Sparse convex optimization toolkit: a mixed-integer framework

Alireza Olama, Eduardo Camponogara and Jan Kronqvist

ABSTRACT
This paper proposes an open-source distributed solver for solving Sparse Convex Optimization (SCO) problems over computational networks. Motivated by past algorithmic advances in mixed-integer optimization, the Sparse Convex Optimization Toolkit (SCOT) adopts a mixed-integer approach to find exact solutions to SCO problems. In particular, SCOT combines various techniques to transform the original SCO problem into an equivalent convex Mixed-Integer Nonlinear Programming (MINLP) problem that can benefit from high-performance and parallel computing platforms. To solve the equivalent mixed-integer problem, we present the Distributed Hybrid Outer Approximation (DiHOA) algorithm that builds upon the LP/NLP-based branch-and-bound and is tailored for this specific problem structure. The DiHOA algorithm combines the so-called single- and multi-tree outer approximation, naturally integrates a decentralized algorithm for distributed convex nonlinear subproblems, and employs enhancement techniques such as quadratic cuts. Finally, we present detailed computational experiments that show the benefit of our solver through numerical benchmarks on 140 SCO problems with distributed datasets. To show the overall efficiency of SCOT we also provide solution profiles comparing SCOT to other state-of-the-art MINLP solvers.

1. Introduction
In recent years, Sparse Convex Optimization (SCO) has gained considerable attention in several disciplines, from machine learning and engineering to economics and finance [5,7,46]. Several mathematical optimization problems in this context can be formulated as a general convex optimization problem subject to a constraint that allows only up to a certain number of decision variables to be nonzero. We refer to this constraint as a sparsity constraint. Hence, any convex optimization problem with the sparsity constraint can be regarded as a SCO problem [3,7,10,43]. Due to the non-convexity and discontinuity of the sparsity constraint, the SCO problems are known to be NP-Hard [7,34]. To overcome the computational difficulties imposed by the sparsity constraint, computationally tractable
convex optimization-based methods have been proposed. One of the popular methods is a $\ell_1$ norm relaxation method where a $\ell_1$ regularizer is imposed on the decision vector. The $\ell_1$ method naturally produces a sparse solution by setting many variables to zero. One of the popular $\ell_1$ based methods is Lasso which is widely used in statistics and machine learning communities [12,15,45].

One of the important reasons behind the popularity of $\ell_1$ based methods is their computational efficiency and scalability to practical-sized problems. However, in spite of their favourable computational properties, these methods can have some shortcomings. For example, they cannot guarantee that $\ell_1$ 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 $\ell_1$ 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 [25], logistic regression [5], compressed sensing [18], and microarray analysis [33].

Another approach to solving the SCO problems is to view the SCO problems as equivalent Mixed-Integer Programming (MIP) problems. 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 [23]. The resulting MIP formulation is flexible and can be adjusted based on the application's needs. Moreover, by defining suitable binary variables, the MIP framework can provide exact sparse solutions to SCO problems. The MIP framework is gaining popularity in various areas such as statistical data analysis and interpretable machine learning [4,6–8], sparse control [1], unit commitment, face recognition, sensor network design [30], portfolio optimization [3], and compressed sensing [20].

The mentioned works focussed on centralized solutions to SCO problems that might not be suitable for modern real-world applications when the data is inherently distributed or available in large volumes. Considering the limitations of centralized architectures, in the past few years, mainly because of the rise of Big data, distributed optimization over networks has gained growing attention [37,38]. The primary purpose of distributed optimization is to solve an optimization problem over a network of computing nodes. Each node performs local computations with access only to a portion of the problem data. Nodes are also capable of exchanging information with other nodes in the network. See [38] for a comprehensive overview of the most common distributed optimization algorithms. This paper introduces SCOT, a distributed optimization solver designed to solve SCO problems over peer-to-peer networks of computing nodes. In essence, SCOT consists of two distributed algorithms developed by the authors to solve SCO problems where an iterative procedure is applied by each network node, alternating communication and computation phases until a solution is found. To the best of our knowledge, SCOT is the first software framework that can solve SCO problems using distributed algorithms, enabling practical applications with a large number of sample data points, while keeping the data private to each node and allowing implementation in a computer cluster.

Formally, we consider the SCO problem as a mathematical programming problem that consists of finding the $\kappa$-sparse optimal solution of a distributed convex optimization
problem of the following form,

$$\min_{x \in \mathbb{R}^n} \sum_{i=1}^{N} f_i(x)$$

subject to $Ax \leq b$

$$\|x\|_0 \leq \kappa$$

(SCO)

where $N$ is the number of nodes of the computation network, $x \in \mathbb{R}^n$ 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 $i \in \{1, \ldots, N\}$. We use the $\ell_0$ norm (i.e. $\|x\|_0 = |\text{supp}(x)| = |\{j : x_j \neq 0\}|$) to define the sparsity constraint, which imposes the number of non-zero elements of $x$ to be less than a given integer $\kappa$. Finally, the matrix $A \in \mathbb{R}^{m \times n}$ and the vector $b \in \mathbb{R}^m$ define the set of given linear constraints assumed to be known by all nodes.

The optimization problem (SCO) finds widespread use in several real-world applications including statistics and machine learning. Two important problems that fall under this category are Distributed Sparse Linear Regression (DSLinR) and Distributed Sparse Logistic Regression (DSLogR) problems. The DLinR 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 $\lambda > 0$, 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 $\theta$. The DLinR problem is defined as,

$$\min_{\theta} \sum_{i=1}^{N} \|X_i \theta - b_i\|^2_2 + \frac{\lambda}{2} \|\theta\|^2_2$$

subject to $\|\theta\|_0 \leq \kappa$.

(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 so-called *logistic loss function* subject to a sparsity constraint. Suppose each computational node has $p_i$ points that represent training examples in a feature space and suppose they are associated with binary labels. The DSLogR problem is defined as,

$$\min_{\theta} \sum_{i=1}^{N} \left[ \sum_{\ell=0}^{p_i} \log \left( 1 + e^{-(\theta^T X_{i,\ell}) \Gamma_{i,\ell}} \right) \right] + \frac{\lambda}{2} \|\theta\|^2_2$$

subject to $\|\theta\|_0 \leq \kappa$.

(DSLogR)

where $X_i \in \mathbb{R}^{p_i \times n}$ and $\Gamma_i \in \mathbb{R}^{p_i}$ are the dataset and response vector of the $i$th node, $X_{i,\ell}$ is the column representation of $\ell$th row of $X_i$, and $\Gamma_{i,\ell}$ is the $\ell$th element of $\Gamma_i$. Clearly, both problems (DSLinR) and (DSLogR) are subclasses of problem (SCO). In both problems (DSLinR) and (DSLogR), 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.

