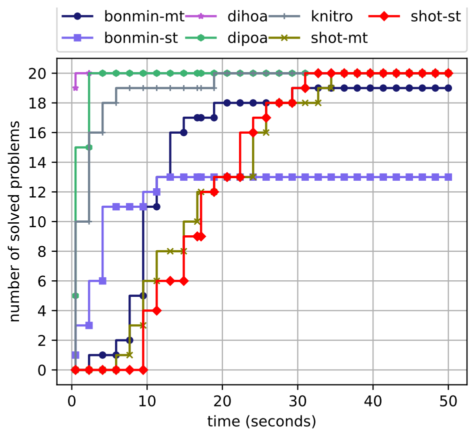
This section presents the numerical evaluation of the `DiHOA` algorithm compared to state of the art MINLP solvers as well as the `DiPOA` algorithm. We generate N random local datasets for both DSLogR and DSLinR problems with zero mean and unit ℓ_2 norm for each column. We perform the numerical benchmarks based on different solver settings by means of solution profiles to compare `SCOT` with several MINLP solvers with different settings. Default settings were adopted for `KNITRO` as an NLP BnB solver. The chosen settings for `SCOT`, `SHOT`, and `BONMIN` are reported in Table 14.

5.8.3.1 Benchmark Results

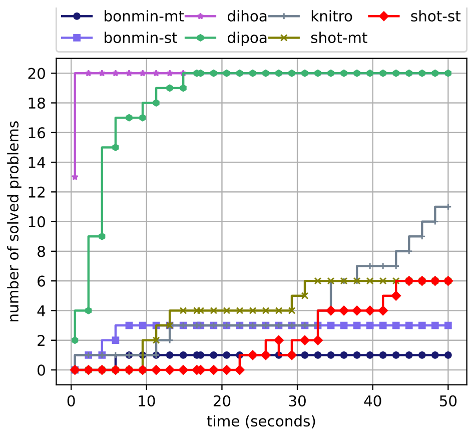
We considered seven benchmark scenarios containing 20 different problem instances with different properties and settings. In each scenario, we generate 15 DSLogR and 5 DSLinR random problem instances with a different number of features and sample points within a given range. Therefore the benchmark set consists of a total of 140 problem instances. Moreover, each algorithm appearing in Table 14 is applied to solve all problem instances with 30 different maximum execution times limits, starting from 0.5 to 50 seconds. Therefore, the total number of algorithm runs for each scenario is 600, leading to 4200 algorithm executions for all scenarios. Table 15 represents settings for each benchmark scenario where n_{min} and n_{max} are the minimum and

Table 15 – Benchmark Settings for solution profiles

	Scenario settings						
scenario	$n_{\min}$	$n_{\max}$	$p_{\min}$	$p_{\max}$	N	n_p	p_{tot}
1	20	30	1000	2500	2	20	5000
2	20	30	1000	5000	2	20	10000
3	25	50	1000	5000	2	20	10000
4	25	100	1000	10000	4	20	40000
5	25	100	1000	20000	4	20	80000
6	25	100	1000	50000	4	20	200000
7	25	200	1000	50000	6	20	300000



(a) 90% sparsity.



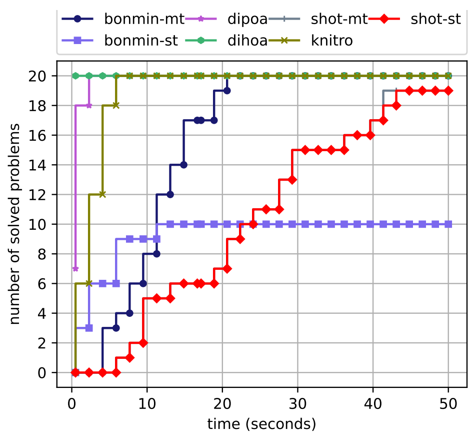
(b) 80% sparsity.

Figure 28 – Benchmark results for scenario 1.

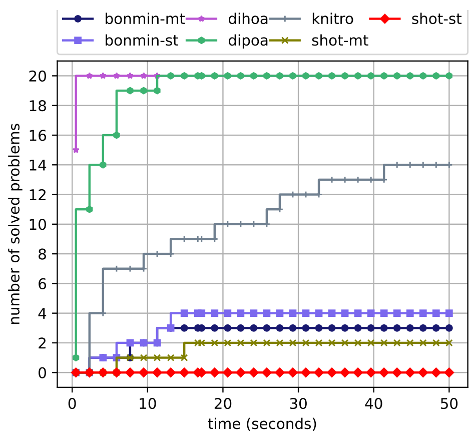
the maximum number of features, $p_{\min}$ and $p_{\max}$ are the minimum and the maximum number of data points for each computational node, n_p is the total number of problems, and p_{tot} is the total number of data points considering all computational nodes. We assumed 80% and 90% sparsity for each scenario presented in Table 15. However, since the number of variables is generated randomly for each problem instance, the value of κ may vary. Nonetheless, to ensure the reproducibility of the numerical results, we fixed the random seed for each scenario. Figures 28-34 compare the solvers `SCOT`, `BONMIN`, `SHOT`, and `KNITRO` with both single-tree and multiple-tree algorithms. The comparison results for each scenario are shown as solution profiles consisting of two different sparsity levels of the solution.

The benchmark results of the first three scenarios are depicted in Figures 28-30 where *small to medium size* problem instances are considered. In both sparsity levels, the DiPOA and DiHOA algorithms show better performance compared to other MINLP solvers. It can be observed in Figure 30 that for larger problem instances the performance gap between `SCOT` and other MINLP solvers increases.

Figures 31-34 depict the solution profiles for the scenarios 4-7 in which *medium to large* problem instances are taken into account. In these scenarios, all MINLP solvers

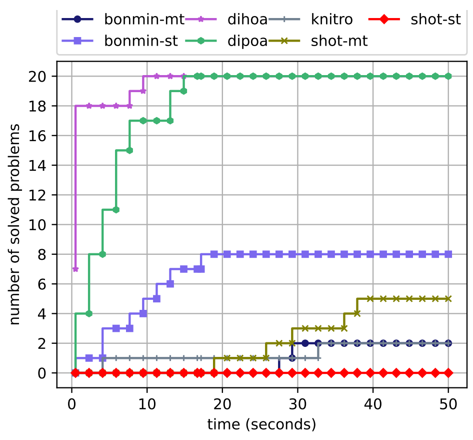


(a) 90% sparsity.

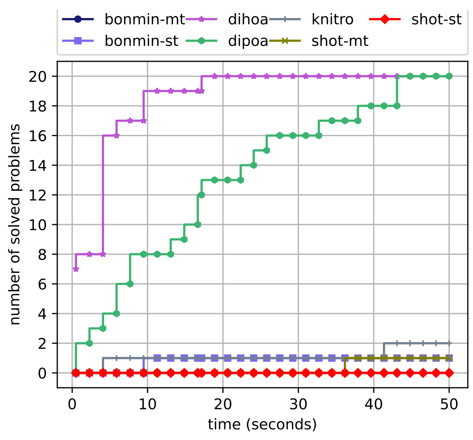


(b) 80% sparsity.

Figure 29 – Benchmark results for scenario 2.

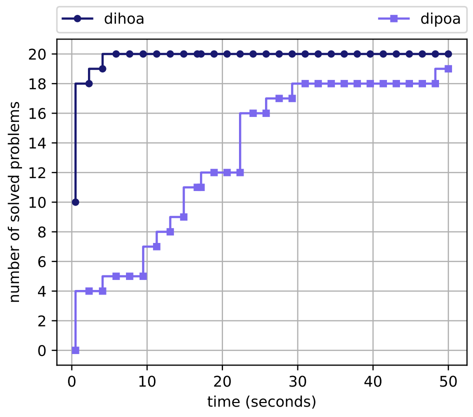


(a) 90% sparsity.

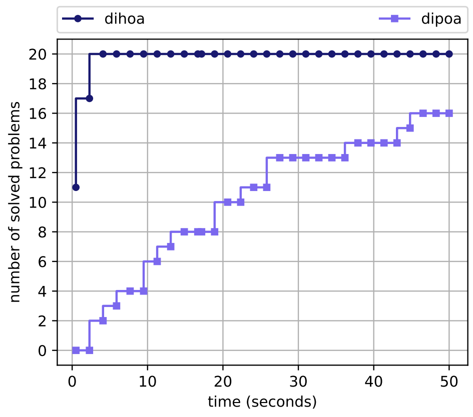


(b) 80% sparsity.

Figure 30 – Benchmark results for scenario 3.

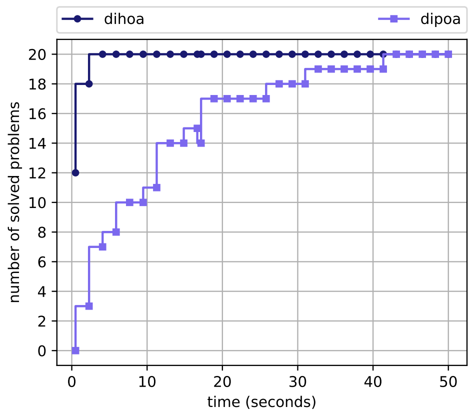


(a) 90% sparsity.

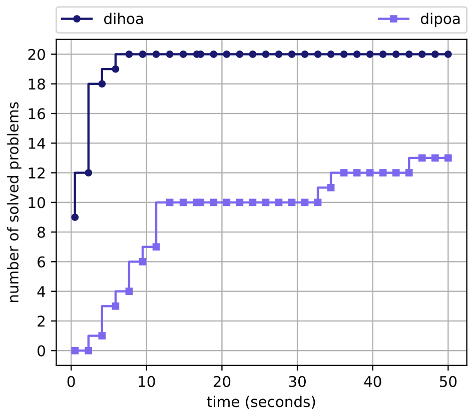


(b) 80% sparsity.

