# Decentralized Optimization over Tree Graphs

Yuning Jiang · Dimitris Kouzoupis · Haoyu Yin · Moritz Diehl · Boris Houska<sup>\*</sup>

Received: date / Accepted: date

Abstract This paper presents a decentralized algorithm for non-convex optimization over tree-structured networks. We assume that each node of this network can solve small-scale optimization problems and communicate approximate value functions with its neighbors based on a novel multi-sweep communication protocol. In contrast to existing parallelizable optimization algorithms for non-convex optimization the nodes of the network are neither synchronized nor assign any central entity. None of the nodes needs to know the whole topology of the network, but all nodes know that the network is treestructured. We discuss conditions under which locally quadratic convergence rates can be achieved. The method is illustrated by running the decentralized asynchronous multi-sweep protocol on a radial AC power network case study.

**Keywords** Decentralized Optimization  $\cdot$  Tree Graph  $\cdot$  Dynamic Programming

### **1** Introduction

Large-scale optimization problems over sparse networks arise in many applications such as resource allocation problems [1], smart grid control problems [2], traffic coordination problems [3,4], power system operation problems [5,6], and statistical learning problems [7]. In terms of existing numerical algorithms,

Y. Jiang, H. Yin and B. Houska School of Information Science and Technology ShanghaiTech University, Shanghai, China E-mail: jiangyn, yinhy, borish@shanghaitech.edu.cn

D. Kouzoupis and M. Diehl Department of Microsystems Engineering (IMTEK) University of Freiburg, Germany E-mail: dimitris.kouzoupis, moritz.diehl@imtek.uni-freiburg.de

<sup>\*</sup>Corresponding author.

which can solve such large-scale optimization problems, one typically distinguishes between distributed and decentralized methods [8], as reviewed below.

Distributed optimization algorithms typically parallelize most of their operations, but communicate results to a central coordinator. This usually requires one to synchronize the whole network, as the central operations can often only be performed after receiving all results from the agents of the network. For example, many variants of the Alternating Direction Method of Multipliers (ADMM) [7,8,9] as well as the Augmented Lagrangian Alternating Direction Inexact Newton (ALADIN) method [10] alternate between solving small-scale optimization problems, which can be done in parallel by the agents of a network, and solving large scale linear equation systems, which is typically done by a central coordinator. Here, the central solver runs advanced sparse linear algebra routines, which can, optionally, parallelize part of their operations [11, 12, 13]. A complete overview of existing distributed optimization algorithms would go beyond the scope of this paper, but an overview of distributed optimization methods, with a particular focus on augmented Lagrangian approaches, can be found in [14,15,16]. Notice that modern distributed optimization algorithms are applicable to convex as well as non-convex optimization problems. For example, convergence conditions for ADMM to local minimizers of non-convex optimization problems can be found in [17]. Similarly, ALADIN has been designed for solving non-convex optimization problems and conditions for global convergence to local minimizers can be found in [10]. In the context of the current paper, we also mention that there exist distributed variants of interior point methods for convex optimization that have been tailored for tree sparse graphs, as analyzed in [18].

In contrast to the above reviewed methods for distributed optimization, decentralized optimization methods do not require a central coordination step. In the most general case, these algorithms are not synchronized and the agents might not even know the whole network structure. However, one usually requires that all agents can communicate with their neighbors [8]. Classical decentralized optimization methods are often based on dual decomposition [19, 20], but there also exist decentralized consensus variants of ADMM [9,21]. However, in contrast to distributed optimization methods, decentralized optimization algorithms often converge for convex problems only and have a linear or even sublinear convergence behavior [7,22,1]. For instance, linear convergence conditions of a fully decentralized ADMM method for consensus optimization over networks has been established in [9]. A unified convergence analysis for decentralized ADMM in dependence on the network structure can also be found in [21].

In summary, there has been a huge amount of research on distributed optimization algorithms, but there are, at the current status of research, no generic asynchronous decentralized optimization algorithms available that are i) applicable to large-scale non-convex optimization problems and ii) locally equivalent to Newton-type methods such that locally superlinear or quadratic convergence rates can be expected. Therefore, this paper asks the question whether such decentralized algorithms can be constructed at all, at least for special classes of networks. Here, our focus is on networks with tree graphs, which arise in some (but not all) applications such as traditional optimal control problems [23] or receding horizon control problems [24], where linear trees occur, scenario multi-stage MPC problems [25, 26, 27], which have less trivial tree structures, or radial power grid networks that possess non-trivial tree structure too [28, 29].

Because there exist dynamic programming (DP) methods [30] as well as approximate DP methods [31,32,33], which can exploit the structure of optimization problems over tree-topologies, we briefly review these methods as well as their closely related min-sum algorithms [34,35] in Section 2. The main contribution of this paper is then presented in Section 3, where we develop a multi-sweep algorithm for solving tree-structured optimization problems. Moreover, Section 4 presents a fully decentralized and asynchronous variant of this novel multi-sweep method, which can still achieve locally quadratic convergence rates. These theoretical developments are illustrated by a case study in Section 5, for a non-trivial radial power grid optimization problem.

### Notation and Preliminaries

Throughout this paper a couple of existing results from the field of parametric nonlinear optimization are used [36]. In order to briefly review these existing results, we consider a general (twice continuously differentiable) parametric equality constrained optimization problem of the form

$$F(x) = \min_{z} f(x, z) \quad \text{s.t.} \quad c(x, z) = 0 \mid \lambda \tag{1}$$

Here,  $f : \mathbb{R}^n \times \mathbb{R}^m \to \mathbb{R}$  and  $c : \mathbb{R}^n \times \mathbb{R}^m \to \mathbb{R}^{n_c}$  denote the twice continuously differentiable objective and constraint functions,  $x \in \mathbb{R}^n$  a parameter, and  $z \in \mathbb{R}^m$  the primal optimization variable. Notice that we use the syntax " $| \lambda$ " after an equality constraint to say that  $\lambda$  denotes the multiplier of this constraint. Consequently, in the context of (1), we have  $\lambda \in \mathbb{R}^{n_c}$ .

**Definition 1.1** We say that a (local) minimizer  $z^*(x)$  of (1) satisfies the linear independence constraint qualification (LICQ) at a given point  $x \in \mathbb{R}^n$ , if the constraint Jacobian matrix

$$\frac{\partial c(x, z^{\star}(x))}{\partial z}$$

has full-rank.

At this point, we recall that the LICQ condition is sufficient to ensure that minimizers of (1) are KKT points. That is, if  $z^*(x)$  is a local minimizer of (1)

at x at which LICQ holds, then there exists a unique multiplier  $\lambda^{\star}(x) \in \mathbb{R}^{n_c}$  such that (see [36, Thm. 12.1])

$$0 = \nabla_z L(x, z^*(x), \lambda^*(x)) \tag{2}$$

$$0 = c(x, z^{\star}(x)) , \qquad (3)$$

where  $L: \mathbb{R}^n \times \mathbb{R}^m \times \mathbb{R}^{n_c} \to \mathbb{R}$  denotes the Lagrangian function,

$$L(x, z, \lambda) = f(x, z) + \lambda^{\top} c(x, z) .$$

In this context, another important regularity conditions is the so-called second order sufficient condition:

**Definition 1.2** We say that a (local) primal-dual minimizer  $(z^*(x), \lambda^*(x))$  of (1) satisfies the second order sufficient condition (SOSC) at x, if for any vector  $v \in \mathbb{R}^n$  with  $v \neq 0$  and

$$\frac{\partial c(x, z^{\star}(x))}{\partial z}v = 0$$

it follows that  $v^{\top} \nabla_z^2 L(x, z^{\star}(x), \lambda^{\star}(x)) v > 0.$ 

In some of the technical derivations of this paper, we assume that both LICQ and SOSC hold at a given point of an optimization problem. This motivates the following definition.