1.1. Main contributions

The main idea behind \textsc{scot} is to transform problem (\textsc{SCO}) into an equivalent \textit{Distributed Mixed-Integer Nonlinear Programming (D-MINLP)} problem. By using the concept of the so-called Local Fusion Centers (LFCs) presented in [39], we recast problem (\textsc{SCO}) as a consensus optimization problem and introduce several constraints to model the sparsity constraint. Based on the Outer Approximation (OA) algorithm [17,22,26], \textsc{scot} consists of two main algorithms, namely, \textit{Distributed Primal Outer Approximation (DiPOA)} and \textit{Distributed Hybrid Outer Approximation (DiHOA)}. The DiPOA algorithm proposed in [40] extends the OA algorithm by embedding a fully decentralized algorithm, namely the Relaxed Hybrid Alternating Direction Method of Multipliers (RH-ADMM) [39]. The RH-ADMM assumes a particular hybrid architecture on the computational network, developed to solve distributed convex optimization problems. In particular, DiPOA solves the primal problem of the OA using the RH-ADMM algorithm and handles distributedly the demanding computational part of the OA algorithm that deals with the solution of convex NLPs. In practice, there exist many cases where a significant part of the solution time is spent on solving the NLP problems. For example, in Table 1 [26], it can be seen that OA spends more than 150 seconds on solving NLP problems when solving a moderate-size convex MINLP. Moreover, for inherently distributed problems in which the data is spread over a possibly large computational network, a single NLP will be challenging, if even possible, to solve in a classical centralized fashion. For such inherently decentralized problems, a distributed algorithm can offer great computational advantages.

Despite solving problem (\textsc{SCO}) distributedly, DiPOA is developed based on a multiple-tree OA algorithm whereby a BnB tree is built from scratch at each iteration of the algorithm. Therefore, most of the overall solution time of DiPOA is usually spent on solving

\begin{table}[h]
\centering
\begin{tabular}{ll}
\hline
Acronym & Meaning \\
\hline
API & Application Programming Interface  \\
BnB & Branch and Bound  \\
CLI & Command Line Interface  \\
DIHOA & Distributed Hybrid Outer Approximation  \\
DiPOA & Distributed Primal Outer Approximation  \\
D-MINLP & Distributed Mixed-Integer Nonlinear Programming  \\
DSLinR & Distributed Sparse Linear Regression  \\
DSLogR & Distributed Sparse Logistic Regression  \\
LFC & Local Fusion Center  \\
MILP & Mixed-Integer Linear Programming  \\
MINLP & Mixed-Integer Nonlinear Programming  \\
MIP & Mixed-Integer Programming  \\
MPI & Message Passing Interface  \\
NLP & Nonlinear Programming  \\
OA & Outer Approximation  \\
QCLP & Quadratically Constrained Linear Programming  \\
RH-ADMM & Relaxed Hybrid Alternating Direction Method of Multipliers  \\
SCO & Sparse Convex Optimization  \\
SCOT & Sparse Convex Optimization Toolkit  \\
SOS-1 & Specially Ordered Set of Type I  \\
\hline
\end{tabular}
\caption{Acronyms.}
\end{table}
MIP sub-problems. In this paper, we tackle the limitations of DiPOA by proposing DiHOA, a distributed algorithm that improves DiPOA performance by gradually building a single BnB tree to avoid constructing and solving many similar MILP problems from scratch. The main idea of DiHOA is to solve problem (SCO) by dynamically updating the MILP sub-problem, in a fashion similar to the LP/NLP-BnB presented by [41]. In principle, DiHOA starts with the multiple-tree search strategy up to a certain iteration and introduces high-quality cuts until a suitable event is triggered. Once the event is triggered, DiHOA switches from the multiple-tree search strategy to the single-tree search strategy that builds a single BnB tree, starting with multiple cuts initially introduced to its root node to augment the initial formulation. The single BnB tree then tightens up the integer relaxations by dynamically introducing more linear approximations (cuts) to the MILP problem. The multiple-tree search strategy is only applied in some initial iterations of the DiHOA since the MIP problems are typically much easier to solve, as the nonlinear constraints are only roughly represented through a few constraints. Moreover, starting the BnB search with a tighter approximation of the nonlinear constraints can result in a much smaller BnB tree, as 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. The main contributions of this work are summarized as follows:

- SCOT, a distributed software framework to model and solve problem (SCO).
- The distributed hybrid outer approximation algorithm to solve problem (SCO).
- Modeling and heuristic techniques to improve the efficiency of the proposed algorithm.
- A computational analysis of the proposed algorithms for various SCO real-world applications.

1.2. Paper organization

The paper is organized as follows. In Section 2, we introduce the distributed SCO problem by reformulating problem (SCO) into an equivalent MINLP problem. Section 3 presents the primal and dual problems used by SCOT algorithms. The DiHOA algorithm is proposed in Section 4. Section 5 introduces SCOT and its main components. Finally, the numerical comparison and algorithm analysis are presented in Section 6. Section 7 concludes the paper.

2. Distributed sparse convex optimization

Various modelling techniques are employed by SCOT to find a solution to problem (SCO) efficiently. First, SCOT transforms problem (SCO) into a consensus optimization problem and then multiple modelling techniques are used to handle the sparsity constraint.

2.1. Consensus optimization modelling

As defined in problem (SCO), the decision variables are shared between the nodes. Each node only has information to construct its objective function, while keeping the local
problem data private from other nodes. This distributed setting is a typical pattern in some significant learning and control applications. For instance, in distributed machine learning, an efficient technique to deal with large volumes of data consists of distributing the data over a network [35,38]. In this case, a significant reduction in memory size for computation can be achieved while keeping the same unknown parameters of the model. To decompose (SCO), we adhere to the concept of hypergraphs, which is a generalization of a regular graph in which an edge can join an arbitrary number of vertices. Multiple computational sources, called LFCs, can be employed by adopting this structure in the network. We consider a hypergraph \( H = (V, E) \) defined as follows: \( V = \{1, 2, \ldots, N\} \) is the set of nodes such that node \( i \) decides upon the values of vector variable \( x_i \); \( E = \{E_\ell \subset V : \ell = 1, \ldots, K\} \) is the set of hyperedges, where a hyperedge \( E_\ell \) connects all nodes \( i \in E_\ell \) and \( K \) is the number of hyperedges. Now we introduce the concept of a path in a hypergraph: a path \( p(i,j) = \langle E'_1, \ldots, E'_\ell \rangle \) connects nodes \( i \) and \( j \) if \( i \in E'_1, j \in E'_\ell \), and \( E'_l \cap E'_{l+1} \neq \emptyset \) for \( l = 1, \ldots, \ell - 1 \), and \( E'_l \in E \) 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 \( H \) is connected, meaning that for all \( i, j \in V \) there exists a path \( p(i,j) \) connecting \( i \) and \( j \). By using the hypergraph structure, the equivalent consensus formulation, C-SCO, for (SCO), is obtained as follows,

\[
\min_{x_1, \ldots, x_N} \sum_{i=1}^{N} f_i(x_i) \\
\text{subject to } Ax_i \leq b, \quad \forall i = 1, \ldots, N, \\
x_i = y_j, \quad \forall i \in E_j, \ E_j \in E \\
\|y_j\|_0 \leq \kappa, \quad \forall j = 1, \ldots, K
\]

where \( x_i \in \mathbb{R}^n \) are vectors of decision variables associated with the nodes and \( y_j \in \mathbb{R}^n \) are auxiliary variables associated with the LFCs, which are represented by the hyperedges.