Figure 31 – Benchmark results for scenario 4.

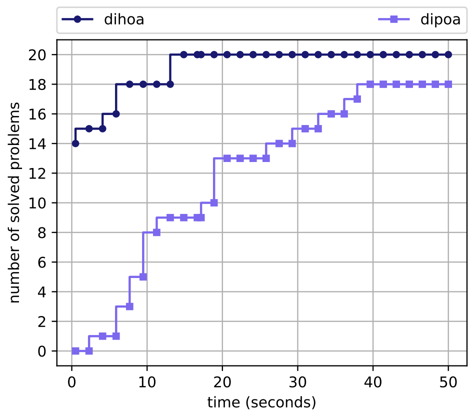


(a) 90% sparsity.

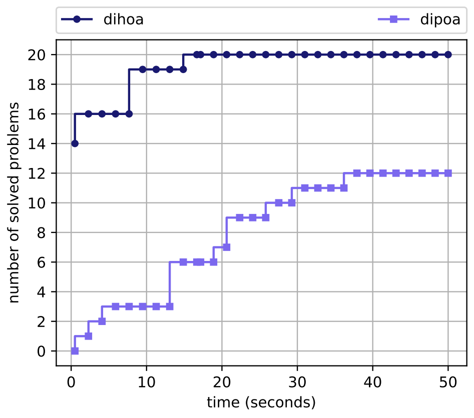


(b) 80% sparsity.

Figure 32 – Benchmark results for scenario 5.

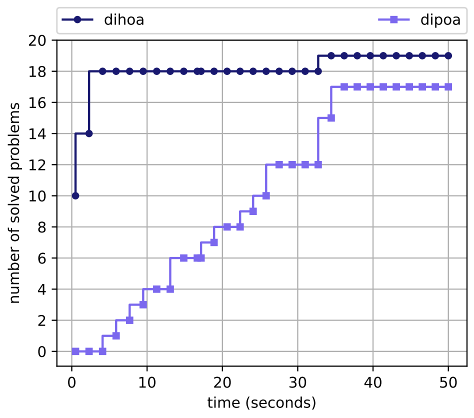


(a) 90% sparsity.

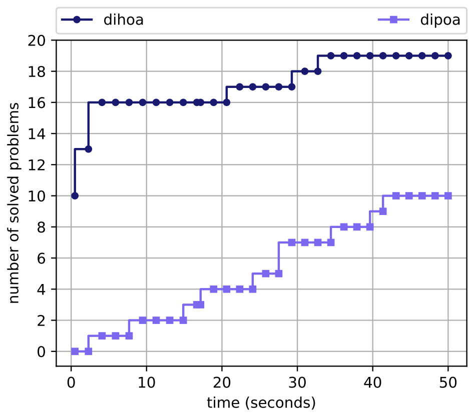


(b) 80% sparsity.

Figure 33 – Benchmark results for scenario 6.



(a) 90% sparsity.



(b) 80% sparsity.

Figure 34 – Benchmark results for scenario 7.

Table 16 – Computational results with different sparsity modeling strategies

	Strategy CPU Time (s)		
n	Big-M	SOS-1	Hybrid
40	0.0943	0.0993	0.0923
80	0.2741	1.6421	0.2399
120	0.5374	7.6315	0.6458
160	0.9063	21.688	0.9484
200	2.7449	44.673	2.9086

failed to solve the problem with the considered maximum execution time limit. Therefore, we only provide solution profiles of DiPOA and DiHOA algorithms. According to the solution profiles, the DiHOA algorithm is more efficient compared to DiPOA as the performance gap between them is increasing for large problem instances. For example, in scenario 7 with 80% of sparsity, DiHOA was able to solve 19 problem instances within the given maximum execution time limit, whereas DiPOA only solved 10 instances. According to the numerical experiments and solution profiles, the DiHOA algorithm achieves better performance and efficiency in all problem instances. However, one should keep in mind that `SCOT` is tailored for SCO problems which is not the case for general-purpose solvers. The results also showed that for large and distributed problems a distributed solver can provide an efficient and reliable solution.

5.8.4 Evaluation of Sparsity Modeling Strategies

We assessed the efficacy of the `DiHOA` algorithm by implementing methodologies to model the sparsity constraint. We investigate three distinct strategies that were previously presented: Big-M, SOS-1, and the hybrid approach, across a varying number of features denoted by n and a fixed number of rows $p_{tot} = 10000$, while limiting the number of non-zero elements to $\kappa = \lfloor \frac{n}{2} \rfloor$. The average computational time for 100 randomly generated DSLogR problems for distinct n appears in Table 16. The table shows that as the problem size n increases, the computational time required to solve the problems increases for all three strategies. However, the Big-M and hybrid approaches require less computational time than the SOS-1 approach for all problem sizes. Specifically, the Big-M approach has the shortest computational time across all problem sizes, followed by the hybrid approach. These results suggest that for solving the problem set considered in this chapter, the Big-M and hybrid sparsity modeling strategies are more efficient than the SOS-1 strategy. However, the choice of strategy may depend on the specific problem characteristics. For example, the SOS-1 strategy may be more appropriate for problems with unknown Big-M values. If both SOS-1 and Big-M constraints are present in the optimization problem, then a sophisticated MIP solver, such as GUROBI, will avoid forming redundant constraints in preprocessing

Table 17 – Computational time comparison of DiHOA and DiPOA with state-of-the-art MINLP solvers to train a DSLogR model with the simulated electrical grid stability data.

	SCOT (s)		MINLP Solvers (s)				
κ	DiHOA	DiPOA	SHOT-ST	SHOT-MT	BONMIN-ST	BONMIN-MT	KNITRO
2	3.591	6.236	17.83	16.13	2.064	12.67	17.01
4	3.031	7.415	71.69	47.96	5.211	51.37	61.74
6	9.940	14.22	95.06	43.21	20.77	32.09	46.71
8	20.86	36.97	67.53	34.71	13.13	21.63	11.04
10	3.922	9.256	39.89	32.07	11.25	4.743	7.851

while maintaining a tight bound in the optimization problem.

5.8.5 Real-world Dataset

This section evaluates the performance of SCOT algorithms in a real-world scenario using a smart grid stability dataset to train a DSLogR model for binary classification aimed at predicting the grid stability status. The dataset used here is obtained from the *UCI Machine Learning Repository* (DUA; GRAFF, 2017) and is referred to as the *Electrical Grid Stability Simulated Data*. The dataset contains 10000 observations and 13 features, including variables such as the reaction time of participants, nominal power consumed, and price elasticity. We refer the reader to (ARZAMASOV; BÖHM; JOCHEM, 2018) for a more comprehensive dataset description. To leverage the benefits of distributed machine learning, we adopted a partitioning strategy to distribute the dataset over a network of four nodes, each containing 2500 observations. This data distribution allows us to utilize the SCOT algorithm for training the DSLogR model.

We trained the DSLogR model using a range of values for κ. Subsequently, we assessed the computational time of DiHOA and DiPOA against the MINLP solvers mentioned in the preceding sections. The results are summarized in Table 17, which presents the computational time of each algorithm measured in seconds. The results provide evidence of the effectiveness of the DiHOA algorithm in handling real-world data. It is evident that DiHOA outperforms general-purpose MINLP solvers, especially when κ is large. However, it is noteworthy that general-purpose solvers perform relatively well, especially when κ is small. This observation can be attributed to the fact that the dataset used in this study is not as large as the synthetic dataset used in previous sections. Additionally, we note that utilizing a distributed algorithm may not always provide significant benefits, particularly when dealing with relatively small datasets. However, a distributed algorithm may still be suggested if the dataset is inherently distributed across multiple nodes.

6 SPARSE CONVEX OPTIMIZATION TOOLKIT (SCOT)

In this chapter, we introduce Sparse Convex Optimization Toolkit (SCOT) and its technical details. These details will include the architecture and design, main components, implementation details, basic syntax, and usage of the toolkit. `SCOT` is an open-source distributed optimization solver [1] that is designed to solve Sparse Convex Optimization (SCO) problems over computational networks. It adopts a mixed-integer approach that enables the identification of exact solutions to SCO problems. The `SCOT` framework is designed to be modular and task-oriented, with most functionalities encapsulated in specific tasks. These tasks are executed in a sequence defined by the algorithms, but the order of execution can also be modified at runtime. This approach makes it easy to modify or expand the behavior of the solver by defining solution strategies that include a list of specific tasks to be performed in either sequential or parallel order. The algorithms proposed in this thesis are all implemented using a task-based strategy, which is discussed in this chapter.

6.1 INTERNAL ALGORITHMS AND EXTERNAL DEPENDECSIES

`SCOT` is mainly designed based on the primal/dual strategy for solving convex MINLP problem presented in the previous chapter. However, it also provides convenient interfaces to access `RH-ADMM` and `DiSGT` algorithms introduced in Chapter 4 and 3 respectively. A full list of algorithms implemented in `SCOT` is provided in Table 18. `SCOT` core module is entirely written in `C++17` and uses a variety of external dependencies that are provided in Tabel 19. The first four libraries listed are optimization solvers which `SCOT` interfaces. IPOPT is an interior point optimizer used to solve local nonlinear programming problems, while Cbc is an open-source MILP solver. Gurobi, on the other hand, is a commercial optimization solver that supports linear programming, mixed-integer programming, and quadratic programming. OSQP is an operator splitting quadratic program solver for convex optimization. The last three libraries listed are supporting libraries for logging, formatting, and argument handling. spdlog is a fast C++ logging library, fmt is a modern C++ formatting library, and Argh is a C++ header-only argument handler library.