**Definition 1.3** We say that a local minimizer  $z^*(x)$  of (1), together with the multiplier  $\lambda^*(x)$ , is a regular minimizer at x, if it satisfies the LICQ and the SOSC conditions.

The above definition is useful, because it allows us to establish the following regularity result for the value function F, which follows by applying the implicit function theorem [36, Thm. A2] (or its generalized version [37]) to the first order necessary KKT conditions (2)-(3); see also [38] for details.

**Lemma 1.1** If  $(z^*(x), \lambda^*(x))$  denotes a regular minimizer and multiplier of (1) at x, then F is twice continuously differentiable at x and we have

 $\nabla_x F(x) = L_x(x) \qquad and \qquad \nabla_x^2 F(x) = L_{xx}(x) - L_{xu}(x) L_{uu}(x)^{-1} L_{ux}(x)$ 

with shorthands  $L_x(x) = \frac{\partial L}{\partial x}(x, z^{\star}(x), \lambda^{\star}(x))$  as well as

$$L_{xx}(x) = \frac{\partial^2 L}{\partial x^2}(x, z^*(x), \lambda^*(x)) ,$$
  

$$L_{ux}(x) = L_{xu}(x)^\top = \begin{pmatrix} \frac{\partial^2 L}{\partial z \partial x}(x, z^*(x), \lambda^*(x)) \\ \frac{\partial c}{\partial x}(x, z^*(x)) \end{pmatrix} ,$$
  
and  

$$L_{uu}(x) = \begin{pmatrix} \frac{\partial^2 L}{\partial z^2}(x, z^*(x), \lambda^*(x)) \frac{\partial c}{\partial z}(x, z^*(x))^\top \\ \frac{\partial c}{\partial z}(x, z^*(x)) & 0 \end{pmatrix} , \qquad (4)$$

where  $L_{uu}(x)$  is invertible at x. Moreover, there exists an open neighborhood  $\mathcal{N} \subseteq \mathbb{R}^n$  of  $x \in int(\mathcal{N})$  such that F is twice differentiable on  $\mathcal{N}$  and such that  $(z^*(x), \lambda^*(x))$  is a regular solution of (1) at y for all  $y \in \mathcal{N}$ .

Notice that the statement of this lemma can be strengthened further under the additional assumption that the second derivatives of f and c are Lipschitz continuous; see [38,37] for details.

**Corollary 1.1** Let  $(z^*(x), \lambda^*(x))$  be a regular minimizer and multiplier of (1) at x. If the second derivatives of f and c are locally Lipschitz continuous, then there exists an open neighborhood  $\mathcal{N}$  of  $x^*$  such that the second derivatives of F are locally Lipschitz continuous on  $\mathcal{N}$ .

Throughout this paper we construct algorithms for solving nonlinear equality constrained problems of the form

$$\min_{x} F(x) \quad \text{s.t.} \quad C(x) = 0 \mid \kappa , \qquad (5)$$

where C typically denotes a twice Lipschitz-continuously differentiable (in the applications of this paper even affine) consensus constraint. Let

$$\Phi(x,y) = F(y) + \mathbf{O}(\|x-y\|^3)$$

denote a local model of F at x and consider an iteration of the form

$$x^{k+1} = \underset{y}{\operatorname{argmin}} \Phi(x^k, y) \quad \text{s.t.} \quad C(y) = 0 \mid \kappa^{k+1}$$
(6)

started at an initial point  $x_0 \in \mathbb{R}^n$ . The following theorem is—at least in very similar versions—known and well-established [36, Thm. 18.4]. However, for the sake of completeness, we provide a short proof in Appendix A.

**Theorem 1.1** Let  $x^*$  be a regular minimizer of (5); that is, such that LICQ and SOSC are satisfied. If F, C and  $\Phi(x, \cdot)$  are twice differentiable functions with locally Lipschitz continuous second derivatives with uniform Lipschitz constant for all x in a neighborhood of  $x^*$ , then the iterates  $x^k$  of (6) converge locally to  $x^*$  with quadratic convergence rate; that is,

$$||x^{k+1} - x^{\star}|| \le \mathbf{O}\left(||x^k - x^{\star}||^2\right)$$

for all  $k \in \mathbb{N}$  whenever the initialization  $x^0$  is in a sufficiently small neighborhood of  $x^*$ .

#### 2 Optimization over tree graphs

This section introduces optimization problems over tree graphs and briefly discusses the advantages and disadvantages of existing dynamic programming methods, which can be used to solve them numerically.

# $2.1~{\rm Tree~graphs}$

Let  $(\mathcal{N}, \mathcal{E})$  denote a graph with node set  $\mathcal{N} = \{1, ..., N\}$  and edge set  $\mathcal{E} \subseteq \mathcal{N} \times \mathcal{N}$ . In the following, we assume that this graph is undirected; that is,  $\mathcal{E}$  is symmetric:  $(i, j) \in \mathcal{E}$  if and only if  $(j, i) \in \mathcal{E}$ . We call  $(\mathcal{N}, \mathcal{E})$  a tree graph if there exists for every  $i \in \mathcal{N} \setminus \{1\}$  exactly one way to walk from the first node to the *i*-th node via the edges of the graph without passing any node more than once. Here, the first node is called the root of the tree graph. However, it is important to keep in mind that the definition of a tree graph does not depend on how the nodes are enumerated. In particular, any node in a tree can be a root as long as we re-enumerate the nodes accordingly. An example for a simple tree structure is shown in the sketch below.



Fig. 1 Example for a tree graph with N = 6 nodes.

In the following, we denote by  $\mathcal{N}_i = \{j \in \mathcal{N} \mid (i, j) \in \mathcal{E}\}$  the set of neighbors and by  $\mathcal{L} = \{i \in \mathcal{N} \mid |\mathcal{N}_i| \leq 1\}$  the set of nodes with at most one neighbor. For example, for the network in Figure 1, we have  $\mathcal{L} = \{1, 3, 5, 6\}$ . In the following, we additionally use the notation

$$\mathcal{L}^{\bullet} = \mathcal{L} \setminus \{1\}$$

which is called the set of leaves. Notice that the definition of  $\mathcal{L}$  does not depend on which node is assigned as root. This is in contrast to the set  $\mathcal{L}^{\bullet}$  of leaves, which may not contain the root node.

**Proposition 2.1** The nodes of a tree graph  $(\mathcal{N}, \mathcal{E})$  can be enumerated in such a way that the graph  $(\mathcal{N}, \mathcal{E}^+)$ , with

$$\mathcal{E}^+ = \{ (i, j) \in \mathcal{E} \mid i < j \} ,$$

is still a tree graph.