### 2.2. Sparsity constraint modelling

Here, we present three modelling techniques implemented by SCOT to express the sparsity constraints, namely the Big-M, Specially Ordered Set of Type I (SOS-1), and a hybrid approach. These techniques are based on introducing a set of binary variables \( \delta \in \{0,1\}^n \) and appropriate constraints into the optimization problem which are discussed in the subsequent sections.

#### 2.2.1. Big-M method

The Big-M method is arguably the simplest technique for modelling 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 recast the sparsity constraint as the following inequalities,

\[
-M_j \delta_{jq} \leq y_{jq} \leq M_j \delta_{jq}, \quad \forall j = 1, \ldots, K, \quad \forall q = 1, \ldots, n
\]
\[
\sum_{q=1}^{n} \delta_{jq} \leq \kappa, \quad \forall j = 1, \ldots, K
\]
\[
\delta_{jq} \in \{0, 1\}, \quad \forall j = 1, \ldots, K, \quad \forall q = 1, \ldots, n
\]

where \(y_{jq}\) is the \(k\)th element of \(y_j\), \(\delta_j\), \(\forall j = 1, \ldots, K\), is a vector of binary variables whose \(k\)th element is denoted by \(\delta_{jq}\), and \(M_j\) is a constant assumed to be a valid upper bound for \(\|y_j\|_\infty\). In this case, if \(\delta_{jq} = 0\) then \(y_{jq} = 0\) and otherwise \(y_{jq}\) is nonzero. Thus, inequalities (1a) and (1b) impose the maximum number of nonzero variables in \(y_j\). The Big-M parameter \(M_j\) is not known a priori, and too small a value of \(M_j\) may lead to a sub-optimal solution. A large \(M_j\), on the other hand, will result in a weak continuous relaxation and strongly affect the number of nodes that need to be explored in BnB. Hence, a good choice of \(M_j\) affects the strength of the formulation, being critical for MIP algorithms to obtain high-quality bounds. In the context of learning and control applications, the Big-M value \(M_j\) can typically be computed from data in statistical learning tasks [8] and from the physical bounds in control applications [1].

### 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 (C-SCO) satisfies the following complementary constraints,

\[
(1 - \delta_{jq}) y_{jq} = 0, \quad \forall j = 1, \ldots, K, \quad \forall q = 1, \ldots, n \tag{2a}
\]
\[
\sum_{q=1}^{n} \delta_{jq} \leq \kappa, \quad \forall j = 1, \ldots, K
\]
\[
\delta_{jq} \in \{0, 1\}, \quad \forall j = 1, \ldots, K, \quad \forall q = 1, \ldots, n \tag{2b}
\]

which is equivalent to the sparsity constraint. In order for constraint (2a) to be satisfied, either \((1 - \delta_{jq})\) or \(y_{jq}\) must be zero. Such constraints can be modelled via integer optimization software using Specially Ordered Sets of Type I (SOS-1) [9] as follows,

\[
(y_{jq}, 1 - \delta_{jq}) : \text{SOS-1}, \quad \forall j = 1, \ldots, K, \quad \forall q = 1, \ldots, n. \tag{3}
\]

It is worth noting that such a constraint is not implemented explicitly employing algebraic equations in our problem representation, as in the Big-M method, but is handled internally by the sub-solver.

### 2.2.3. MINLP reformulation

The main reformulation that SCOT seeks to solve is expressed as,

\[
\begin{align*}
\min_{\gamma} \quad & \gamma \\
\text{subject to} \quad & \sum_{i=1}^{N} f_i(x_i) - \gamma \leq 0
\end{align*}
\]

(MINLP)
\[ \text{Ax}_i \leq b, \quad \forall i = 1, \ldots, N, \]
\[ \text{x}_i = y_j, \quad \forall i \in E_j, E_j \in \mathcal{E} \]
\[ -M_j \delta_{jq} \leq y_{jq} \leq M_j \delta_{jq}, \quad \forall j = 1, \ldots, K, \quad \forall q = 1, \ldots, n \]
\[ (y_{jq}, 1 - \delta_{jq}) : \text{SOS-1}, \quad \forall j = 1, \ldots, K, \quad \forall q = 1, \ldots, n \]
\[ \sum_{q=1}^{n} \delta_{jq} \leq \kappa, \quad \forall j = 1, \ldots, K \]
\[ \delta_{jq} \in \{0, 1\}, \quad \forall j = 1, \ldots, K, \quad \forall q = 1, \ldots, n \]

where \( \gamma \in \mathbb{R} \) is an auxiliary variable and the equivalent epigraph reformulation is favoured. Moreover, both SOS-1 and Big-M constraints are considered. Using both Big-M and SOS-1 constraints gives \text{SCOT} the ability to provide better \( M_j \) coefficients for the MIP solver. Problem (MINLP) is a mixed-integer nonlinear optimization problem with a separable structure and a linear objective function. In the following sections, we introduce a distributed formulation and algorithm that \text{SCOT} implements to solve problem (MINLP).

### 3. SCOT primal and dual problem

Here, we present the dual and primal problems and we discuss two algorithms that \text{SCOT} implements in the next section. Akin to other decomposition-based MINLP algorithms, \text{SCOT} decomposes problem (MINLP) into two main sub-problems, namely, the primal and dual problems \([26,31]\). We use the term *primal solution* and *primal bound* as the optimal solution and objective value of the primal problem respectively. Similarly, *dual solution* and *dual bound* are used for the dual problem. It should be noted that we use the terminology dual problem for a problem whose optimal solution provides a valid lower bound on the optimal objective value of problem (MINLP) and whose feasible set contains all feasible solutions of problem (MINLP).

The primal solution is assumed to satisfy all linear, nonlinear, and consensus constraints of problem (MINLP) to a given tolerance. The current best-known primal solution found by the algorithm is referred to as the *incumbent solution*, and its objective value is the current primal bound. In the standard OA algorithm, the primal problem is a convex NLP problem; owing to the distributed nature of problem (MINLP), the primal problem of \text{SCOT} is a distributed convex NLP problem which will be discussed later.