6.2 ARCHITECTURE

A high-level overview of `SCOT` architecture, its main layers, and components are shown in Figure 35. As observed in the figure, the primary layers of the framework are `SCOTPY`, `SMCLI`, `SCOT Solver`, and `Computing Network`, each of which consists of different tools and components to build and solve the optimization problem. The layered

[1] `https://github.com/Alirezalm/scot`

Table 18 – SCOT Algorithms

Algorithm	Description
DiPOA	Multiple-tree algorithm for solving SCO problems.
DiHOA	Hybrid multi-single tree algorithm for solving SCO problems.
RH-ADMM	Distributed solver for solving convex sub-problems.
DiSGT	Consensus augmented Langrangian method solve problem (30).
TN	Truncated Newton's Method for solving unconstrained local NLPs.

Table 19 – External Libraries Used in SCOT

Name	Description
IPOPT	Interior point optimizer to solve local NLPs.
CBC	Open-source MILP solver
Gurobi	Commercial MIP Solver
OSQP	Operator Splitting Quadratic solver.
MPI	Message Passing Interface for parallel computing
spdlog	C++ logging library
fmt	Modern C++ formatting library
Argh	C++ argument handler library

architecture of `SCOT` leads to a highly modular framework that can be easily extended by new features and algorithms. In the following, we describe each layer of `SCOT` and its components.

6.2.1 SCOTPY

`SCOTPY` is the main API written in `Python 3.8`, which provides various modules and classes to define the optimization problem and solver settings. This layer consists of four components, namely, Problem Reformulation, Problem Parser, Solver Setting, and Input/Output File generators. In principle, `SCOTPY` receives the problem input and algorithm settings from the application code layer and, after parsing the input data and settings, writes the optimization problem and settings in `JSON` format with a specific naming convention. Moreover, according to the number of nodes given by the user, N objective functions with different problem data are created and stored on the file system as different `JSON` files.

Therefore, the resulting computation of this layer is to express the optimization problem and solver settings in various `JSON` files that can be read by the subsequent layers. Representing the optimization problem using files provides more flexibility since it decouples the optimization model from the optimization solver. Therefore, it is possible to write the optimization model in any language of choice and call the optimization solver from a different programming language or framework. Coupling the optimization model and the optimization solver through file formats is well-known in the optimization

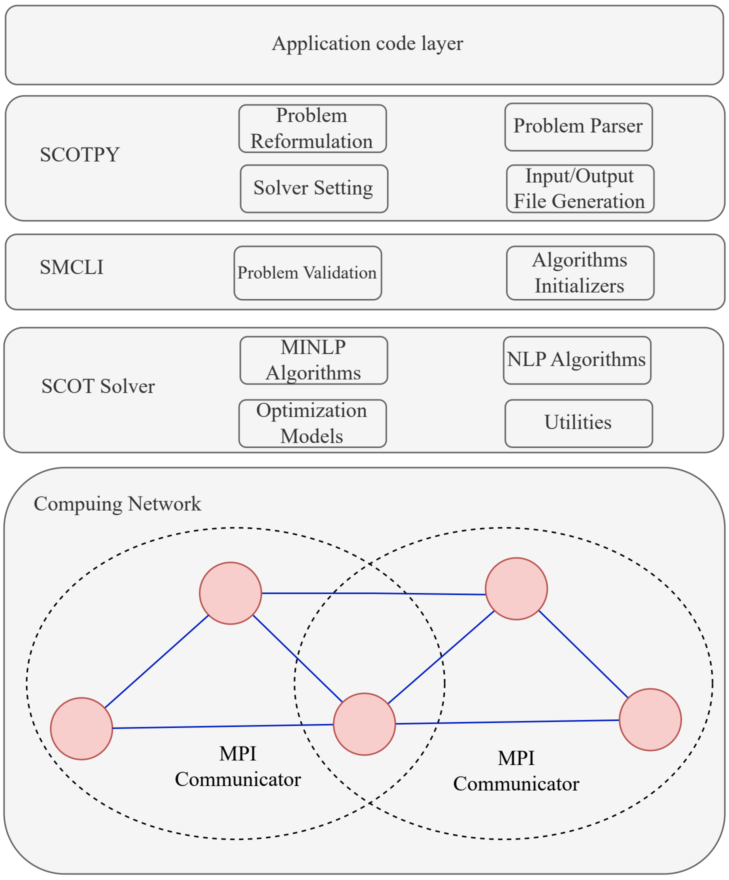


Figure 35 – SCOT Architecture

software industry and has been widely used for decades. We refer the interested reader to (LEGAT et al., 2021) for more details.

6.2.2 SMCLI

SCOT MPI Command Line Interface (SMCLI) is the main layer utilized to directly execute `SCOT Solver` according to different input and setting files. `SMCLI` can be used directly without `SCOTPY` interface, however the problem definition using `SCOTPY` is more appropriate. The Problem Validation component of `SMCLI` is responsible for validating the problem input and settings files, providing suitable data structures containing the optimization problem data for the `SCOT Solver`. Additionally, initializing computational nodes and various software libraries used in `SCOT Solver` are among the responsibilities of `SMCLI` layer.

6.2.3 SCOT Solver

At its core, the `SCOT` framework consists of`SCOT Solver` layer which is responsible to solve the optimization problem using a proper algorithm and settings. The main components of this layer are MINLP Algorithms, NLP Algorithms, Optimization Models, and Utilities which are discussed in this section.

The NLP Algorithms component consists of various modules and classes to distributedly solve the primal problem (130) and deliver the primal solution to the MINLP algorithms. Owing to its flexible and modular implementation, the NLP Algorithms component can be easily extended by introducing user-defined and custom-distributed convex optimization algorithms and solvers. In principle, this component requires the solution of local NLP problems for which multiple open-source and commercial solvers are available. At its core, NLP Algorithms consists of a sub-module, that provides a flexible interface to third-party solvers that can solve local optimization problems. The supported solvers are `OSQP`, `IPOPT`, and `Gurobi`. As a final note, the NLP Algorithms component is called by all internal algorithms of `SCOT` and handles most of `MPI` communications and collective operations. More importantly, the quality of outer approximations depends on this component since it provides feasible points around which nonlinear functions are approximated.

One of the most critical components of `SCOT Solver` is the MINLP algorithms component that implements Algorithm 4 and DiPOA. The MINLP algorithms component consists of two main modules, namely `dipoa`, and `dihoa`, which are responsible for implementing their corresponding algorithm. Because all the implemented algorithms must manage and monitor outer approximations, the MINLP algorithms component also includes an Cut Managers module. This module implements several classes to support the necessary data structures that generate and store both first- and second-order outer approximations. Moreover, the Cut Managers module implements the event-triggered schemes to improve the outer approximations' quality and switch from `MultipleTreeSearch` to `SingleTreeSearch` strategy. Among all classes, the Cut Managers module consists of two important classes, namely `CutStorage` and `CutGeneration`. `CutStorage` provides a simple way to validate and store the linear and quadratic outer approximations. The primary responsibility of the `CutGeneration` class is to generate necessary outer approximations from the information received from the NLP Algorithms component. The MINLP Algorithm module also provides a flexible functionality to interface third-party MIP solvers such as `Gurobi` and `Cplex` for solving the MIP dual problem (126).

The `model` component's primary responsibility is to generate a concrete internal reformulation of the optimization problem to be used by algorithms. This component consists of various classes to present different types of nonlinear objective functions and linear and sparsity constraints. By accessing the `model` component, the `algorithm` will be able to access the optimization problem data whenever necessary during the computations.

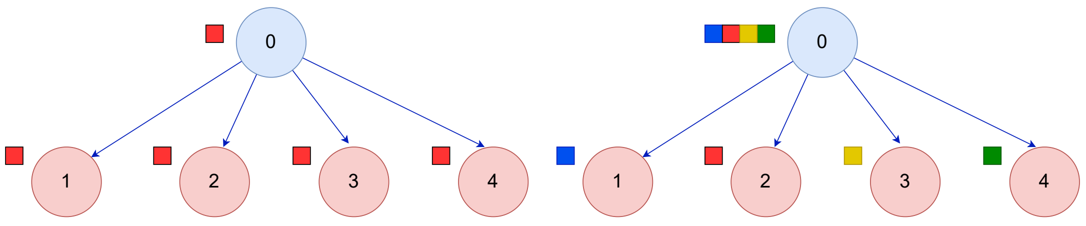
Finally, the `utilities` module implements classes and functions to provide commonly required functionalities, such as low-level parsing, measuring the CPU time, writing logs, constant parameters, exceptions, and file handling functions.

6.2.4 Computing Network

This layer is responsible for presenting and managing the computational network using a graph data structure and Message Passing Interface (MPI) library. The `computing network` layer is in tight communication with the `SCOT Solver` layer since all distributed algorithms use MPI for performing distributed computations and interprocess communications.

6.3 DISTRIBUTED PROGRAMMING AND MESSAGE PASSING INTERFACE (MPI)

Taking into account the computer hardware structure available today, there exist three levels of parallelism that can be utilized. The lowest level of the hierarchy of parallelism regards *vectorization* which supports the *single instruction multiple data* programming style. In this case, a limited number of instructions can be performed per CPU cycle in parallel. Most of the compilers of the high-performance programming languages such as `C/C++` and `FORTRAN` support vectorization inherently. The next level of parallelism consists of *multi-threading*, which allows the programmer to perform *shared memory* parallel computations using different threads on a single machine. Finally, on the highest level of the parallelism hierarchy, the Message Passing Interface (MPI) is implemented (GROPP; LUSK; SKJELLUM, 1999) to allow *distributed computations* on a *distributed memory* machine, such as high-performance computer clusters. As the main benefit, MPI supports *data-passing* between various computing nodes, where the local computations are performed. Considering that each computing node in MPI can perform multi-threaded computations, it is possible to use MPI and multi-threaded programming at the same time. This style of programming related to distributed computation is called the *hybrid programming paradigm*. In this chapter, we implement the `DiPOA` algorithm according to the hybrid programming style to take advantage of both parallel computation styles. The parallel implementation is one of the most important features of the `RH-ADMM` algorithm, which is used to solve the D-NLP sub-problems of the `DiPOA` algorithm. It means that `RH-ADMM` renders the solution of the D-NLP problem fully decentralized for which modern CPU architectures can be utilized. In general, based on the communication topology of the `RH-ADMM` algorithm, it is convenient to think of `RH-ADMM` as a message-passing algorithm on a hypergraph, where each node corresponds to a subsystem and hyperedges correspond to shared variables. As mentioned, a well-known protocol for implementing this form of communication structure for parallel algorithms is the MPI. This section provides an overview of MPI and its important operations. `MPI` is a message-passing library that supports parallel and distributed computations. For being independent of any programming language, MPI is arguably the most widely used platform for high-performance distributed computing nowadays. MPI is a software library for which various interfaces exist from various



(a) MPI broadcast. (b) MPI scatter.