The proof of the above proposition is constructive: we can start at any node and call it the root by assigning the label 1. Next, the root enumerates its children in increasing order, the children enumerate their children, and so on, until all nodes have a number and such that the set

$$\mathcal{C}_i = \{ j \in \mathcal{N}_i \mid j \ge i \}$$

corresponds to the set of children of the i-th node. Notice that this enumeration procedure also ensures that the equation

$$\mathcal{L}^{\bullet} = \{ i \in \mathcal{N} \mid |\mathcal{C}_i| = 0 \}$$

holds; that is, leaves have no children. Last but not least, every node with number  $i \geq 2$  has a unique parent node  $\pi_i \in \mathcal{N}_i \setminus \mathcal{C}_i$ .

### 2.2 Tree-structured optimization problems

This paper concerns structured optimization problems of the form

$$V^* = \min_{x} \sum_{i \in \mathcal{N}} F_i(x_i) \quad \text{s.t.} \quad \begin{cases} \forall (i,j) \in \mathcal{E}^+, \\ S_{i,j}x_i = S_{j,i}x_j \end{cases}$$
(7)

Here, the functions  $F_i : \mathbb{R}^{n_i} \to \mathbb{R}$  denote objective functions,  $S_{i,j} \in \mathbb{R}^{n_{i,j} \times n_i}$ and  $S_{j,i} \in \mathbb{R}^{n_{i,j} \times n_j}$  given connectivity matrices and  $(\mathcal{N}, \mathcal{E})$  a tree graph, where  $\mathcal{E}^+$  is defined as in Proposition 2.1. Notice that in the most general case, there are no further assumptions on the functions  $F_i$  needed as long as the minimizer of (7) exists, although for some of the algorithmic developments below, we will work with stronger local regularity assumptions on the functions  $F_i$ , such as twice Lipschitz-continuous differentiability. Notice that in the context of (7) the function  $F_i$  could denote the minimum value of a parametric optimization problem [39,40] that is solved by the *i*-th node in order to evaluate  $F_i$  at the parameter  $x_i$ ,

$$F_i(x_i) = \min f_i(x_i, z_i) \quad \text{s.t.} \quad c_i(x_i, z_i) , \qquad (8)$$

where the function  $c_i$  can be used to enforce nonlinear equality constraints. As long as the regularity assumptions from Lemma 1.1 and Corollary 1.1 are satisfied for the functions  $f_i$  and  $c_i$ , these results can be used to ensure that  $F_i$  is locally twice continuously differentiable.

Now, the goal of this paper is to develop decentralized optimization algorithms for solving (7) allowing neighbor-to-neighbor communication only. Here, we are particularly interested in algorithms that specify decentralized communication protocols that are independent of the particular structure of the tree  $(\mathcal{N}, \mathcal{E})$ . This means that none of nodes should be required to know the complete structure of the tree.

#### 2.3 Dynamic programming

Problem (7) can be solved by implementing the dynamic programming recursion [23,41]

$$V_i(p) = \min_{x_i} F_i(x_i) + \sum_{j \in \mathcal{C}_i} V_j(S_{i,j}x_i) \text{ s.t. } S_{i,\pi_i}x = p$$
(9)

for all  $i \neq 1$ . Notice that this optimization problem can be solved by the *i*-th node as soon as it has received the value functions  $V_j$  from all its children. Moreover, the recursion starts at the leaves  $i \in \mathcal{L}^{\bullet}$ , because these nodes do not have children. In the last backward recursion step, the root node solves the optimization problem

$$V^* = \min_{x_1} F_1(x_1) + \sum_{j \in \mathcal{C}_1} V_j(S_{1,j}x_1)$$
(10)

finding the optimal value  $V^*$  of (7). Finally, the root initializes a so-called forward sweep by sending linear combinations,  $S_{1,j}x_1^*$ , to its children such that they can find their solutions  $x_j^*$ , send linear combinations of these solutions to their children, and so on, until all nodes know the optimal solution. Notice that a desirable advantage of dynamic programming is that the globally optimal solution is found. Moreover, the number of communication steps of dynamic programming is relatively low: we need only one backward sweep from the leaves to the root and one forward sweep from the root to the leaves. However, one disadvantage of dynamic programming is that one needs to construct the functions  $V_i$ , which is only possible with high numerical precision if further regularity assumptions on the functions  $F_i$  are introduced [42,43]. If the dimension  $n_{i,j}$  of the coupling variables is small, this is not a big problem, but, in general, dynamic programming is affected by the so-called curse of dimensionality.

Remark 2.1 Variants of dynamic programming for general tree structured networks have been developed in [34] under the name *min-sum algorithms*. An approximate variant of these min-sum algorithms for piecewise quadratic optimization problems can be found in [35], which is related to the developments in this paper, although we consider a much more general class of non-convex optimization problems.

#### 2.4 Run-time considerations

In order to briefly discuss the run-time properties of dynamic programming, we introduce the following definition.

**Definition 2.1** We denote with d(i, j) the minimum number of edges over which one has to walk in order to get from any node  $i \in \mathcal{N}$  to a node  $j \in \mathcal{N}$ .

**Proposition 2.2** The function  $d : \mathcal{N} \times \mathcal{N} \to \mathbb{N}$ , as introduced in Definition 2.1, is a metric on  $(\mathcal{N}, \mathcal{E})$ . Moreover, d is invariant under re-enumeration of the nodes.

*Proof* It is easy to check that d(i, j) = 0 if and only if i = j, d(i, j) = d(j, i), and  $d(i,k) \leq d(i,j) + d(j,k)$  for all  $i, j, k \in \mathcal{N}$  and d is indeed a metric. The additional invariance statement follows trivially from the fact that the definition of d depends only on how the nodes are connected by edges—not on how they are enumerated.

Now, if the dynamic programming method is run in synchronous mode; that is, such that every node executes at most one dynamic programming step per sampling time  $\delta > 0$ , then the root node 1 receives all value functions after time  $\delta D$ , where

$$D = \max d(1, j)$$

denotes the depth of the enumerated tree. Because the forward sweep takes equally long, the total run-time of a synchronized dynamic programming method with sampling time  $\delta$  is given by

$$T = 2\delta D , \qquad (11)$$

where D denotes the depth of the tree. This implies in particular that the runtime of dynamic programming (DP) is not invariant under re-enumeration of the nodes, because the depth of a tree depends on which node is assigned as the root node 1.

*Example 2.1* Let us come back to the network from Figure 1, whose depth is given by

$$\max_{i} d(1,j) = 3$$

Equation (11) implies that the synchronized DP run-time of this network is  $6\delta$ . However, if we would re-enumerate the nodes of this network, such that the node with label 2 becomes the new root, the synchronized DP run-time improves to  $4\delta$ .