A solution point whose objective value provides a valid lower bound for the optimum of problem (SCO), but not necessarily satisfying all constraints, is referred to as a dual solution. Like the standard OA algorithm, \text{SCOT} obtains dual solutions by solving relaxed problems that approximate the nonlinear constraints with polyhedral outer approximations. Depending on the type of outer approximations, the dual problem can be MILP, MIQP, or Mixed-Integer Quadratically Constrained Linear (Quadratic) (MIQCL(Q)P) problems. Moreover, the dual bound is the best possible objective value of the dual problem. The primal and dual sub-problems are then iteratively solved by proper MIP algorithms. The MIP algorithms are distinguished depending on how the sub-problems are constructed, solved, and coordinated. Regardless of the solution algorithms adopted by \text{SCOT}, the dual and primal sub-problems are two primary components of the algorithms.
The main problem reformulation that SCOT attempts to solve by default is problem (MINLP), which enforces both Big-M and SOS-1 constraints and epigraph-reformulation. The separability of nonlinear functions in problem (MINLP) allows SCOT to employ an alternative formulation, the so-called lifted formulation [28]. In this context, we use the lifted formulation for each nonlinear function, $f_i$, which results in tighter outer approximations when approximating nonlinear functions [24,28,44].

According to the idea of lifted formulation and also following the procedure presented in [28], at each iteration $k$, we construct the dual problem as the following MIP problem,

$$
\begin{align*}
\min \ & \sum_{i=1}^{N} \gamma_i \\
\text{subject to} \ & \tilde{f}_i^k(x_i) - \gamma_i \leq 0, \quad \forall \ x_i^k \in \mathcal{X}_i^k, \quad \forall \ i = 1, \ldots, N, \\
& Ax_i \leq b, \quad \forall \ i = 1, \ldots, N, \\
& x_i = y_j, \quad \forall \ i \in \mathcal{E}_j, \mathcal{E}_j \in \mathcal{E} \\
& \frac{1}{M} \gamma_j q \leq y_j q \leq \frac{1}{M} \gamma_j q, \quad \forall \ j = 1, \ldots, K, \quad \forall \ q = 1, \ldots, n \\
& \sum_{q=1}^{n} \delta_{jq} \leq \kappa, \quad \forall \ j = 1, \ldots, K \\
& \delta_{jq} \in \{0, 1\}, \quad \forall \ j = 1, \ldots, K, \quad \forall \ q = 1, \ldots, n
\end{align*}
$$

where $\gamma_i \in \mathbb{R}$, $i = \{1, \ldots, N\}$, are new auxiliary decision variables, $x_i^k$ is a feasible point that satisfies linear and consensus constraints, obtained at iteration $k$, $\tilde{f}_i^k(x_i)$ is the outer approximation of $f_i(x_i)$ around $x_i^k$ and $\mathcal{X}_i^k$ is defined as,

$$
\mathcal{X}_i^k = \left\{ x_i^\ell : Ax_i^\ell \leq b, \quad \forall \ \ell \in \{1, \ldots, k\} \right\},
$$

The problem (Dual) is a relaxation of problem (MINLP) since approximations of nonlinear constraints are used and its objective value is a lower bound of (MINLP). At each iteration of SCOT algorithms, $k$, a new outer approximation is generated by each node of the network and cooperatively added to problem (Dual) producing a tighter representation of the nonlinear constraints. The quality of outer approximations generated by $\tilde{f}_i^k(x_i)$ directly impacts the convergence of the 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 $f_i(x_i)$ functions, we can express constraints $\tilde{f}_i^k(x_i) - \gamma_i \leq 0$ as,

$$
\begin{align*}
f_i(x_i^k) + \nabla f_i(x_i^k)^T (x_i - x_i^k) - \gamma_i \leq 0,
\end{align*}
$$

which are linear inequalities. In case the nonlinear functions, $f_i(x_i)$, are strongly convex functions, SCOT utilizes the following quadratic inequalities,

$$
\begin{align*}
f_i(x_i^k) + \nabla f_i(x_i^k)^T (x_i - x_i^k) + \frac{m_i^k}{2} \| (x_i - x_i^k) \|^2 - \gamma_i \leq 0
\end{align*}
$$
where \( m^k_i > 0 \) is a constant such that \( \nabla^2 f_i(x_i) \succeq m^k_i I \). With \( m^k_i > 0 \), it is clear that the cut given by (5) is stronger than the cut given by (4). However, the quadratic cuts (5) tend to result in more challenging sub-problems in BnB. Therefore, there can still be a computational advantage of the linear cuts.

**Remark 3.1:** For general strongly convex functions, \( m^k_i \) is not obtained easily. However, in some practical problems found in statistical learning and control, the computation of \( m^k_i \) 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, \( m^k_i \) 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, \( m^k_i \) can be computed from the regularization term.

A crucial step to forming the outer approximations is the computation of the approximation points \( x^k_i \) for which various strategies and methods exist. One of the well-known methods to obtain \( x^k_i \) is fixing the local binary decision variables of problem (MINLP), \( \delta_{jq} = \delta^k_{jq} \), and solving the resulting nonlinear optimization problem. The problem of solving (MINLP) for fixed binary variables is the primal problem of SCOT, which, at each iteration \( k \), is defined as,

\[
\begin{align*}
\min_{x_1, \ldots, x_N, y_1, \ldots, y_K} & \sum_{i=1}^{N} \gamma_i \\
\text{subject to} & \quad f_i(x_i) - \gamma_i \leq 0 \\
& \quad Ax_i \leq b, \quad \forall \ i = 1, \ldots, N, \\
& \quad x_i = y_j, \quad \forall \ i \in E_j, \ E_j \in \mathcal{E} \\
& \quad -M_j \delta_{jq}^k \leq y_{jq} \leq M_j \delta_{jq}^k, \quad \forall \ j = 1, \ldots, K, \ \forall \ q = 1, \ldots, n.
\end{align*}
\]

(Primal)

The optimal solution of problem (Primal) 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 (Primal).

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 (Primal), some numerical considerations have to be taken into account. Particularly, in this case, in addition to all linear and nonlinear constraints, the consensus constraints \( x_i = y_j, \ \forall \ i \in E_j, \ E_j \in \mathcal{E} \) 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 the computational network is large. Another challenge in solving the primal problem distributedly is the communication burden between the nodes of the network and the LFCs.
4. Distributed hybrid outer approximation

In this section, we propose the Distributed Hybrid Outer Approximation (DiHOA) algorithm to solve problem (MINLP). In essence, DiHOA is developed based on the LP/NLP-based BnB algorithm proposed in [41]. The LP/NLP-based BnB algorithm is an implementation of the standard OA algorithm, where only a single BnB tree is built and outer approximations are added dynamically to the MIP master problem. Since only one BnB tree is constructed during the solution procedure, the LP/NLP-based BnB method is also called the single-tree OA algorithm. Similarly, the standard implementation of the OA algorithm where a BnB tree is constructed at each iteration of the algorithm is called multiple-tree OA.