Figure 36 – MPI broadcast and scatter communication patterns.

programming languages, including `C/C++` and `Python`. MPI adopts the *Single Program, Multiple Data* (SPMD) programming paradigm to provide an efficient way to perform distributed computing. Using the SPMD paradigm, each node of the computing network runs the same program code, but it works with its own set of local variables and a separate subset of the data. To perform efficient distributed computing, MPI provides point-to-point and collective communications between the nodes of the computing network. Point-to-point communications refer to sending and receiving data between two different nodes, whereas collective communication is primarily performed among a set of nodes. In the following, we review some of the standard collective operations.

6.3.1 MPI Broadcast

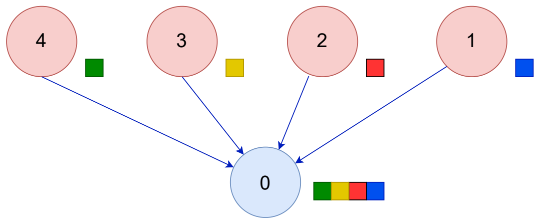
A *broadcast* is one of the basic collective communication strategies where one process sends the same data to all processes. Figure 36(a) illustrates the broadcast communication pattern.

6.3.2 MPI Scatter

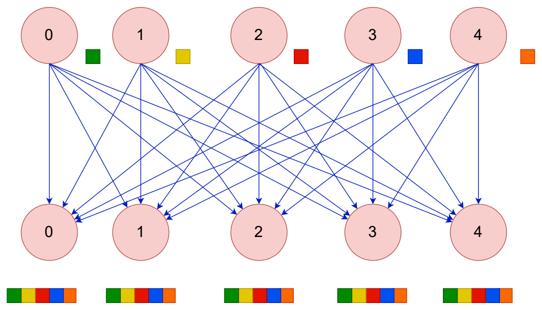
A *scatter* is a collective routine that involves a designated root process sending data to all other processes. The main difference between MPI scatter and broadcast is that while the MPI broadcast sends the *same* piece of data to all other processes, the MPI scatter sends *chunks of an array* to different processes, as shown in Figure 36(b).

6.3.2.1 MPI Gather and AllGather

In principle, the MPI *gather* is the inverse of the MPI scatter. Instead of distributing elements from one process to many processes, MPI gather takes elements from many processes and brings them together in one single process. The MPI *Allgather* collects all of the elements and then distributes them to all the processes. In the most basic scenario, MPI allgather is an MPI gather followed by an MPI broadcast. For example, Figure 37 illustrates the communication pattern in MPI gather and allgather.

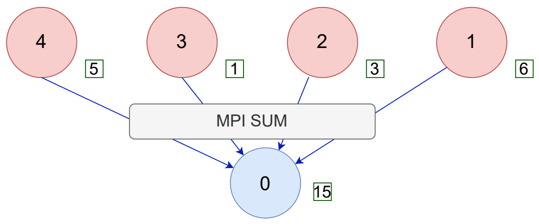


(a) MPI Gather.

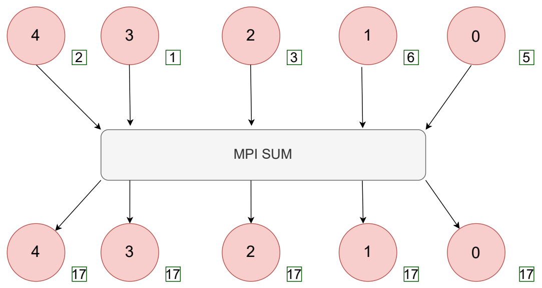


(b) MPI Allgather.

Figure 37 – MPI Gather and Allgather communication patterns



(a) MPI Reduce.



(b) MPI Allreduce.

Figure 38 – MPI Reduce and Allreduce Communication Patterns.

6.3.3 MPI Reduce and Allreduce

MPI *reduce* involves reducing a set of elements into a small set of elements via a function. The MPI reduce takes an array of input elements from each process and returns an array of output elements to the root process. In a complementary style of MPI allgather to MPI gather, MPI allreduce will reduce the values and distribute the results to all processes. MPI reduce and allreduce communication patterns are depicted in Figure 38.

6.4 TASK-BASED PROGRAMMING AND IMPLEMENTATION DETAILS

In this section, we discuss task-based paradigm algorithm implementation that SCOT uses to implement algorithm provided in Table 18. Task-based programming is a programming paradigm that focuses on creating software components that can be executed independently and concurrently. In the context of optimization solvers, this means decomposing the underlying optimization algorithm into a series of small tasks that can be executed in squentially or in parallel to solve the optimization problem. The SCOT framework is designed to support task-based programming for algorithm implementation. The framework is composed of several task classes, each of which is responsible for a specific task in the optimization process. These classes include tasks such as solving primal and dual problems, algorithm termination, timing and so on. Some of the important tasks implemented in SCOT is provided in Table 20.

Table 20 – Some of SCOT Task Classes and Descriptions

Task Class Name	**Task Description**
`TaskAddDualSolution`	Adds a dual solution
`TaskAddLinearOuterApproximation`	Adds a linear outer approximation to the dual problem
`TaskAddQuadraticOuterApproximation`	Adds a quadratic outer approximation to the dual problem
`TaskCheckDuration`	Checks if the algorithm has reached its maximum time limit
`TaskCheckHybridEvent`	Checks if a hybrid event has occurred
`TaskCheckSocEvent`	Checks if a second-order cone (SOC) event has occurred
`TaskCheckTerminationGap`	Checks if the optimality gap has fallen below a specified threshold
`TaskCreateMultipleTreeDualProblem`	Creates a multiple-tree dual problem
`TaskDistributedNlpSolution`	Solves the primal problem
`TaskInitializeMultipleTreeDualSolver`	Initializes the solver for the multiple-tree dual problem
`TaskInitializeNewIteration`	Initializes a new iteration of the solver
`TaskInitializeSingleTreeDualSolver`	Initializes the solver for the single-tree dual problem
`TaskSolveDualProblem`	Solves the dual problem
`TaskSolveSingleTreeDualProblem`	Solves the single-tree dual problem

Each task class inherits from the `TaskBase` base class, which includes four methods: `activate`, `deactivate`, `initialize`, and `execute`. These methods enable SCOT to activate, deactivate, initialize, and execute any task as needed. This approach results in a more modular and flexible algorithm implementation, as tasks can be manipulated at runtime. Thus, by utilizing various combinations of task classes, SCOT can implement a range of algorithms. In order to achieve this, SCOT defines a task queue for each algorithm, which specifies the sequence of tasks to be executed. The main algorithm loop functions as a task-scheduler, responsible for managing and executing the tasks, while ensuring that the dependencies between them are satisfied. This modular approach not only facilitates algorithm customization, but also enables efficient parallelization of the optimization process, potentially leading to significant speedups. Furthermore, the SCOT framework provides an easy way to customize the optimization process by allowing users to define their own tasks or modify existing ones. This flexibility allows users to tailor the optimization process to their specific needs, such as incorporating problem-specific knowledge (*e.g.* user-cuts) or implementing new optimization algorithms. Moreover, SCOT implements various classes to control the execution of the algorithms. These classes are accessible by the tasks classes and provide necessary information about the progress of SCOT algorithms. Table table:commons represents the common classes used by SCOT tasks and their description.

Table 21 – Some of SCOT Task Classes and Descriptions

Common Classes	**Description**
`Environment`	Solver Container
`Results`	Holds information of optimization progress
`Report`	Reports algorithm data
`Timer`	Controls the execution time of differet components
`PrimalProblem`	Holds data structures of primal problem.
`DualProblem`	Holds data structures of dual problem.
`Model`	Represents optimization problem (112)
`MessagePassingInterface`	Performs MPI operations

The `Envorinmet` class is a container for a collection of SCOT objects that share common settings. By using `Envorinmet` class, SCOT allows for more flexibility in terms of changing solver settings for different parts of a program or for different threads running in parallel. It also enables more efficient use of system resources by allowing multiple models to share the same environment. Each task in SCOT can utilize the shared classes and objects provided by the `Environment` class to perform the necessary computations. This allows for a more efficient and organized way of solving optimization problems, as each task can access and modify the necessary data structures without the need for redundant computations or data duplication. By sharing the environment among different tasks, SCOT can significantly reduce the memory footprint and computational time required for optimization.

6.5 PROBLEM REPRESENTATION AND SOLVER OPTIONS

SCOT has its own parser which enables it to read problems in JSON format. This file contains essential information about the problem class that SCOT needs to solve. It specifies the dataset names and their addresses where they are stored, as well as the objective function, constraints, and variables.

The solver options are essential to customize the solving process in SCOT. These options are provided in a text-based pair-value format, where each option is specified in the form of `option=value`. The options are organized into different categories based on their functionalities, such as `Algorithms`, `TerminationCriteria`, and `Logging`. For example, the option `Algorithms.SingleTree` belongs to the `Algorithms` category and determines whether the `SingleTree` algorithm should be used during the solving process.