### 3 Multi-sweep method

In order to avoid the curse of dimensionality of standard dynamic programming, we replace (9) by an approximate dynamic programming recursion. For this aim, we introduce parametric auxiliary optimization problems of the form

$$\Omega_{i,k}(p) = \min_{x_i} F_i(x_i) + \sum_{j \in \mathcal{N}_i \setminus \{k\}} W_{j,i}(S_{i,j}x_i)$$
  
s.t.  $S_{i,k}x_i = p$  (12)

for any  $k \in \mathcal{N}_i$  and recursively constructed approximation functions  $W_{i,k} \approx \Omega_{i,k}$ . At this point, one can, in principle, admit all kinds of approximation functions, but the main limitation is that the functions  $W_{i,k}$  should be representable in a suitable storage format, such that they can be sent over the network links with reasonable effort. For example, one could construct quadratic approximations that can be stored in the form of Hessian matrix, gradient vector and a scalar. Notice that if we set  $k = \pi_i$  in (12) and propagate the

approximation in backward mode from the leaves to the root, then this construction is such that

$$W_{i,\pi_i} \approx \Omega_{i,\pi_i} \approx V_i$$
;

that is,  $W_{i,\pi_i}$  can be interpreted as an approximation of the function  $V_i$ . Notice that one can repeat the approximate dynamic programming recursion multiple times in order to refine the accuracy of the approximation. In order to develop such a multi-sweep variant, we assume that the *i*-th node has two modes: a backward and a forward mode. Moreover, the *i*-th node (with  $i \neq 1$ ) is initially set to backward mode and it is initialized with a model  $W_{\pi_i,i}$  of its parent. If no such model is available, we may, for example, set  $W_{\pi_i,i} = 0$ . Now, if the *i*-th node is in backward mode, it waits until it receives models  $W_{j,i}$  from all children  $j \in C_i$ , and then solves the auxiliary optimization problem

$$\min_{y_i} F_i(y_i) + \sum_{j \in \mathcal{N}_i} W_{j,i}(S_{i,j}y_i) .$$
 (13)

Let  $y_i^{\star}$  denote a minimizer of this problem. Next, the approximation  $W_{i,\pi_i} \approx \Omega_{i,\pi_i}$  in (12) is constructed in such a way that we have

$$W_{i,\pi_i}(p) = \Omega_{i,\pi_i}(p) + \mathbf{O}(\|S_{i,\pi_i}y_i^{\star} - p\|^{q+1}), \qquad (14)$$

where q denotes the order of the approximation, and sent to the parent node  $\pi_i$ . After this, the *i*-th node is set to forward mode. In this mode, it waits until a new model  $W_{\pi_i,i}$  is sent from the parent node, then (13) is solved once more in order to update  $y^*$ . The corresponding updated models  $W_{i,j} \approx \Omega_{i,j}$ , with

$$W_{i,j}(p) = \Omega_{i,j}(p) + \mathbf{O}(\|S_{i,j}y_i^* - p\|^{q+1}), \qquad (15)$$

are sent to all children  $j \in C_i$ . Notice that the protocol for the root node, i = 1, is completely analogous with the only difference being that this node immediately switches to forward mode as soon as the models from all children are received. The complete multi-sweep procedure is summarized in Algorithm 1.

#### 3.1 Termination conditions

Notice that once the protocols on all nodes are "switched on", Algorithm 1 keeps on updating its model functions forwever. If one wishes to introduce a termination condition, this can be done by modifying Line 9 of Algorithm 1 as follows: if the *j*-th node does not only send the model function  $W_{j,\pi_j}$  but also the projected solution  $S_{j,\pi_j}y_j^*$  to its parent after solving (13), then its parent can evaluate the residual

$$r_i = \max_{j \in \mathcal{C}_i} \left\| S_{i,j} y_i^\star - S_{j,\pi_j} y_j^\star \right\|_{\infty} .$$

### **Algorithm 1** Multi-sweep method of order q

### 1: Initialization:

- 2: Set all nodes to backward mode.
- 3: If  $i \neq 1$ , choose an initial model function  $W_{\pi_i,i}$ .
- 4: Repeat the following protocol on every node  $i \in \mathcal{N}$ :
- 5: **IF** the node is in backward mode:
- 6: Wait for model updates  $W_{j,i}$  from all children  $j \in C_i$ .
- 7: If i = 1, switch to forward mode and **BREAK**
- 8: Solve (13) and construct  $W_{i,\pi_i} \approx \Omega_{i,\pi_i}$  such that (14) holds.
- 9: Send  $W_{i,\pi_i}$  to the parent and switch to forward mode.
- 10: **ELSE** (the node is in forward mode):
- 11: If  $i \neq 1$ , wait for the model update  $W_{\pi_i,i}$  from the parent.
- 12: Solve (13) and construct  $W_{i,j} \approx \Omega_{i,j}$  such that (15) holds.
- 13: Send  $W_{i,j}$  to all nodes  $j \in C_i$ , switch to backward mode.

#### 14: **ENDIF**

Thus, if all these residual values are forwarded, the root node can evaluate the infinity norm, r, of the primal equality constraint violation as

$$r = \max_{i \in \mathcal{N} \setminus \mathcal{L}} r_i . \tag{16}$$

If this constraint violation is small,  $r \leq \epsilon$ , for a small numerical tolerance  $\epsilon > 0$ , the root can send out a termination message that can be forwarded by the children until all nodes terminate. Notice that this termination condition merely ensures that the primal consensus constraint violation is small, but, in general, this is not sufficient to ensure that the solutions  $y_i^*$  of the nodes are close to a minimizer  $x_i^*$  of (7) upon termination. Nevertheless, if one assumes that further regularity assumptions hold, for example if all minimizers of (7) are regular KKT points, one can show that the termination condition  $r \leq \epsilon$  ensures

$$\|x^{\star} - y^{\star}\| \le \mathbf{O}(\epsilon),$$

see [40, 36] for details.

#### 3.2 Construction of model functions

As mentioned in the previous section, Algorithm 1 can, in the most general case, be applied without further assumptions on the functions  $F_i$  as long as one ensures that all minimizers are well-defined. However, if one is interested in constructing practical algorithms with  $q \geq 1$ , one might be interested in matching the first q derivatives of the functions  $W_{i,k}$  and  $\Omega_{i,k}$ , which is only possible if the  $F_i$ s are sufficiently often differentiable and if the minimizers

of (12) are regular KKT points for all possible evaluation points p. A practical example is summarized below.

Example 3.1 Let us assume that F is three times continuously differentiable and has bounded third derivatives and that (12) has a regular parametric minimizer such that  $\Omega_{i,k}$  is twice differentiable with bounded third order weak derivatives (see Corollary 1.1). In this case, the models

$$W_{i,k}(p) = \frac{1}{2} p^{\top} H_{i,k} p + g_{i,k}^{\top} p + \sigma_{i,k} \|p - S_{i,k} y_i^{\star}\|^3 + \text{const.}$$

can be constructed by setting

$$H_{i,k} = \nabla^2 \,\Omega_{i,k}(S_{i,k}y_i^\star)$$

and

$$g_{i,k} = \nabla \Omega_{i,k}(S_{i,k}y_i^\star) - H_{i,k}S_{i,k}y_i^\star .$$

Notice that these first and second order derivatives can be computed easily by using Lemma 1.1. Here,  $\sigma_{i,k} \geq 0$  is a cubic regularization constant, which is chosen such that  $\sigma_{i,k} \geq \frac{1}{6} \|\nabla^3 \Omega_{i,k}(p)\|$  is a bound on the third order (weak) derivatives of  $\Omega_{i,k}$  on a suitably defined validity domain of the model. Notice that a Taylor expansion of the function  $\Omega_{i,k}$  at the point  $S_{i,k}y_i^*$  yields the relation

$$\Omega_{i,k}(p) \leq \underbrace{\Omega_{i,k}(S_{i,k}y_{i}^{\star}) + y_{i}^{\star}S_{i,k}^{\top}H_{i,k}S_{i,k}y_{i}^{\star}}_{\text{const.}} + g_{i,k}^{\top}p + \frac{1}{2}p^{\top}H_{i,k}p + \sigma_{i,k}\|p - S_{i,k}y_{i}^{\star}\|^{3}.$$
(17)

Thus, our particular construction of  $W_{i,k}$  is such that  $W_{i,k}(p) \ge \Omega_{i,k}(p)$  as long as the constant offset of  $W_{i,k}$  is chosen appropriately. Moreover, the associated multi-sweep method has order 2, since the above Taylor expansion based construction implies that

$$W_{i,k}(p) = \Omega_{i,k}(p) + \mathbf{O}(||p - S_{i,k}y_i^{\star}||^3).$$

Notice that, in the above example, one only needs to send symmetric matrices  $H_{i,k}$ , vectors  $g_{i,k}$  and  $y_i^*$ , and the regularization constant  $\sigma_{i,k}$  over the link (i, k), as constant offsets of objective functions do not affect the optimal solutions of (13). The complexity for storing these three variables, including the matrix  $H_{i,k}$ , is given by  $\mathbf{O}(n_{i,k}^2)$  recalling that  $n_{i,k}$  denotes the number of coupling variables between the *i*-th and the *k*-th node of the network.