Before discussing the DiHOA algorithm, we briefly present Distributed Primal Outer Approximation (DiPOA) which is proposed in [40]. DiPOA is the baseline algorithm developed to solve problem (MINLP) which, in essence, extends the standard OA algorithm so that the main computational parts related to the NLP sub-problems are handled in a distributed fashion. From the numerical point of view, DiPOA is a hierarchical algorithm that consists of three main computational layers, namely, primal, cutting-plane manager, and master levels.

The primal level deals with the nonlinear optimization part, particularly problem (Primal), consisting of two sub-levels responsible for a specific computational task. The first sub-level aims to simultaneously solve multiple local nonlinear optimization problems in a fully decentralized fashion. The local nodes then synchronously communicate to the second sub-level, which is responsible for another phase of computations. The second sub-level is usually responsible for aggregating the solution of local NLP problems connected to each LFC by a series of unconstrained NLP problems. The solution of this level is then sent to the first sub-level until a consensus is reached.

The main purpose of the cutting-plane manager level is to generate and manage the OA linearizations, which are obtained around the generated feasible points provided by the primal level. This level is also responsible for adding the linearizations into the master’s problem (Dual).

Finally, the master’s level corresponds to solving problem (Dual) based on the cuts generated by the cutting-plane manager. The master MIP problem approximates the nonlinear functions of problem (MINLP). As the number of cuts increases, this approximation improves until a good piece-wise outer approximator is achieved. The binary solution of the master level is then sent to the primal level, which, together with the nonlinear optimization problems, solves another set of NLP problems.

Generally speaking, when solving problem (C-SO) with DiPOA, most of the total solution time is usually spent on solving the MIP master’s problem. In such a scenario, the MIP problems are similar in consecutive iterations since they only differ by a few linear constraints. In particular, at iteration \(k\) of DiPOA, a new feasible point is provided by the primal, around which a new linear approximation constraint is generated and added to the master’s problem. In the next iteration, \(k + 1\), the master’s problem is reconstructed and solved from scratch.

Based on what we proposed in [40] and according to [41], we develop the DiHOA algorithm to avoid constructing many similar MIP BnB trees. The main idea of DiHOA is to iteratively build a single branch-and-bound tree whereby the primal problem is
solved distributedly, while dynamically updating problem (Dual) without reconstructing the branch-and-bound tree. However, the single-tree OA algorithm may lead to a large BnB 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 problem (Primal) 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 invoked, 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 when the MIP integer relaxation results in a feasible integer solution to problem (MINLP).

A detailed description of the DiHOA algorithm is summarized in Algorithm 1. It can be observed in Algorithm 1 that DiHOA consists of three primary computational phases, namely, Initialization, MultipleTreeSearch, and SingleTreeSearch steps. After the algorithm is successfully initialized, DiHOA starts a multiple-tree strategy with second-order outer approximations 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, \( k_{\text{switch}} \), at which a single-tree search strategy is started. In the following, we describe each computational step in detail.

4.1. Initialization step

The initialization step is started by solving the integer relaxation of problem (MINLP) to construct problem (Dual) in the first iteration of the algorithm. The integer relaxation of problem (MINLP) is a consensus optimization problem that is solved using the RH-ADMM algorithm. In case the relaxation obtains a feasible solution with respect to problem (MINLP), we terminate DiHOA with the optimal solution. Otherwise, we generate \( N \) first-order outer approximations by using the local information available in each computational node of the network and construct the first MIP dual problem according to (Dual).

4.2. 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 \( k_{\text{switch}} \) is a large number, a pure multiple-tree
Algorithm 1: DiHOA Algorithm

(1) Initialization
1.1 obtain a relaxed solution $\tilde{x}_i^0, \tilde{y}_j^0, \tilde{\delta}_j^0$, $i \in \{1, \ldots, N\}, j \in \{1, \ldots, K\}$ by solving an integer relaxation of problem (MINLP)
1.2 generate $N$ outer approximations according to (4) and construct problem (Dual)
1.3 store the generated outer approximations.
1.4 set $k = 1$, $ub^0 = +\infty$, $lb^0 = -\infty$

(2) MultipleTreeSearch
\[ \text{while } lb^k - lb^{k-1} > \epsilon \text{ do} \]
\[ \quad 2.1 \text{ solve problem (Dual) and obtain a dual solution } \delta_j^k \]
\[ \quad 2.2 \text{ } lb^k \leftarrow \sum_{i=1}^N \gamma_i \]
\[ \quad 2.3 \text{ solve problem (Primal) and obtain a primal solution } x_i^k, y_j^k \]
\[ \quad 2.4 \text{ } ub^k \leftarrow \min \left( \sum_{i=1}^N f_i(x_i^k), ub^{k-1} \right) \]
\[ \quad 2.5 \text{ If } ub^k - lb^k \leq \tau, \text{ return } x_i^k \text{ as an optimal solution } x^* \]
\[ \quad 2.6 \text{ generate and store outer approximations according to (5) and update problem (Dual) by adding new outer approximations} \]
\[ \quad 2.7 \text{ } k \leftarrow k + 1 \]
\[ \text{end} \]

(3) SingleTreeSearch
3.1 start BnB search for the MIP problem obtained in the last step with all second-order outer approximations accumulated in the root
\[ \text{while } ub^k - lb^k > \epsilon \text{ do} \]
\[ \quad 3.2 \text{ if a new feasible integer solution, } \delta_j^k, \text{ is found, then } \delta_j^k \leftarrow \tilde{\delta}_j^k \text{ and update} \]
\[ \quad \text{ } lb^k \text{ according to the current BnB tree} \]
\[ \quad 3.3 \text{ solve problem (Primal) and update the primal solution } x_i^k, y_j^k \]
\[ \quad 3.4 \text{ } ub^k \leftarrow \min \left( \sum_{i=1}^N f_i(x_i^k), ub^{k-1} \right) \]
\[ \quad 3.5 \text{ generate outer approximations according to (4) and add them to the current and open nodes of BnB} \]
\[ \quad 3.6 \text{ resolve the integer relaxation problem for the node which resulted in the integer combination} \]
\[ \quad 3.7 \text{ } k \leftarrow k + 1 \text{ and continue exploring BnB tree} \]
\[ \text{end} \]

algorithm with second-order outer approximations is obtained. Otherwise, the resulting algorithm becomes the single-tree OA. Therefore, $k_{\text{switch}}$ should be determined in such a way that maximizes the DiHOA performance. To do so, we introduce an event-triggered scheme that selects $k_{\text{switch}}$ based on the difference between two consecutive lower bounds (i.e. $lb^k - lb^{k-1}$). 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 $\epsilon > 0$ and triggers a switching event, $E_\epsilon$, if $lb^k - lb^{k-1} \leq \epsilon$. As soon as $E_\epsilon$ 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.