Apart from Boolean options like `Algorithms.SingleTree`, there are other option types such as string, float, and integer-valued options. These options allow the user to specify various aspects of the solving process, such as the maximum number of iterations, the search strategy to be used, and the log file name. The solver options and their corresponding values are documented in a configuration file with a `*.cfg` extension. This file is used to provide default values for the solver options, which can be overridden by command-line arguments or environment variables. The configuration file makes it easy for users to modify the default settings and experiment with different options without changing the solver code.

6.6 FEASIBILITY PUMP AND INFEASIBILITY DETECTION

By default, `SCOT` does not utilize the SFP algorithm. However, it can be enabled by setting the `Algorithms.SFP.Level` option to a value of 1 or 2. The `Algorithms.SFP` option has three levels that can be configured. A value of 0 indicates that the SFP algorithm will not be utilized at all. A value of 1 means that a centralized SFP algorithm will be employed, where the objective function in the SFP relaxation problem (140) is ignored. On the other hand, a value of 2 indicates that a distributed SFP algorithm will be utilized to solve the mixed-integer linear programming problem. If an optimization problem involves nonlinear constraints, the SFP method described in Algorithm 5 may encounter difficulties in converging. To address this issue, `SCOT` offers the `Algorithms.SFP.MaxTimeLimit` option, which allows users to set a maximum time limit for the SFP algorithm to run. By setting this option, SCOT will continue running the SFP algorithm for the specified time which can be particularly useful in cases where the optimization problem is complex and may require more time to converge due to the presence of nonlinear constraints.

Moreover, `SCOT` utilizes infeasibility detection when solving an optimization problem. However, if the problem only involves the sparsity constraint, this feature can be disregarded. To disable infeasibility detection in such cases, the `Algorithms.Primal.InfeasibilityDetection` option can be set to a value of 0. By disabling infeasibility detection, SCOT can potentially achieve faster computation times in scenarios where the sparsity constraint is the only constraint present.

6.7 MIP AND DISTRIBUTED SUB-SOLVERS

`SCOT` extensively depends on subsolvers in its primary algorithms, with the majority of the workload being handled by MIP and distributed NLP solvers. To solve the MIP subproblems, SCOT relies on third-party solvers such as GUROBI and Cbc. On the other hand, the distributed NLP solver, which is based on the RH-ADMM algorithm, is implemented as a separate module within the framework. SCOT offers interfaces for both MIP and distributed NLP solvers, which are discussed in this section.

6.7.1 MIP Solver

Currently, `SCOT` is capable of interfacing with two solvers: the commercial solver GUROBI, and the open-source solver Cbc. The MIP solver can be selected by setting the `Dual.Solver` option. However, it is important to note that Cbc does not support quadratic terms in the constraints. As a result, linear outer approximations are the approximations that are used in the algorithms. Furthermore, the interface to Cbc does not support callbacks. Therefore, if Cbc is selected as the default MIP solver, the single tree search strategy is disabled by default. `SCOT` automatically passes on some relevant algorithmic options to the mixed-integer linear programming (MIP) solvers, including the number of threads, absolute and relative gap, and time limits. Additionally, users can specify a range of solver-specific parameters, which are documented in the SCOT manual.

6.7.2 Distributed NLP Solver

SCOT provides users with easy access to the RH-ADMM module, which is specifically designed to solve distributed nonlinear programming (NLP) problems. This module is fully integrated into the SCOT framework and can be accessed through the `Primal.Solver` category. Users can customize the RH-ADMM algorithm by specifying various options, such as the local NLP solver to be used. For example, if a user wants to use IPOPT as the local NLP solver, they can set the value of `Primal.Solver.LocalSolver` to "ipopt".

6.8 TERMINATION

By default, `SCOT` is terminated based on the relative and absolute objective gaps, which are calculated as follows:

$$\begin{aligned} G_{\mathtt{abs}} &= P_b - D_b \\ G_{\mathtt{rel}} &= \frac{P_b - D_b}{|P_b| + 10^{-8}} \end{aligned} \tag{146}$$

Here, P_b represents the best primal objective value found so far, while D_b represents the best dual bound provided by the dual problem. The termination criteria can be modified by changing the values of `Termination.RelativeGap` and `Termination.AbsoluteGap` options. These options control the relative and absolute optimality gap values, respectively. Additionally, `SCOT` offers options for termination based on maximum iteration and time limits, which can be set using `Termination.TimeLimit` and `Termination.IterLimit`. The behavior of the iteration limit differs depending on whether the single or multiple tree algorithms are used. In the case of multiple tree algorithms, the iteration limit is counted as the number of iterations that the MIP solver is called. On the other hand, in the single tree case, an iteration is considered as whenever the lazy callback is activated, indicating that a new integer feasible solution has been found. It is worth noting that for the same problem, the number of iterations is often much higher in the single tree case, but the solution time per iteration is often shorter since NLP subproblems are not solved at each node of the BnB tree.

6.9 RESULTS AND OUTPUT

After the solution process, the results and statistics are provided in a `JSON` format. For debugging purposes, `SCOT` offers an option called `Output.Debug.Enable`, which allows the user to specify a directory for storing intermediate files. These files can provide valuable insights into the solution process and help identify any issues that may arise during computation. Alternatively, the user can choose to store these files in a default temporary directory. To further enhance user control, `SCOT` allows customization of the amount of output displayed on the screen or written to the log file during the solution process. This feature enables the user to focus on relevant information, such as errors or warnings, and avoid cluttering the display with excessive data. By providing these options, `SCOT` ensures that users have the flexibility and transparency necessary to optimize their workflows and achieve optimal results.

6.10 BASIC SYNTAX AND USAGE

In this section, we present an illustrative example to show how `SCOT` Python API, `SCOTPY`, is used to solve a distributed sparse logistic regression problem with random data. To do so, we first import the required classes from `SCOTPY` as the following code snippet shows,

```
from scotpy import (AlgorithmType, ProblemType, ScotModel, ScotPy,
    ScotSettings
                    )
```

Listing 6.1 – Import statement of SCOT Python API

Here `ScotPy` is the main class that executes `SCOT` for a given problem and settings defined by `ScotModel` and `ScotSettings`, respectively. The `AlgorithmType` and `ProblemType` classes determine what problem class is solved and which algorithm will be used. In order to create the optimization problem and `SCOT` settings, the following code snippet can be used,

```
# Create a classification dataset with 1000 rows and 20 columns dataset,
res = make_classification(n_samples = 1000, n_features = 20)
scp = ScotModel(problem_name = "logistic_regression", rank = 0, kappa =
   5, ptype
= ProblemType.CLASSIFICATION)

# Set problem data with normalization
 scp.set_data(dataset, res, normalized_data = True)

# Create corresponding files that represent the optimization problem.
scp.create()

scot_settings = ScotSettings( relative_gap = 1e-5, time_limit = 100,
   verbose = True, algorithm = AlgorithmType.DIHOA)
```

Listing 6.2 – Problem definition and settings

where `make_classification` function, imported from Python `scikit-learn` library, is used to generate a random classification dataset. The `ScotModel` object is then created by a given problem name, MPI rank, number of nonzeros, and problem type. The solver settings can be defined by creating an object from `ScotSettings` class. We note that by executing MPI, the above code snippets are simultaneously executed by each node of the network. Hence, each node can use its own problem data. Finally, we solve the optimization problem by using the following code,

```
solver = ScotPy(problem, scot_settings)
status_code = solver.run()
```

Listing 6.3 – SCOT Execution

where `ScotPy` class is responsible for creating a solver object for a given problem and settings. By executing the `run` method of `ScotPy`, MPI execution with N nodes is started.

6.11 CONCOLUSION

In conclusion, this chapter has provided an overview of the SCOT solver and its technical details, including its architecture, basic syntax, and features. The modular design of SCOT allows for easy modification of existing algorithms and implementation of user-defined algorithms to solve SCO problems. Furthermore, the use of MPI as the main library allows for efficient distributed and parallel computation, which is essential in solving distributed optimization problems. Overall, the SCOT solver is a optimization tool that can be used across a wide range of applications.

7 FINAL REMARKS

This thesis proposed a distributed optimization framework and dedicated software tools designed to solve SCO problems across a network of computational agents. Within this setting, each agent is only aware of a portion of the optimization problem, while sharing the same decision vector with other agents. The following sections summarize the main accomplishments and findings from each chapter in the thesis and some future research directions.

7.1 CONCLUSIONS

Chapter 3 introduced DiGST as an initial algorithm. DiGST is a fully distributed approach with inexpensive computational iterations that splits the SCO problem into a distributed unconstrained convex optimization and a sparsity projection step. The sparsity projection step directly addresses the sparsity constraint and the distributed computation is performed by means of the gradient tracking algorithm. As a result, the DiGST only relies on local computation and communication and respects the data privacy in the network. However, despite these strengths, DiGST is not well-suited for large-scale applications due to difficult penalty parameter tuning and an inability to handle general linear and/or nonlinear constraints.

Motivated by the limitations of DiGST, we proposed a framework built upon MINLP and large-scale convex optimization. This framework consists of different distributed algorithms, heuristics, and software tools (SCOT), each of which was introduced in a dedicated chapter. Chapter 4 introduced the RH-ADMM algorithm as the first component of the framework. RH-ADMM is a distributed algorithm that can efficiently solve large-scale convex optimization problems with coupling constraints. Its development was crucial because the subsequent algorithms in the framework rely heavily on RH-ADMM to solve convex subproblems in a distributed manner.