#### 3.3 Conservation laws and convergence

The convergence properties of Algorithm 1 depend on the particular construction of the approximations  $W_{i,k} \approx \Omega_{i,k}$  and their relation to the exact value functions  $V_{i,k}$ , which we define recursively as

$$V_{i,k}(p) = \min_{x_i} F_i(x_i) + \sum_{j \in \mathcal{N}_i \setminus \{k\}} V_{j,i}(S_{i,j}x_i)$$
  
s.t.  $S_{i,k}x = p$  (18)

for all  $k \in \mathcal{N}_i$  and all  $i \in \mathcal{N}$ . In this context, the following general conservation laws are useful for the composition of convergence conditions.

Lemma 3.1 Algorithm 1 has the following properties.

1. If all approximations  $W_{i,k} \approx \Omega_{i,k}$  are lower bounds, such that  $W_{i,k}(p) \leq \Omega_{i,k}(p)$  for all p, then

$$W_{i,k}(p) \le V_{i,k}(p)$$

holds globally for all p, for all  $k \in \mathcal{N}_i$  all  $i \in \mathcal{N}$ , and during all iterations. 2. The above statement also holds after replacing all  $\leq$  signs with  $\geq$  signs; that is, the conservation of upper bounds holds, too.

*Proof* For the case  $|\mathcal{N}| = 1$  the statement of the lemma is trivial and we, thus, assume  $|\mathcal{N}| > 1$ . The proof of the first statement of this lemma follows by a tree-structured induction that starts with  $i \in \mathcal{L}^{\bullet}$  and propagates through the tree. Here, our induction start uses that the inequality<sup>1</sup>

$$W_{i,\pi_i}(p) \leq \Omega_{i,\pi_i}(p) = V_{i,\pi_i}(p)$$

holds for all  $i \in \mathcal{L}$  and their associated parent nodes  $\pi_i \in \mathcal{N}_i$ . Next, our induction assumption is that

$$W_{k,i}(p) \le V_{k,i}(p)$$

holds at a given node  $i \in \mathcal{N}$  for all children  $k \in \mathcal{C}_i$ . Now, the definition of  $\Omega_{i,k}$ and  $V_{i,k}$  in (12) and (18) implies that

$$\Omega_{i,\pi_i}(p) \le V_{i,\pi_i}(p) \quad \Rightarrow \quad W_{i,\pi_i}(p) \le V_{i,\pi_i}(p) , \qquad (19)$$

where the latter statement can be interpreted as an intermediate induction conclusion yielding that the inequality

$$W_{i,\pi_i}(p) \le V_{i,\pi_i}(p)$$

holds for all  $i \in \mathcal{N} \setminus \{1\}$ . Similarly, the same induction argument can be repeated in forward mode, which yields that

$$W_{i,k}(p) \le V_{i,k}(p)$$

<sup>&</sup>lt;sup>1</sup> The assumption  $|\mathcal{N}| > 1$  ensures that  $\pi_i$  exists and is well-defined for all  $i \in \mathcal{L}^{\bullet}$ .

also holds for all children  $k \in C_i$  and all  $i \in \mathcal{N} \setminus \mathcal{L}^{\bullet}$ . The proof of the second statement of the lemma is completely analogous, as we can replace all  $\leq$  signs with  $\geq$  signs without altering the logic of the proof.

The above lemma can be used as a basis for convergence proofs of Algorithm 1. For example, a locally quadratic convergence statement for second order variants of Algorithm 1 can be summarized as follows.

**Theorem 3.1** Let us assume that the approximation functions in Algorithm 1 satisfy  $W_{i,k}(p) \ge \Omega_{i,k}(p)$ . If the functions  $F_i$  as well as the functions  $W_{i,k}$  are all twice differentiable with Lipschitz continuous second derivatives, if Algorithm 1 has order q = 2 and if all leaves are initialized in a local neighborhood of a regular minimizer  $x^*$  of (7); that is, such that  $y_i^*$  is in a local neighborhood of  $x_i^*$  for all  $i \in \mathcal{L}^{\bullet}$ , then the iterates of Algorithm 1 converge with locally quadratic convergence rate.

**Proof** Because  $x^*$  is assumed to be regular minimizer of (7) and because we assume that the functions  $F_i$  are twice differentiable with Lipschitz continuous second derivatives, a recursive application of Corollary 1.1 to the dynamic programming recursion yields that the functions  $V_{i,k}$  are locally twice differentiable with Lipschitz continuous second derivatives. Next, because the second statement in Lemma 3.1 ensures that

$$W_{i,k}(p) \ge \Omega_{i,k}(p) \quad \Rightarrow \quad W_{i,k}(p) \ge V_{i,k}(p) ,$$

$$(20)$$

the iterates  $y_i^*$  are stable and remain in a neighborhood of  $x_i^*$ . Moreover, because we assume that Algorithm 1 has order q = 2, we have

$$W_{i,k}(p) \le \Omega_{i,k}(p) + \mathbf{O}(\|p - S_{i,k}y_i^{\star}\|^3)$$

and Lemma 3.1 can be used to propagate lower bounds, too, finding

$$W_{i,k}(p) \le V_{i,k}(p) + \mathbf{O}(\|p - S_{i,k}y_i^\star\|^3) .$$
(21)

Thus, by using inequalities (20) and (21), it follows that

$$W_{i,k}(p) = V_{i,k}(p) + \mathbf{O}(||p - S_{i,k}y_i^{\star}||^3)$$

for all  $k \in \mathcal{N}_i$  all  $i \in \mathcal{N}$ . Consequently, Algorithm 1 is locally equivalent to the exact dynamic programming method [16] up to terms of order 3. Moreover, the functions  $V_{i,k}$  and  $W_{i,k}$  are all locally twice differentiable with Lipschitz continuous second derivatives. Consequently, using once more that  $x^*$  is a regular minimizer, we can apply Theorem 1.1 to show that the iterates converge with locally quadratic convergence rate.

Remark 3.1 Notice that the conditions of Theorem 3.1 are satisfied for the approximation functions that have been constructed in Example 3.1. Moreover under the additional assumptions that the cubic regularization constants  $\sigma_{i,k}$  are sufficiently large, the third derivatives of  $F_i$  are bounded, and all KKT points of (13) are regular, one can show that Algorithm 1 converges globally to stationary points of (7)—this convergence result is obtained in analogy to Nesterov's cubic regularization method [44].