4.3. Single-tree search step

The single-tree search step is activated for $k > k_{\text{switch}}$ after the $E_s$ event is triggered. 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 problem (Dual). In each node of the BnB tree, a 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 problem (Primal) 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 (MINLP) or until the node can be pruned off. Finally, since all variables in problem (SCO) are bounded, then the assumptions A1–A3 in [19] and assumptions in [26] are valid and Slater’s constraint qualification holds for problem (MINLP) when the binary variables are fixed. Hence, no NLP subproblem will be infeasible and the convergence of Algorithm 1 is ensured.

5. SCOT: a software framework for sparse optimization

This section discusses the technical features of SCOT, including architecture and design, main components, basic syntax, and usage. SCOT is entirely written in C++17 and at its core uses the Message Passing Interface (MPI) [21] to perform distributed computation operations, communication, and message-passing between nodes of the network. To handle local NLP optimization problems, SCOT relies on various open-source solvers, such as OSQP [42] and IPOPT [47]. SCOT also implements a truncated Newton’s method [36] to solve unconstrained optimization problems. Moreover, SCOT integrates commercial and open-source MIP solvers such as Gurobi [23] and CBC to solve MIP dual problems. To solve the distributed NLP problem (Primal), SCOT implements the RH-ADMM algorithm proposed in [39] using the main MPI operations. Finally, both DiPOA and DiHOA algorithms are implemented within SCOT and are available through SCOT Python API, ScotPy, or SCOT Command Line Interface (CLI).

5.1. Architecture

A high-level overview of SCOT architecture, its main layers, and components are shown in Figure 1. 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
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.

5.1.1. 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 software industry and has been widely used for decades. We refer the interested reader to [29] for more details.

5.1.2. 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.

5.1.3. 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 distribute solve problem (Primal) 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 DiHOA (Algorithm 1) 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 the 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** for solving the MIP problem (**Dual**).

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.

### 5.1.4. 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 inter-process communications.

### 5.2. Basic syntax and usage

Here, 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,

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

# 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", MPI_rank=0,
                sparse_constraint_cardinality=5, problem_type=ProblemType.CLASSIFICATION)

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

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

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,
# Set SCOT settings.
scot_settings = ScotSettings(relative_gap=1e-5, time_limit=100, verbose=True,
algorithm=AlgorithmType.DIHOA)

Listing 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)
# Solve the optimization problem
status_code = solver.run()

Listing 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. Numerical experiments

In this section, we evaluate the SCOT performance by comparing it to various state-of-the-art MINLP solvers equipped with single-tree, multiple-tree, and nonlinear BnB algorithms. We consider SHOT [32] and BONMIN [11] as decomposition-based and KNITRO [14] as nonlinear BnB solvers. However, one should keep in mind that these are general-purpose solvers, and unlike SCOT they are not tailored for the specific problem structure of sparse convex optimization problems. From the application point of view, we focus on problems (DSLinR) and (DSLogR) with data distributed over a network of computational nodes. The benchmark results are presented using solution profiles showing the number of individual problems that a solver is able to solve as a function of time [27,32].

6.1. 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. The source code of the solver is available on https://github.com/Alirezalm/scot. 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 [13] 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 $\epsilon = 10^{-5}$. For the MINLP solvers included in GAMS, we chose the same value $\epsilon$ for optCA and optCR.

6.2. Benchmark results

We generate $N$ random local datasets for problems (DSLinR) and (DSLogR) with zero mean and unit $\ell_2$ norm for each column. We perform the numerical benchmarks based
on different MINLP solver settings using solution profiles to compare SCOT with several MINLP solvers with different settings. The chosen settings for SCOT, SHOT, BONMIN, and KNITRO are reported in Table 2.

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 2 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 3 represents settings for each benchmark scenario where \( n_{\text{min}} \) and \( n_{\text{max}} \) are the minimum and the maximum number of features, \( p_{\text{min}} \) and \( p_{\text{max}} \) are the minimum and the maximum number of data points for each computational node \( N \), \( n_p \) is the total number of problems, and \( p_{\text{tot}} \) is the total number of data points considering all computational nodes. We assumed 80% and 90% sparsity for each scenario presented in Table 3. However, since the number of variables is generated randomly for each problem instance, the value of \( \kappa \) may vary. Nonetheless, to ensure the reproducibility of the numerical results, we fixed the random seed for each scenario.

Figures 2–8 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 2–4 for small to medium size problem instances. In both sparsity levels, the DiPOA and DiHOA algorithms perform better than other MINLP solvers. It can be observed in Figure 4 that, for

### Table 2. MINLP solver settings.

| Solver   | Algorithm    | Name       | MIP solver | NLP solver  |
|----------|--------------|------------|------------|-------------|
| SCOT     | DIPOA        | SCOT-MT    | GUROBI     | RH-ADMM     |
| SCOT     | DIPOA        | SCOT-ST    | GUROBI     | RH-ADMM     |
| 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      |

### Table 3. Benchmark settings for solution profiles.

| Scenario settings | NSW | NMA | PW | PMA | N | NP | PTOT |
|-------------------|-----|-----|----|-----|---|----|------|
| SC                | 20  | 30  | 1,000 | 2,500 | 2 | 20 | 5,000 |
| 2                 | 20  | 30  | 1,000 | 5,000 | 2 | 20 | 10,000 |
| 3                 | 25  | 50  | 1,000 | 5,000 | 2 | 20 | 10,000 |
| 4                 | 25  | 100 | 1,000 | 10,000 | 4 | 20 | 40,000 |
| 5                 | 25  | 100 | 1,000 | 20,000 | 4 | 20 | 80,000 |
| 6                 | 25  | 100 | 1,000 | 50,000 | 4 | 20 | 200,000 |
| 7                 | 25  | 200 | 1,000 | 50,000 | 6 | 20 | 300,000 |
Figure 2. Benchmark results for scenario 1. (a) 90% sparsity and (b) 80% sparsity.

Figure 3. Benchmark results for scenario 2. (a) 90% sparsity and (b) 80% sparsity.

Figure 4. Benchmark results for scenario 3. (a) 90% sparsity and (b) 80% sparsity.
Figure 5. Benchmark results for scenario 4. (a) 90% sparsity and (b) 80% sparsity.

Figure 6. Benchmark results for scenario 5. (a) 90% sparsity and (b) 80% sparsity.

Figure 7. Benchmark results for scenario 6. (a) 90% sparsity and (b) 80% sparsity.
larger problem instances, the performance gap between SCOT and other MINLP solvers increases.

Figures 5–8 depict the solution profiles for the scenarios 4–7 accounting for medium to large problem instances. In these scenarios, all MINLP solvers 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 than DiPOA as the performance gap between them increases for large problem instances. For example, in scenario 7 with 80% of sparsity, DiHOA solved 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 could provide an efficient and reliable solution.