Built upon RH-ADMM and the multiple-tree OA algorithm, Chapter 5 first introduced the DiPOA algorithm which is capable of solving solves SCO problems distributedly. DiPOA was the first algorithm that handle large-scale SCO problems with both linear and nonlinear constraints by taking advantage of the multi-core architecture of modern processors. To improve DiPOA performance, we introduced the DiHOA algorithm that is built upon DiPOA and LP/NLP BnB algorithms. The DiHOA algorithm constructs an initial BnB tree with a sufficient number of second-order outer approximations introduced in the root of the tree and lazily introduces outer approximations by interrupting the BnB tree when a feasible-integer dual solution is found. The lazy outer approximations are distributedly generated by using the RH-ADMM algorithm through callbacks. The single-tree strategy leads to a significant improvement in the DiPOA algorithm as the numerical benchmarks showed. In addition to the algorithms, the SFP algorithm was introduced to warm-start the DiPOA and DiHOA algorithms, and an event-triggered cut-generation scheme and practical infeasibility detection were proposed to improve solution quality and reduce the number of iterations. Based on the numerical benchmarks provided in Chapter 5, we found that the distributed algorithms proposed in this thesis, namely DiPOA and DiHOA, outperform existing centralized MINLP solvers in terms of solution quality and scalability.

In Chapter 6, we presented SCOT, a novel distributed software framework that implements the distributed optimization framework proposed in this thesis. To the author's knowledge, SCOT is the first software tool capable of handling SCO problems in distributed computing environments. SCOT utilizes MPI technology to facilitate parallel and distributed computations and can be used in high-performance computing platforms. SCOT implementation is highly adaptable and modular, allowing users to incorporate their own custom-distributed optimization algorithms and solvers.

7.2 FUTURE WORKS

The results presented in this thesis demonstrate the potential of SCOT and its algorithms to solve SCO problems that were previously considered unsolvable due to their size or distributed nature. Future research in sparse optimization has significant potential to be continued, particularly in SCO applications where distributed solutions are essential. In this section, we provide an overview of several crucial research directions in SCO.

- Development of more efficient algorithms: While SCOT algorithms enabled solving previously intractable problems, they can still be computationally intensive. Future research can focus on the development of more efficient distributed optimization algorithms.
- Asynchronous parallel updates: SCOT algorithms use synchronous updates to perform some certain iterations such as lazy callback updates. Future research can explore how the SCOT algorithms can be used by means of asynchronous updates.
- Theoretical analysis of the DiSGT algorithm to derive a closed-form formula for calculating the penalty parameter ρ that ensures convergence.
- Extend large-scale mixed-integer linear optimization methods, like branch and price algorithms, to the distributed setting to enhance the scalability of DiPOA and DiHOA algorithms.
- The design and development of problem-specific lazy cutting planes to tighten the problem formulation and reduce the number of nodes in the DiHOA algorithm.

- Investigation of MINLP primal heuristics and their applications in SCO problems.
- The investigation of alternative sparsity constraint modeling techniques such as perspective formulation. This technique involves the reformulation of sparsity constraint by means of perspective functions that incorporate. The resulting optimization problem can then be formulated as a convex MINLP problem whose distributed solution can be studied.

In conclusion, this thesis highlights the potential of distributed SCO problems, and its implications for future research are vast and diverse. The ongoing exploration of this area has the potential to address fundamental challenges in various fields, from statistics and machine learning to control engineering and power systems, and beyond.

REFERENCES

AGUILERA, Ricardo P; URRUTIA, Gabriel; DELGADO, Ramón A; DOLZ, Daniel; AGÜERO, Juan C. Quadratic model predictive control including input cardinality constraints. **IEEE Transactions on Automatic Control**, IEEE, v. 62, n. 6, p. 3068–3075, 2017.

ARZAMASOV, Vadim; BÖHM, Klemens; JOCHEM, Patrick. Towards concise models of grid stability. In: IEEE International Conference on Communications, Control, and Computing Technologies for Smart Grids (SmartGridComm). [S.l.: s.n.], 2018. P. 1–6.

AYTUG, Haldun. Feature selection for support vector machines using Generalized Benders Decomposition. **European Journal of Operational Research**, Elsevier, v. 244, n. 1, p. 210–218, 2015.

BAI, Jianchao; ZHANG, Hongchao; LI, Jicheng. A parameterized proximal point algorithm for separable convex optimization. **Optimization Letters**, v. 12, n. 7, p. 1589–1608, 2018.

BAI, Yanqin; LIANG, Renli; YANG, Zhouwang. Splitting augmented Lagrangian method for optimization problems with a cardinality constraint and semicontinuous variables. **Optimization Methods and Software**, Taylor & Francis, v. 31, n. 5, p. 1089–1109, 2016.

BAI, Yanqin; LIANG, Renli; YANG, Zhouwang. Splitting augmented Lagrangian method for optimization problems with a cardinality constraint and semicontinuous variables. **Optimization Methods and Software**, v. 31, n. 5, p. 1089–1109, 2016.

BAUSCHKE, Heinz H.; COMBETTES, Patrick L. **Convex Analysis and Monotone Operator Theory in Hilbert Spaces**. [S.l.]: Springer, 2017.

BERTHOLD, Timo; LODI, Andrea; SALVAGNIN, Domenico. Ten years of feasibility pump, and counting. **EURO Journal on Computational Optimization**, Elsevier, v. 7, n. 1, p. 1–14, 2019.

BERTSEKAS, Dimitri P. **Convex Optimization Algorithms**. [S.l.]: Athena Scientific, 2015.

BERTSIMAS, Dimitris; CORY-WRIGHT, Ryan. A scalable algorithm for sparse portfolio selection. **INFORMS Journal on Computing**, INFORMS, 2022.

BERTSIMAS, Dimitris; DUNN, Jack; KAPELEVICH, Lea; ZHANG, Rebecca. Sparse regression over clusters: SparClur. **Optimization Letters**, Springer, p. 1–16, 2022.

BERTSIMAS, Dimitris; KING, Angela. Logistic Regression: From Art to Science. **Statistical Science**, v. 32, n. 3, p. 367–384, 2017.

BERTSIMAS, Dimitris; KING, Angela. Logistic regression: From art to science. **Statistical Science**, JSTOR, p. 367–384, 2017.

BERTSIMAS, Dimitris; KING, Angela; MAZUMDER, Rahul. Best subset selection via a modern optimization lens. **The annals of statistics**, Institute of Mathematical Statistics, v. 44, n. 2, p. 813–852, 2016.

BERTSIMAS, Dimitris; MUNDRU, Nishanth. Sparse convex regression. **INFORMS Journal on Computing**, INFORMS, v. 33, n. 1, p. 262–279, 2021.

BERTSIMAS, Dimitris; MUNDRU, Nishanth. Sparse convex regression. **INFORMS Journal on Computing**, INFORMS, v. 33, n. 1, p. 262–279, 2021.

BERTSIMAS, Dimitris; PAUPHILET, Jean; VAN PARYS, Bart. Sparse classification: a scalable discrete optimization perspective. **Machine Learning**, Springer, v. 110, n. 11, p. 3177–3209, 2021.

BERTSIMAS, Dimitris; SHIODA, Romy. Algorithm for cardinality-constrained quadratic optimization. **Computational Optimization and Applications**, v. 43, n. 1, p. 1–22, 2009.

BERTSIMAS, Dimitris; WEISMANTEL, Robert. **Optimization over integers**. [S.l.]: Dynamic Ideas Belmont, MA, 2005.

BIENSTOCK, Daniel. Computational study of a family of mixed-integer quadratic programming problems. **Mathematical Programming**, Springer, v. 74, n. 2, p. 121–140, 1996.

BIENSTOCK, Daniel. Computational study of a family of mixed-integer quadratic programming problems. **Mathematical Programming, Series B**, v. 74, 2 PART A, p. 121–140, 1996. ISSN 00255610.

BOLOGNANI, Saverio; CARLI, Ruggero; TODESCATO, Marco. State estimation in power distribution networks with poorly synchronized measurements. In: IEEE 53rd Annual Conference on Decision and Control (CDC). [S.l.: s.n.], 2014. P. 2579–2584.

BOYD, Stephen; BOYD, Stephen P; VANDENBERGHE, Lieven. **Convex optimization**. [S.l.]: Cambridge university press, 2004.

BOYD, Stephen; PARIKH, Neal; CHU, Eric; PELEATO, Borja; ECKSTEIN, Jonathan, et al. Distributed optimization and statistical learning via the alternating direction method of multipliers. **Foundations and Trends® in Machine Learning**, NOW Publishers, Inc., v. 3, n. 1, p. 1–122, 2011.

CAMPONOGARA, Eduardo; SCHERER, Helton F. Distributed optimization for model predictive control of linear dynamic networks with control-input and output constraints. **IEEE Transactions on Automation Science and Engineering**, v. 8, n. 1, p. 233–242, 2011.

CARNEVALE, Guido; FARINA, Francesco; NOTARNICOLA, Ivano; NOTARSTEFANO, Giuseppe. GTAdam: Gradient tracking with adaptive momentum for distributed online optimization. **IEEE Transactions on Control of Network Systems**, IEEE, 2022.

CARNEVALE, Guido; NOTARNICOLA, Ivano; MARCONI, Lorenzo; NOTARSTEFANO, Giuseppe. Triggered gradient tracking for asynchronous distributed optimization. **Automatica**, Elsevier, v. 147, p. 110726, 2023.

CARNEVALE, Guido; NOTARSTEFANO, Giuseppe. Nonconvex Distributed Optimization via Lasalle and Singular Perturbations. **IEEE Control Systems Letters**, IEEE, v. 7, p. 301–306, 2022.

CARRON, Andrea; TODESCATO, Marco; CARLI, Ruggero; SCHENATO, Luca. An asynchronous consensus-based algorithm for estimation from noisy relative measurements. **IEEE Transactions on Control of Network Systems**, v. 1, n. 3, p. 283–295, 2014.