### 4 Simultaneous multi-sweep method

This section develops a variant of Algorithm 1 that is invariant with respect to permutations of the enumeration of the nodes of the tree graph. Here, the main idea is to start an approximate dynamic programming recursion at all nodes  $i \in \mathcal{L}$  simultaneously without assigning a root recalling that the definition of  $\mathcal{L}$  does not depend on the enumeration of the nodes. This leads to a simultaneous multi-sweep method as summarized in Algorithm 2.

Algorithm 2 Simultaneous multi-sweep method

- 1: Initialization of node *i*: Set  $\mathcal{R} = \emptyset$  and  $\ell = 0$ .
- 2: Repeat the following protocol on every node  $i \in \mathcal{N}$ :
- 3: If  $\ell = 0$ , denote with  $\mathcal{J} \subseteq \mathcal{N}_i$  the indices of all neighbors that have sent updates  $W_{j,i}$  after the last reset of the collection  $\mathcal{R}$  and update

$$\mathcal{R} \leftarrow \mathcal{R} \cup \mathcal{J}$$
 .

If  $|\mathcal{R}| = |\mathcal{N}_i|$  set  $\ell = i$ .

- 4: If  $\ell > 0$  and  $\ell \neq i$ , wait until a model update  $W_{\ell,i}$  from the  $\ell$ -th node is arriving.
- 5: If  $\ell > 0$ , solve (13) and construct  $W_{i,j} \approx \Omega_{i,j}$  for all indices  $j \in \mathcal{N}_i$  such that

$$W_{i,j}(p) = \Omega_{i,j}(p) + \mathbf{O}(\|S_{i,j}y_i^{\star} - p\|^{q+1}).$$

If  $\ell \neq i$ , send  $W_{i,j}$  to all neighbors  $j \in \mathcal{N}_i \setminus \{\ell\}$ . Otherwise, send  $W_{i,j}$  to all neighbors  $j \in \mathcal{N}$ . Set  $\ell = 0$ , and reset  $\mathcal{R} = \emptyset$ .

6: If  $|\mathcal{R}| = |\mathcal{N}_i| - 1$  with  $\{k\} = \mathcal{N}_i \setminus \mathcal{R}$ , solve (13) and construct  $W_{i,k} \approx \Omega_{i,k}$  such that

$$W_{i,k}(p) = \Omega_{i,k}(p) + \mathbf{O}(\|S_{i,k}y_i^{\star} - p\|^{q+1}).$$

If  $k \neq i$ , apply a short randomly chosen time delay: if the the line (i, k) is not blocked, block the line (k, i) and send  $W_{i,k}$  to node k, set  $\ell = k$  and unblock all lines again. Otherwise, if the line (i, k) is blocked, skip.

Notice that the communication protocol for the nodes  $i \in \mathcal{N}$  does not specify a-priori which neighbors act as children and which as parent. Here, every node keeps a local integer variable  $\ell$ , which is set to 0 whenever the node is in backward mode. While being in backward mode, Line 3 allows this node to collect updated models from all of its neighbors. This step assumes that Node *i* has a buffer such that local copies of all arriving models can be stored temporarily. The required maximum storage capacity of this buffer can be determined a-priori and for each node separately, as long as every node knows its number of neighbors. Notice that Step 3 assigns the value  $\ell = i$ , if it receives updates from all neighbors. In this case, the *i*-th node decides spontaneously to act as root. Moreover, Step 6 ensures that the node forwards an updated model to a spontaneously assigned parent as soon as it receives model updates from at least  $|\mathcal{N}_i| - 1$  neighbors. Last but not least, Step 5 ensures that the node performs updates and sends out new models to all children whenever it is switched to forward mode.

Because the protocol in Algorithm 2 does not specify which node is assigned as root, it is possible that different nodes spontaneously act as a root while the algorithm is running. Thus, there arises the question how many nodes can act as a root at the same time and, more generally, what the precise differences between Algorithm 2 and Algorithm 1 are. In order to give answers to these questions, we first analyze Algorithm 2 under the additional assumption that all nodes act synchronously, based on a global clock and equal sampling time. However, a completely asynchronous implementation of Algorithm 2 is possible, too, as discussed further below (see Section 4.2).



Fig. 2 Visualization of the first 8 iterations of Algorithm 2 for a tutorial network with 6 nodes.

Example 4.1 Figure 2 visualizes the first 8 iterations that are obtained when executing Algorithm 2 on all 6 nodes of our tutorial network. In the first iteration, because we initialize with  $\mathcal{R} = \emptyset$ , the condition  $|\mathcal{R}| = |\mathcal{N}_i| - 1$  in Step 6 of Algorithm 2 is satisfied for all nodes  $i \in \mathcal{L} = \{1, 3, 5, 6\}$ . Thus, these nodes

solve optimization problems in parallel (indicated by circles) and send approximate value functions to the central nodes (indicated by arrows). In the second iteration, the nodes with labels 2 and 4 are solving their optimization problems. Here, we assume that Node 4 happens to be quicker and sends its result to Node 2 (indicated by a solid arrow), but blocks the reverse line (indicated by a dashed arrow). Thus, in the third iteration, Node 2 acts spontaneously as root (see Step 5 of Algorithm 2) and sends its updated approximate value functions to all neighbors,  $\mathcal{N}_2 = \{1, 3, 4\}$ . Notice that the iterations continue with some nodes acting in forward mode while others are in backward mode. There is no preassigned root. For example, in the sixth iteration Node 4 happens to act has root, while in the third iteration the second node had taken this role.

#### 4.1 Synchronous multi-sweeps

If the protocols of Algorithm 2 are executed synchronously, the global behavior of this algorithm can be classified in dependence on the cardinality of the set of central nodes, which is defined as follows.

**Definition 4.1** Let  $\Gamma \subseteq \mathbb{N}_+$  denote the set of central nodes,

$$\label{eq:Gamma} \varGamma = \mathop{\mathrm{argmin}}_{i \in \mathcal{N}} \, \max_{j \in \mathcal{N}} \, d(i,j) \; .$$

We call  $|\Gamma|$  the parity of the (undirected) graph  $(\mathcal{N}, \mathcal{E})$ .

Notice that the set  $\Gamma$  can be interpreted as the set of nodes, which, if assigned as root, leads to a dynamic programming implementation with minimal runtime, since  $\Gamma$  is the set of all nodes *i*, whose depth is minimal. For example, for the tree structured network in Figure 1, we have  $\Gamma = \{2, 4\}$  with parity  $|\Gamma| = 2$ .

**Lemma 4.1** The parity of a tree structured graph is either even,  $|\Gamma| = 2$ , or odd,  $|\Gamma| = 1$ . Moreover, if it is even, then the two central nodes are neighbors.

*Proof* Let  $L = \max_{i,j} d(i,j)$  denote the length of a longest path in  $(\mathcal{N}, \mathcal{E})$  and

$$\mathbb{L} = \{(i,j) \mid d(i,j) = L\}$$

the set of pairs (i, j) with maximum distance. We distinguish two cases.