6.3. Evaluation of sparsity modelling 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 introduced in Section 2: 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 $\kappa = [\frac{n}{2}]$. The average computational time for 100 randomly generated DSLogR problems for distinct $n$ appears in Table 4. 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 paper, the Big-M and hybrid sparsity modelling strategies are more efficient than the SOS-1 strategy. However, the choice of strategy may depend on the specific problem characteristics. For
Table 4. Computational results with different sparsity modelling strategies.

| Strategy CPU time (s) | n | Big-M | SOS-1 | Hybrid |
|-----------------------|---|-------|-------|--------|
| n                    |   |       |       |        |
| 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 |

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 while maintaining a tight bound in the optimization problem.

6.4. 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 [16] 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 [2] 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 $\kappa$. 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 5, 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 $\kappa$ is large. However, it is noteworthy that general-purpose solvers perform relatively well, especially when $\kappa$ is small. This observation can be attributed to the fact that the dataset used in this study is not as large as

Table 5. 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.

| $\kappa$ | SCOT (s) | MINLP solvers (s) |
|----------|----------|-------------------|
|          | 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   |
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.

7. Conclusion

In this work, we introduced SCOT and DiHOA algorithm that SCOT implements to solve problem (SCO). The DiHOA implements an outer approximation algorithm to solve an MINLP equivalent to the SCO problem, combining single- and multiple-tree outer approximation with a decentralized algorithm for solving convex nonlinear subproblems, which generates primal solutions and cuts for the master dual problem. The numerical benchmark results and comparison to state-of-the-art MINLP solvers indicated that SCOT equipped with DiHOA can be efficiently used for solving sparse convex optimization problems in different application domains. The performance and efficiency of the solver were achieved by augmenting SCOT with modern convex and mixed-integer optimization techniques. Continuing work aims at enhancing SCOT effectiveness by means of the development of decentralized mixed-integer algorithms and new heuristic techniques to obtain outer approximations.

Disclosure statement

No potential conflict of interest was reported by the author(s).

Funding

This work was funded in part by Fundação de Amparo à Pesquisa e Inovação do Estado de Santa Catarina (FAPESC) [grant number 2021TR2265] and Coordenação de Aperfeiçoamento de Pessoal de Nível Superior (CAPES, Brazil) [project PrInt CAPES-UFSC 698503P1]. Financial support from Digital Futures at KTH and C3.ai Digital Transformation Institute for the project ‘AI techniques for Power Systems Under Cyberattacks’ is also gratefully acknowledged.

Notes on contributors

Alireza Olama obtained his M.Sc. degree in control engineering from Shiraz University of Technology (SUTECH) in 2017. He is currently pursuing a Ph.D. in automation and systems engineering at the Federal University of Santa Catarina (UFSC), Florianópolis, Brazil, where his research focuses on distributed mathematical optimization in distributed control and machine learning.

Eduardo Camponogara received a Ph.D. degree in electrical and computer engineering from Carnegie Mellon University, USA, in 2000. After being a postdoctoral fellow at the Institute for Complex Engineered Systems, USA, he joined the faculty of the Department of Automation and Systems Engineering at the Federal University of Santa Catarina, Brazil, in 2002. His research interests include systems optimization, distributed decision-making, and traffic control engineering.

Jan Kronqvist is an Assistant Professor in Optimization and Systems Theory at the Department of Mathematics at KTH Royal Institute of Technology in Stockholm, Sweden, since 2021. Before joining KTH, Jan was a Newton International Fellow at Imperial College London. Jan did his Ph.D. at Åbo Akademi University in Finland and received the Rosenbrock Prize 2021 for best paper in the journal Optimization and Engineering. In 2018, Jan was one of the winners of the COIN-OR Cup for their work on the SHOT solver. Jan is an editorial board member of the Journal of Global Optimization.
References

[1] R.P. Aguilera, G. Urrutia, R.A. Delgado, D. Dolz, and J.C. Agüero, *Quadratic model predictive control including input cardinality constraints*, IEEE. Trans. Automat. Contr. 62 (6) (2017), pp. 3068–3075.

[2] V. Arzamasov, K. Böhm, and P. Jochem, *Towards concise models of grid stability*, IEEE International Conference on Communications, Control, and Computing Technologies for Smart Grids (SmartGridComm), Aalborg, Denmark. 2018, pp. 1–6.

[3] D. Bertsimas and R. Cory-Wright, *A scalable algorithm for sparse portfolio selection*, INFORMS J. Comput. 34 (3) (2022), pp. 1489–1511.

[4] D. Bertsimas, J. Dunn, L. Kapelevich, and R. Zhang, *Sparse regression over clusters: SparClur*, Optim. Lett. 16 (2) (2022), pp. 433–448.

[5] D. Bertsimas and A. King, *Logistic regression: From art to science*, Stat. Sci. 32 (3) (2017), pp. 367–384.

[6] D. Bertsimas, A. King, and R. Mazumder, *Best subset selection via a modern optimization lens*, Ann. Stat. 44 (2) (2016), pp. 813–852.

[7] D. Bertsimas and N. Mundru, *Sparse convex regression*, INFORMS J. Comput. 33 (1) (2021), pp. 262–279.

[8] D. Bertsimas, J. Pauphilet, and B. Van Parys, *Sparse classification: A scalable discrete optimization perspective*, Mach. Learn. 110 (11–12) (2021), pp. 3177–3209.

[9] D. Bertsimas and R. Weismantel, *Optimization Over Integers*, Dynamic Ideas Belmont, MA, 2005.

[10] D. Bienstock, *Computational study of a family of mixed-integer quadratic programming problems*, Math. Program. 74 (1996), pp. 121–140.

[11] P. Bonami, L.T. Biegler, A.R. Conn, G. Cornuéjols, I.E. Grossmann, C.D. Laird, J. Lee, A. Lodi, F. Margot, and N. Sawaya, et al. *An algorithmic framework for convex integer nonlinear programs*, Discrete. Optim. 5 (2) (2008), pp. 186–204.

[12] S. Boyd, N. Parikh, E. Chu, B. Peleato, and J. Eckstein, et al. *Distributed optimization and statistical learning via the alternating direction method of multipliers*, Found. Trends® Mach. Learn. 3 (2011), pp. 1–122.

[13] M.R. Bussieck and A. Meeraus, *General algebraic modeling system (gams)*, in *Modeling Languages in Mathematical Optimization*. Applied Optimization, vol. 88, Kallrath J., eds., Springer, Boston, MA, 2004. pp. 137–157. doi:10.1007/978-1-4613-0215-5_8.