CHANG, Tsung-Hui. A Proximal Dual Consensus ADMM Method for Multi-Agent Constrained Optimization. **IEEE Transactions on Signal Processing**, v. 64, n. 14, p. 3719–3734, July 2016.

CHANG, Xiaokai; LIU, Sanyang; ZHAO, Pengjun; LI, Xu. Convergent prediction–correction-based ADMM for multi-block separable convex programming. **Journal of Computational and Applied Mathematics**, v. 335, p. 270–288, 2018.

CHEN, Gang; YANG, Qing. Distributed constrained optimization for multi-agent networks with nonsmooth objective functions. en. **Systems & Control Letters**, v. 124, p. 60–67, Feb. 2019.

DALL'ANESE, Emiliano; SIMONETTO, Andrea. Optimal power flow pursuit. **IEEE Transactions on Smart Grid**, v. 9, n. 2, p. 942–952, 2018.

DANESHMAND, Amir; SCUTARI, Gesualdo; KUNGURTSEV, Vyacheslav. Second-order guarantees of distributed gradient algorithms. **SIAM Journal on Optimization**, SIAM, v. 30, n. 4, p. 3029–3068, 2020.

DAVIS, Damek; YIN, Wotao. Faster convergence rates of relaxed Peaceman-Rachford and ADMM under regularity assumptions. **Mathematics of Operations Research**, v. 42, n. 3, p. 783–805, 2017.

DI LORENZO, Paolo; SCUTARI, Gesualdo. NEXT: In-network nonconvex optimization. **IEEE Transactions on Signal and Information Processing over Networks**, IEEE, v. 2, n. 2, p. 120–136, 2016.

DUA, Dheeru; GRAFF, Casey. **UCI Machine Learning Repository**. [S.l.: s.n.], 2017. Available from: `http://archive.ics.uci.edu/ml`.

DURAN, Marco A; GROSSMANN, Ignacio E. An outer-approximation algorithm for a class of mixed-integer nonlinear programs. **Mathematical Programming**, Springer, v. 36, n. 3, p. 307–339, 1986.

ECKSTEIN, Jonathan; YAO, Wang. Understanding the convergence of the alternating direction method of multipliers: Theoretical and computational perspectives. **Pacific Journal of Optimization**, v. 11, n. 4, p. 619–644, 2015.

ELDAR, Yonina C; KUTYNIOK, Gitta. **Compressed sensing: theory and applications**. [S.l.]: Cambridge university press, 2012.

FARINA, Francesco; GARULLI, Andrea; GIANNITRAPANI, Antonio; NOTARSTEFANO, Giuseppe. A distributed asynchronous method of multipliers for constrained nonconvex optimization. **Automatica**, v. 103, p. 243–253, May 2019.

FLETCHER, Roger; LEY, Sven. Solving Mixed Integer Nonlinear Programs by Outer Approximation 1 Introduction. **October**, v. 66, p. 327–349, 1996.

FOUCART, Simon; RAUHUT, Holger. An Invitation to Compressive Sensing. In: A Mathematical Introduction to Compressive Sensing. [S.l.]: Springer, 2013. P. 1–39.

FRANGIONI, Antonio; GENTILE, Claudio. Perspective cuts for a class of convex 0–1 mixed integer programs. **Mathematical Programming**, Springer, v. 106, p. 225–236, 2006.

FRANGIONI, Antonio; GENTILE, Claudio; LACALANDRA, Fabrizio. Tighter approximated MILP formulations for unit commitment problems. **IEEE Transactions on Power Systems**, IEEE, v. 24, n. 1, p. 105–113, 2008.

GABAY, Daniel; MERCIER, Bertrand. **A dual algorithm for the solution of non linear variational problems via finite element approximation**. [S.l.]: Institut de recherche d'informatique et d'automatique, 1975.

GHADIMI, Euhanna; TEIXEIRA, André; SHAMES, Iman; JOHANSSON, Mikael. Optimal parameter selection for the alternating direction method of multipliers (ADMM): quadratic problems. **IEEE Transactions on Automatic Control**, v. 60, n. 3, p. 644–658, 2015.

GISELSSON, Pontus; BOYD, Stephen. Diagonal scaling in Douglas-Rachford splitting and ADMM. In: IEEE 53rd Annual Conference on Decision and Control (CDC). [S.l.: s.n.], 2014. P. 5033–5039.

GISELSSON, Pontus; BOYD, Stephen. Linear convergence and metric selection for Douglas-Rachford splitting and ADMM. **IEEE Transactions on Automatic Control**, v. 62, n. 2, p. 532–544, 2017.

GLOWINSKI, Roland; MARROCO, A. Sur l'approximation, par éléments finis d'ordre un, et la résolution, par pénalisation-dualité d'une classe de problèmes de Dirichlet non linéaires. **Revue française d'automatique, informatique, recherche opérationnelle. Analyse numérique**, EDP Sciences, v. 9, R2, p. 41–76, 1975.

GROPP, William; LUSK, Ewing; SKJELLUM, Anthony. **Using MPI: Portable Parallel Programming with the Message-Passing Interface**. [S.l.]: MIT Press, 1999. v. 1.

GROSSMANN, Ignacio. Review of Nonlinear Mixed-Integer and Disjunctive Programming Techniques. **Optimization and Engineering**, v. 3, n. 3, p. 227–252, 2002. ISSN 1389-4420.

GUROBI OPTIMIZATION, LLC. **Gurobi Optimizer Reference Manual**. [S.l.: s.n.], 2020. Available from: `http://www.gurobi.com`.

HE, Bingsheng; HOU, Liusheng; YUAN, Xiaoming. On full Jacobian decomposition of the augmented Lagrangian method for separable convex programming. **SIAM Journal on Optimization**, v. 25, n. 4, p. 2274–2312, 2015.

HIJAZI, Hassan; BONAMI, Pierre; OUOROU, Adam. An outer-inner approximation for separable mixed-integer nonlinear programs. **INFORMS Journal on Computing**, INFORMS, v. 26, n. 1, p. 31–44, 2014.

HONG, Mingyi. A distributed, asynchronous and incremental algorithm for nonconvex optimization: an ADMM approach. **IEEE Transactions on Control of Network Systems**, v. 5, n. 3, p. 935–945, 2017.

IUTZELER, Franck; BIANCHI, Pascal; CIBLAT, Philippe; HACHEM, Walid. Explicit convergence rate of a distributed alternating direction method of multipliers. **IEEE Transactions on Automatic Control**, v. 61, n. 4, p. 892–904, 2016.

KAMIYA, Shunsuke; MIYASHIRO, Ryuhei; TAKANO, Yuichi. Feature subset selection for the multinomial logit model via mixed-integer optimization. In: PMLR. THE 22nd International Conference on Artificial Intelligence and Statistics. [S.l.: s.n.], 2019. P. 1254–1263.

KIM, Yuwon; KIM, Jinseog; KIM, Yongdai. Blockwise sparse regression. **Statistica Sinica**, JSTOR, p. 375–390, 2006.

KOCH, Thorsten et al. MIPLIB 2010: mixed integer programming library version 5. **Mathematical Programming Computation**, Springer, v. 3, p. 103–163, 2011.

KRONQVIST, Jan; BERNAL, David E.; GROSSMANN, Ignacio E. Using regularization and second order information in outer approximation for convex MINLP. **Mathematical Programming**, v. 180, n. 1-2, p. 285–310, 2020.

KRONQVIST, Jan; BERNAL, David E.; LUNDELL, Andreas; GROSSMANN, Ignacio E. review and comparison of solvers for convex MINLP. **Optimization and Engineering**, v. 20, n. 2, p. 397–455, 2019.

KRONQVIST, Jan; LUNDELL, Andreas; WESTERLUND, Tapio. Reformulations for utilizing separability when solving convex MINLP problems. **Journal of Global Optimization**, Springer, v. 71, n. 3, p. 571–592, 2018.

LEGAT, Benoit; DOWSON, Oscar; GARCIA, Joaquim Dias; LUBIN, Miles. MathOptInterface: a data structure for mathematical optimization problems. **INFORMS Journal on Computing**, INFORMS, v. 34, n. 2, p. 672–689, 2021.

LEWIS, Frank L. Wireless sensor networks. **Smart environments: technologies, protocols, and applications**, Wiley Online Library, p. 11–46, 2004.

LIU, Ji; HUANG, Jizhou; ZHOU, Yang; LI, Xuhong; JI, Shilei; XIONG, Haoyi; DOU, Dejing. From distributed machine learning to federated learning: A survey. **Knowledge and Information Systems**, Springer, v. 64, n. 4, p. 885–917, 2022.

LU, Zhaosong; ZHANG, Yong. Sparse approximation via penalty decomposition methods. **SIAM Journal on Optimization**, SIAM, v. 23, n. 4, p. 2448–2478, 2013.

LUNDELL, Andreas; KRONQVIST, Jan. Integration of polyhedral outer approximation algorithms with MIP solvers through callbacks and lazy constraints. **AIP Conference Proceedings**, v. 2070, March, 2019. ISSN 15517616.

MA, Meng; NIKOLAKOPOULOS, Athanasios N.; GIANNAKIS, Georgios B. Fast Decentralized Optimization over Networks. **arXiv preprint arXiv:1804.02425**, 2018.

MA, Shuangge; SONG, Xiao; HUANG, Jian. Supervised group Lasso with applications to microarray data analysis. **BMC bioinformatics**, Springer, v. 8, n. 1, p. 1–17, 2007.

MOTA, Joao F. C.; XAVIER, Joao M. F.; AGUIAR, Pedro M. Q.; PUSCHEL, Markus. D-ADMM: A communication-efficient distributed algorithm for separable optimization. **IEEE Transactions on Signal Processing**, v. 61, n. 10, p. 2718–2723, 2013.