**Case 1:** L is even. In this case, there exists for every  $(i, j) \in \mathbb{L}$  an odd number of nodes on the shortest path from i and j, which implies that there is a (unique) central node k on this path with

$$d(i,k) = d(j,k) = \frac{L}{2} .$$

Let us assume that there is another pair  $(i', j') \in \mathbb{L}$ , whose central node k' is not equal to k. Then, we must have

$$d(i,k') + d(k',j) > d(i,j) = L, \qquad (22)$$

since k' is not on the shortest path from i to j. But this means that either d(i,k') > L/2 or d(j,k') > L/2, which is a contradiction, as  $d(i,k') \leq L/2$  and  $d(j,k') \leq L/2$  must hold due to the construction of the central node k'. Thus, in summary, all pairs  $(i,j) \in \mathbb{L}$  share the same central node k and we must have  $\Gamma = \{k\}$  by construction.

**Case 2:** L is odd. In this case, there exist for every  $(i, j) \in \mathbb{L}$  two central nodes  $k_1, k_2$  with  $d(k_1, k_2) = 1$  and such that

$$d(k_1, i) = (L-1)/2$$
 and  $d(k_2, j) = (L+1)/2$ .

Now, one can use a similar argument as in Case 1 to show that all  $(i, j) \in \mathbb{L}$  share the same central nodes, finding that  $\Gamma = \{k_1, k_2\}$ . Both cases together yield the statement of this lemma.

An immediate consequence of the above lemma is that the synchronized version of Algorithm 2 is actually equivalent to Algorithm 1 with one of the central nodes acting as root. For the case that the graph is odd,  $|\Gamma| = 1$ , the central root node is unique. Otherwise, for  $|\Gamma| = 2$ , one of the central nodes acts as root during every complete backward-forward sweep. However, in general, it cannot be predicted a-priori which of these central nodes act as root, as we have introduced the short random time delay in Step 6 of the protocol in Algorithm 2.

**Corollary 4.1** Algorithm 2 is equivalent to a variant of Algorithm 1, where one of the central nodes acts as root during one complete backward-forward sweep. In particular, Algorithm 2 converges under the same assumptions as Algorithm 1.

*Proof* Notice that the proof of Lemma 3.1 uses an induction argument under the assumption that the root node is fixed. In Algorithm 2 the root is assigned dynamically, but there is one unique root during each backward-forward sweep, which means that the induction argument from the Lemma 3.1 remains valid during every such complete sweep. But this means that the implications

$$W_{i,k}(p) \ge \Omega_{i,k}(p) \quad \Rightarrow \quad W_{i,k}(p) \ge V_{i,k}(p)$$

and 
$$W_{i,k}(p) \le \Omega_{i,k}(p) \Rightarrow W_{i,k}(p) \le V_{i,k}(p)$$
 (23)

also hold for Algorithm 2 observing that these relations are independent of which node is assigned as root. By using once more that Algorithm 2 has a unique root during each sweep, the local convergence rate estimate argument from the proof of Theorem 3.1 remains valid, too. Thus, the statement of this corollary is a direct consequence of Theorem 3.1.  $\Box$ 

### 4.2 Asynchronous multi-sweeps

An important observation of the previous section is that the root node is not assigned a-priori, but online while the algorithm is running. This allows us to run Algorithm 2 in asynchronous mode. In this case, every node executes the protocol from Algorithm 2 repeatedly, without coordinating the sampling time with other nodes. The analysis of this asynchronous variant is basically analogous to the synchronous case, but any node can act as the root node not only the central nodes. The convergence analysis is, however, unaffected; that is, running Algorithm 2 in asynchronous mode has no disadvantages in terms of its convergence properties.

### 5 Application to a radial AC power network

This section applies Algorithm 2 to a state estimation problem for a radial AC power network. We use MATPOWER v7.0 [45] to generate an IEEE 33-bus benchmark radial AC power network as shown in Figure 3.



Fig. 3 Topology of IEEE 33-bus benchmark.

For this network we have  $\mathcal{N} = \{1, \ldots, 33\}$  and  $\mathcal{E}$  is defined precisely as visualized in the figure. The following section briefly reviews the model equations for power networks [5] and the related least-squares state estimation problem [46], which is here used as a benchmark problem for decentralized optimization.

### 5.1 Model equations

Let  $v_i$  denote the voltage magnitude and  $\theta_i$  the phase shift at the *i*-th node of the power network. The active and reactive power at this node is given by the power flow equations [5],

$$p_i(v,\theta) = v_i \sum_{j \in \mathcal{N}_i} v_j (G_{ij} \cos(\theta_i - \theta_j) + B_{ij} \sin(\theta_i - \theta_j))$$
$$q_i(v,\theta) = v_i \sum_{j \in \mathcal{N}_i} v_j (G_{ij} \sin(\theta_i - \theta_j) + B_{ij} \cos(\theta_i - \theta_j)) .$$

The line conductance and susceptance matrices G and B are here assumed to be constant. The specific parameter values for these matrices can be found in [45]. In the following, we are additionally interested in the current  $I_{i,j}(v,\theta)$  at the transmission line  $(i,j) \in \mathcal{E}$ , which can be worked out explicitly in dependence on the voltages  $v_j$  and phase shifts  $\theta_j$ ,

$$I_{ij}(v,\theta) = \sqrt{\frac{P_{ij}(v,\theta)^2 + Q_{ij}(v,\theta)^2}{v_i^2}}$$

where  $P_{ij}(v,\theta)$  and  $Q_{ij}(v,\theta)$  denote the active and reactive power in the transmission line,

$$P_{ij}(v,\theta) = v_i^2 G_{ij} - v_i v_j \left[ G_{ij} \cos(\theta_i - \theta_j) + B_{ij} \sin(\theta_i - \theta_j) \right]$$
  
$$Q_{ij}(v,\theta) = v_i v_j \left[ G_{ij} \sin(\theta_i - \theta_j) + B_{ij} \cos(\theta_i - \theta_j) \right] - v_i^2 B_{ij}$$

for all  $(i, j) \in \mathcal{E}^+$ . The above model equation will next be used to formulate an AC power system state estimation problem.

### 5.2 Power system state estimation

Let  $\hat{v}_i, \hat{\theta}_i, \hat{p}_i$ , and  $\hat{q}_i$ , denote measurements of the voltages, phase shifts and active- and reactive powers at the nodes and let  $\hat{I}_{ij}$  denote measurements for the currents in the transmission lines  $(i, j) \in \mathcal{E}^+$ . Next, we consider the least-squares state estimation problem

$$\min_{v,\theta} \sum_{i\in\mathcal{N}} \left[ \left\| \begin{pmatrix} v_i - \hat{v}_i \\ \theta_i - \hat{\theta}_i \\ p_i(v,\theta) - \hat{p}_i \\ q_i(v,\theta) - \hat{q}_i \end{pmatrix} \right\|_{\Sigma_i}^2 + \sum_{j\in\mathcal{N}_i} \left\| I_{ij}(v,\theta) - \hat{I}_{ij} \right\|_{\Theta_i}^2 \right].$$

This problem is not yet in standard form, but it can easily be written in the form (7) by introducing auxiliary variables, where each node keeps copies of the voltage and phase shifts of its neighbors. In order to set up a realistic case study, the measurements are obtained by running a realistic scenario simulation in MATPOWER adding randomly generated process noise. Notice that the details of this problem formulation, including the details about how to introduce auxiliary variables and consensus constraints, as well as values for the weighting matrices  $\Sigma_i$  and  $\Theta_i$  can all be found in [46].