[14] R.H. Byrd, J. Nocedal, and R.A. Waltz, *KNITRO: An Integrated Package for Nonlinear Optimization*, in *Large-Scale Nonlinear Optimization*. Nonconvex Optimization and Its Applications, Di Pillo G., Roma, M., eds., vol. 83. Springer, Boston, MA. https://doi.org/10.1007/0-387-30065-1_4

[15] S.S. Chen, D.L. Donoho, and M.A. Saunders, *Atomic decomposition by basis pursuit*, SIAM Rev. 43 (1) (2001), pp. 129–159.

[16] D. Dua and C. Graff, *UCI machine learning repository* (2017). Available at https://archive.ics.uci.edu/ml.

[17] M.A. Duran and I.E. Grossmann, *An outer-approximation algorithm for a class of mixed-integer nonlinear programs*, Math. Program. 36 (3) (1986), pp. 307–339.

[18] Y.C. Eldar and G. Kutyniok, *Compressed Sensing: Theory and Applications*, Cambridge University Press, 2012.

[19] R. Fletcher and S. Leyffer, *Solving mixed integer nonlinear programs by outer approximation*, Math. Program. 66 (1–3) (1994), pp. 327–349.

[20] S. Foucart and H. Rauhut, *An invitation to compressive sensing*, in *A Mathematical Introduction to Compressive Sensing*. Applied and Numerical Harmonic Analysis. Birkhäuser, New York, NY. https://doi.org/10.1007/978-0-8176-4948-7_1
[21] W. Gropp, E. Lusk, and A. Skjellum, *Using MPI: Portable Parallel Programming with the Message-Passing Interface*, Vol. 1, MIT Press, 1999.

[22] I. Grossmann, *Review of nonlinear mixed-integer and disjunctive programming techniques*, Optim. Eng. 3 (3) (2002), pp. 227–252.

[23] L. Gurobi Optimization, *Gurobi optimizer reference manual* (2020). Available at [http://www.gurobi.com](http://www.gurobi.com).

[24] H. Hijazi, P. Bonami, and A. Ouorou, *An outer-inner approximation for separable mixed-integer nonlinear programs*, INFORMS. J. Comput. 26 (1) (2014), pp. 31–44.

[25] Y. Kim, J. Kim, and Y. Kim, *Blockwise sparse regression*, Stat. Sin. 16 (2) (2006), pp. 375–390. Available at [http://www.jstor.org/stable/24307549](http://www.jstor.org/stable/24307549).

[26] J. Kronqvist, D.E. Bernal, and I.E. Grossmann, *Using regularization and second order information in outer approximation for convex MINLP*, Math. Program. 180 (1–2) (2020), pp. 285–310.

[27] J. Kronqvist, D.E. Bernal, A. Lundell, and I.E. Grossmann, *Review and comparison of solvers for convex MINLP*, Optim. Eng. 20 (2) (2019), pp. 397–455.

[28] J. Kronqvist, A. Lundell, and T. Westerlund, *Reformulations for utilizing separability when solving convex MINLP problems*, J. Glob. Optim. 71 (3) (2018), pp. 571–592.

[29] A. Lundell and J. Kronqvist, *Integration of polyhedral outer approximation algorithms with MIP solvers through callbacks and lazy constraints*, in Emmerich, M. T. M., Deutz, A. H., Hille, S. C., and Sergeyev, Y. D., eds., *Proceedings LeGO : 14th International Global Optimization Workshop*, Leiden, the Netherlands, 18–21 September 2018. American Institute of Physics. doi:10.1063/1.5088979.

[30] A. Lundell, J. Kronqvist, and T. Westerlund, *The supporting hyperplane optimization toolkit for convex minlp*, J. Glob. Optim. 84 (1) (2022), pp. 1–41.

[31] B. Legat, O. Dowson, J.D. Garcia, and M. Lubin, *MathOptInterface: A data structure for mathematical optimization problems*, INFORMS. J. Comput. 34 (2021), pp. 672–689.

[32] A. Lundell, J. Kronqvist, and T. Westerlund, *The supporting hyperplane optimization toolkit for convex minlp*, J. Glob. Optim. 84 (1) (2022), pp. 1–41.

[33] B.K. Natarajan, *Sparse approximate solutions to linear systems*, SIAM J. Comput. 24 (2) (1995), pp. 227–234.

[34] A. Nedić and J. Liu, *Distributed optimization for control*, Annu. Rev. Control Robot. Auton. Syst. 1 (1) (2018), pp. 77–103.

[35] J. Nocedal and S. Wright, *Numerical Optimization*, Springer Science & Business Media, 2006.

[36] I. Notarnicola, Y. Sun, G. Scutari, and G. Notarstefano, *Distributed big-data optimization via block-iterative convexification and averaging*, IEEE 56th Annual Conference on Decision and Control (CDC), Melbourne, Australia, Dec. 2017, pp. 2281–2288.

[37] G. Notarstefano, I. Notarnicola, and A. Camisa, et al. *Distributed optimization for smart cyber-physical networks*, Found. Trends® Syst. Control. 7 (2019), pp. 253–383.

[38] A. Olama, N. Bastianello, P.R. Mendes, and E. Camponogara, *Relaxed hybrid consensus ADMM for distributed convex optimisation with coupling constraints*, IET Control Theory Appl. 13 (17) (2019), pp. 2828–2837.

[39] A. Olama, E. Camponogara, and P. Mendes, *Distributed primal outer approximation algorithm for sparse convex programming with separable structures*, J. Glob. Optimiz. 86 (2023), pp. 637–670. doi:10.1007/s10898-022-01266-5.

[40] I. Quesada and I.E. Grossmann, *An LP/NLP based branch and bound algorithm for convex MINLP optimization problems*, Comput. Chem. Eng. 16 (10–11) (1992), pp. 937–947.

[41] B. Stellato, G. Banjac, P. Goulart, A. Bemporad, and S. Boyd, *OSQP: An operator splitting solver for quadratic programs*, Math. Program. Comput. 12 (4) (2020), pp. 637–672.

[42] X. Sun, X. Zheng, and D. Li, *Recent advances in mathematical programming with semi-continuous variables and cardinality constraint*, J. Oper. Res. Soc. China 1 (1) (2013), pp. 55–77.
[44] M. Tawarmalani and N.V. Sahinidis, *A polyhedral branch-and-cut approach to global optimization*, Math. Program. 103 (2) (2005), pp. 225–249.

[45] R. Tibshirani, *Regression shrinkage and selection via the lasso*, J. R. Stat. Soc. B. Methodol. 58 (1996), pp. 267–288.

[46] A.M. Tillmann, D. Bienstock, A. Lodi, and A. Schwartz, *Cardinality minimization, constraints, and regularization: A survey*, arXiv preprint arXiv:2106.09606 (2021).

[47] A. Wächter and L.T. Biegler, *On the implementation of an interior-point filter line-search algorithm for large-scale nonlinear programming*, Math. Program. 106 (1) (2006), pp. 25–57.