NATARAJAN, Balas Kausik. Sparse approximate solutions to linear systems. **SIAM journal on computing**, SIAM, v. 24, n. 2, p. 227–234, 1995.

NECOARA, I; SUYKENS, JAK. Interior-point lagrangian decomposition method for separable convex optimization. **Journal of Optimization Theory and Applications**, Springer, v. 143, n. 3, p. 567, 2009.

NEDIĆ, Angelia; LIU, Ji. Distributed Optimization for Control. **Annual Review of Control, Robotics, and Autonomous Systems**, v. 1, p. 77–103, 2018.

NEDIĆ, Angelia; OLSHEVSKY, Alex. Distributed optimization over time-varying directed graphs. **IEEE Transactions on Automatic Control**, v. 60, n. 3, p. 601–615, 2015.

NEDIĆ, Angelia; OLSHEVSKY, Alex; SHI, Wei. Achieving geometric convergence for distributed optimization over time-varying graphs. **SIAM Journal on Optimization**, SIAM, v. 27, n. 4, p. 2597–2633, 2017.

NOTARNICOLA, Ivano; SUN, Ying; SCUTARI, Gesualdo; NOTARSTEFANO, Giuseppe. Distributed big-data optimization via block-iterative convexification and averaging. In: IEEE 56th Annual Conference on Decision and Control (CDC). [S.l.: s.n.], Dec. 2017. P. 2281–2288.

NOTARSTEFANO, Giuseppe; NOTARNICOLA, Ivano; CAMISA, Andrea, et al. Distributed Optimization for Smart Cyber-Physical Networks. **Foundations and Trends® in Systems and Control**, Now Publishers, Inc., v. 7, n. 3, p. 253–383, 2019.

OLAMA, Alireza; BASTIANELLO, Nicola; MENDES, Paulo RC; CAMPONOGARA, Eduardo. Relaxed hybrid consensus ADMM for distributed convex optimisation with coupling constraints. **IET Control Theory & Applications**, IET, v. 13, n. 17, p. 2828–2837, 2019.

OLAMA, Alireza; CAMPONOGARA, Eduardo; KRONQVIST, Jan. Sparse Convex Optimization Toolkit: A Mixed-Integer Framework. **arXiv preprint arXiv:2210.16896**, 2022.

OLAMA, Alireza; CAMPONOGARA, Eduardo; MENDES, Paulo. Distributed Primal Outer Approximation Algorithm for Sparse Convex Programming with Separable Structures. **arXiv preprint arXiv:2210.06913**, 2021.

OLAMA, Alireza; CARNEVALE, Guido; NOTARSTEFANO, Giuseppe; CAMPONOGARA, Eduardo. A Tracking Augmented Lagrangian Method for ℓ_0 Sparse Consensus Optimization. **9th International Conference on Control, Decision and Information Technologies (CoDIT), Roma, Italy (Accepted)**, 2023.

PENG, Zhimin; XU, Yangyang; YAN, Ming; YIN, Wotao. ARock: an algorithmic framework for asynchronous parallel coordinate updates. **SIAM Journal on Scientific Computing**, v. 38, n. 5, a2851–a2879, 2016.

QUESADA, Ignacio; GROSSMANN, Ignacio E. An LP/NLP based branch and bound algorithm for convex MINLP optimization problems. **Computers & Chemical Engineering**, Elsevier, v. 16, n. 10-11, p. 937–947, 1992.

RYU, Ernest K; BOYD, Stephen. Primer on monotone operator methods. **Appl. Comput. Math**, v. 15, n. 1, p. 3–43, 2016.

SANT'ANNA, Leonardo Riegel; OLIVEIRA, Alan Delgado de; FILOMENA, Tiago Pascoal; CALDEIRA, João Frois. Solving the index tracking problem based on a convex reformulation for cointegration. **Finance Research Letters**, Elsevier, v. 37, p. 101356, 2020.

SCHERER, Helton; CAMPONOGARA, Eduardo; NORMEY-RICO, Júlio; ÁLVAREZ, José Domingo; GUZMÁN, José Luis. Distributed MPC for resource-constrained control systems. **Optimal Control Applications and Methods**, v. 36, n. 3, p. 272–291, 2015.

SCUTARI, Gesualdo; SUN, Ying. Distributed nonconvex constrained optimization over time-varying digraphs. **Mathematical Programming**, Springer, v. 176, n. 1-2, p. 497–544, 2019.

SHI, Wei; LING, Qing; WU, Gang; YIN, Wotao. EXTRA: An exact first-order algorithm for decentralized consensus optimization. **SIAM Journal on Optimization**, SIAM, v. 25, n. 2, p. 944–966, 2015.

SLAVAKIS, Konstantinos; GIANNAKIS, Georgios B; MATEOS, Gonzalo. Modeling and optimization for big data analytics: (statistical) learning tools for our era of data deluge. **IEEE Signal Processing Magazine**, v. 31, n. 5, p. 18–31, 2014.

SONG, Changkyu; YOON, Sejong; PAVLOVIC, Vladimir. Fast ADMM Algorithm for Distributed Optimization with Adaptive Penalty. In: THE Thirtieth AAAI Conference on Artificial Intelligence. [S.l.: s.n.], 2016. P. 753–759.

SU, Lijie; TANG, Lixin; BERNAL, David E; GROSSMANN, Ignacio E. Improved quadratic cuts for convex mixed-integer nonlinear programs. **Computers & Chemical Engineering**, v. 109, p. 77–95, 2018.

SUN, Xiaoling; ZHENG, Xiaojin; LI, Duan. Recent Advances in Mathematical Programming with Semi-continuous Variables and Cardinality Constraint. **Journal of the Operations Research Society of China**, v. 1, n. 1, p. 55–77, 2013.

TAWARMALANI, Mohit; SAHINIDIS, Nikolaos V. A polyhedral branch-and-cut approach to global optimization. **Mathematical programming**, Springer, v. 103, n. 2, p. 225–249, 2005.

TIAN, Yingjie; ZHANG, Yuqi. A comprehensive survey on regularization strategies in machine learning. **Information Fusion**, Elsevier, v. 80, p. 146–166, 2022.

TIBSHIRANI, Robert. Regression shrinkage and selection via the lasso. **Journal of the Royal Statistical Society: Series B (Methodological)**, Wiley Online Library, v. 58, n. 1, p. 267–288, 1996.

TILLMANN, Andreas M; BIENSTOCK, Daniel; LODI, Andrea; SCHWARTZ, Alexandra. Cardinality minimization, constraints, and regularization: A survey. **arXiv preprint arXiv:2106.09606**, 2021.

WANG, Fenlan; CAO, Liyuan. A new algorithm for quadratic integer programming problems with cardinality constraint. **Japan Journal of Industrial and Applied Mathematics**, Springer, v. 37, n. 2, p. 449–460, 2020.

WANG, Rui; LI, Qiqiang; ZHANG, Bingying; WANG, Luhao. Distributed Consensus Based Algorithm for Economic Dispatch in a Microgrid. **IEEE Transactions on Smart Grid**, v. 10, p. 3630–3640, 4 2018.

WANG, Zheming; ONG, Chong Jin. Distributed model predictive control of linear discrete-time systems with local and global constraints. **Automatica**, v. 81, p. 184–195, 2017.

WANG, Zheming; ONG, Chong-Jin. Accelerated distributed MPC of linear discrete-time systems with coupled constraints. **IEEE Transactions on Automatic Control**, v. 63, p. 3838–3849, 11 2018.

WEI, Ermin; OZDAGLAR, Asuman. On the $O(1/k)$ convergence of asynchronous distributed alternating direction method of multipliers. In: IEEE Global Conference on Signal and Information Processing (GlobalSIP). [S.l.: s.n.], 2013. P. 551–554.

XI, Chenguang; XIN, Ran; KHAN, Usman A. ADD-OPT: Accelerated distributed directed optimization. **IEEE Transactions on Automatic Control**, IEEE, v. 63, n. 5, p. 1329–1339, 2017.

XIA, Haibo; LI, Qiang; XU, Ruilin; CHEN, Tao; WANG, Jianguo; HASSAN, Muhammad Arshad Shehzad; CHEN, Minyou. Distributed Control Method for Economic Dispatch in Islanded Microgrids With Renewable Energy Sources. **IEEE Access**, IEEE, v. 6, p. 21802–21811, 2018.

XIE, Pei; YOU, Keyou; TEMPO, Roberto; SONG, Shiji; WU, Cheng. Distributed Convex Optimization with Inequality Constraints over Time-Varying Unbalanced Digraphs. **IEEE Transactions on Automatic Control**, v. 63, n. 12, p. 4331–4337, Dec. 2018.

XU, Zheng; FIGUEIREDO, Mário A. T.; YUAN, Xiaoming; STUDER, Christoph; GOLDSTEIN, Tom. Adaptive relaxed ADMM: Convergence theory and practical implementation. In: IEEE Conference on Computer Vision and Pattern Recognition (CVPR). [S.l.: s.n.], 2017. P. 7234–7243.

XU, Zheng; TAYLOR, Gavin; LI, Hao; FIGUEIREDO, Mario; YUAN, Xiaoming; GOLDSTEIN, Tom. Adaptive consensus ADMM for distributed optimization. **arXiv preprint arXiv:1706.02869**, 2017.

YILDIRIM, Kasım Sinan; CARLI, Ruggero; SCHENATO, Luca; TODESCATO, Marco. A distributed dual-ascent approach for power control of wireless power transfer networks. In: IEEE 56th Annual Conference on Decision and Control (CDC). [S.l.: s.n.], 2017. P. 3507–3512.

ZHANG, Ruiliang; KWOK, James. Asynchronous distributed ADMM for consensus optimization. In: INTERNATIONAL Conference on Machine Learning. [S.l.: s.n.], 2014. P. 1701–1709.