### 5.3 Numerical results

The diamonds in Figure 4 show the numerical results that were obtained by running Algorithm 2 for the above IEEE 33-bus benchmark case study using models  $W_{i,j}$  that are locally accurate up to order 2, as elaborated in Example 3.1. The black crosses show results for the same algorithm, but with Gauss-Newton Hessian approximations instead of exact Hessians. These results must be compared to the red circles, which show the iterates of the centralized solver



Fig. 4 Distance of current iterates to the optimal solution: Ipopt (red circles), Algorithm 2 with exact Hessians (blue diamonds), and Algorithm 2 with Gauss-Newton Hessians (black crosses).

**Ipopt** [47]. Here, it should be mentioned that, in this figure, one "iteration" of Algorithm 2 refers to a full backward-forward sweep. As predicted by Theorem 3.1, we can observe either linear or quadratic local convergence rates of Algorithm 2 depending on the accuracy of the communicated model functions.

## 6 Conclusions

This paper has presented a novel multi-sweep algorithm for asynchronous decentralized optimization over networks with tree graph structure. A first prototype for this method has been presented in the form of Algorithm 1, which has then been developed further arriving at a fully decentralized multi-sweep communication protocol for non-convex optimization, as presented in Algorithm 2. We have established conditions under which the proposed method has locally quadratic convergence rate, which have been summarized in Theorem 3.1. This theoretical result has been confirmed numerically by testing the method on a radial AC power network.

Acknowledgements YJ, HY, and BH acknowledge support by ShanghaiTech University, Grant-Nr. F-0203-14-012. DK and MD acknowledge support by BMWi via eco4wind (0324125B) and DyConPV (0324166B), and by DFG via Research Unit FOR 2401.

### A Proof of Theorem 1.1

Let us introduce the shorthands

$$z^{k+1} = \begin{pmatrix} z_1^{k+1} \\ z_2^{k+1} \end{pmatrix} = \begin{pmatrix} x^{k+1} \\ \kappa^{k+1} \end{pmatrix} \quad \text{and} \quad z^{\star} = \begin{pmatrix} x^{\star} \\ \kappa^{\star} \end{pmatrix}$$

to denote, respectively, the primal dual minimizer of (6) at the k-th iteration of the algorithm and the primal-dual minimizer of (5). Due to the regularity of  $x^*$  the LICQ condition must be satisfied in a neighborhood of  $x^\star,$  which implies that the first order necessary KKT conditions

$$R(x^k, z^{k+1}) = 0$$
 and  $R(x^*, z^*) = \widetilde{R}(z^*) = 0$  (24)

with shorthands

$$R(\xi,\zeta) = \nabla_z \left[ \Phi(\xi,\zeta_1) + \zeta_2^\top C(\zeta_1) \right] \quad \text{and} \quad \widetilde{R}(\zeta) = R(\zeta_1,\zeta) = \nabla_z \left[ F(\zeta_1) + \zeta_2^\top C(\zeta_1) \right]$$

are satisfied recalling that  $\Phi$  is a locally accurate approximation of F. Now, because the derivative of R with respect to its second argument,  $\nabla_z R(x, \cdot)$ , is uniformly Lipschitz continuous function in a neighborhood of  $z^*$ , the first equation in (24) yields

$$0 = R(x^{k}, z^{k+1}) = R(x^{k}, z^{k}) + \int_{0}^{1} \nabla_{z} R(x^{k}, z^{k} + s(z^{k+1} - z^{k}))(z^{k+1} - z^{k}) \,\mathrm{d}s \quad (25)$$

$$= \widetilde{R}(z^{k}) + M(z_{k})(z^{k+1} - z^{k}) + \mathbf{O}\left(\|z^{k+1} - z^{k}\|^{2}\right) , \qquad (26)$$

where we have set  $M(z^k) = \nabla_z R(x^k, z^k) = \nabla_z \widetilde{R}(z^k)$  and used that  $\widetilde{R}(z^k) = R(x^k, z^k)$ . Notice that the KKT matrix  $M(z_k)$  is invertible for all  $z^k$  in an open neighborhood of  $z^*$  as we assume that the LICQ and SOSC condition are satisfied at  $z^*$ . Consequently, because we have  $\widetilde{R}(z^k) = \mathbf{O}(||z^k - z^*||)$ , the above equation implies that

$$z^{k+1} = z^k - M(z^k)^{-1}\widetilde{R}(z^k) + \mathbf{O}(\|z^k - z^\star\|^2) .$$
(27)

From here on, the proof is very similar to the standard proof of quadratic convergence of Newton's method (see, e.g. [36, Thm. 3.5]); that is we use (27) to establish the inequality

$$||z^{k+1} - z^{\star}|| = \left\| z^{k} - z^{\star} - M(z^{k})^{-1} \widetilde{R}(z^{k}) \right\| + \mathbf{O}(||z^{k} - z^{\star}||^{2})$$

$$= \left\| z^{k} - z^{\star} - M(z^{k})^{-1} \left( \widetilde{R}(z^{k}) - \widetilde{R}(z^{\star}) \right) \right\| + \mathbf{O}(||z^{k} - z^{\star}||^{2})$$

$$= \left\| \left( I - M(z^{k})^{-1} \int_{0}^{1} \nabla_{z} \widetilde{R}(z^{k} + s(z^{k} - z^{\star})) \, \mathrm{d}s \right) (z^{k} - z^{\star}) \right\|$$

$$+ \mathbf{O}(||z^{k} - z^{\star}||^{2})$$

$$= \underbrace{\left\| I - M(z^{k})^{-1} \nabla_{z} \widetilde{R}(z^{k}) \right\|}_{=0} \left\| z^{k} - z^{\star} \| + \mathbf{O}(||z^{k} - z^{\star}||^{2}) \right\|. \tag{28}$$

Because the LICQ condition holds the iterates of the multiplier sequence  $\kappa^k$  is uniquely determined by the sequence  $x^k$  (since  $x^{k+1}$  depends only on  $x^k$ , but not on  $\kappa^k$ ), the above equation also implies that

$$||x^{k+1} - x^{\star}|| = \mathbf{O}(||x^k - x^{\star}||^2)$$
.

The latter equation corresponds to the statement of the theorem establishing local quadratic convergence.

#### References

- 1. Nedić, A., Olshevsky, A., Shi, W.: Decentralized consensus optimization and resource allocation. In: Large-Scale and Distributed Optimization, pp. 247–287. Springer (2018)
- Braun, P., Grüne, L., Kellett, C.M., Weller, S.R., Worthmann, K.: A distributed optimization algorithm for the predictive control of smart grids. IEEE Transactions on Automatic Control 61(12), 3898–3911 (2016)

- Hult, R., Zanon, M., Gros, S., Falcone, P.: Primal decomposition of the optimal coordination of vehicles at traffic intersections. In: 2016 IEEE 55th Conference on Decision and Control (CDC), pp. 2567–2573. IEEE (2016)
- 4. Jiang, Y., Zanon, M., Hult, R., Houska, B.: Distributed algorithm for optimal vehicle coordination at traffic intersections. IFAC-PapersOnLine **50**(1), 11,577–11,582 (2017)
- Engelmann, A., Jiang, Y., Mühlpfordt, T., Houska, B., Faulwasser, T.: Toward distributed OPF using ALADIN. IEEE Transactions on Power Systems 34(1), 584–594 (2018)
- Molzahn, D.K., Dörfler, F., Sandberg, H., Low, S.H., Chakrabarti, S., Baldick, R., Lavaei, J.: A survey of distributed optimization and control algorithms for electric power systems. IEEE Transactions on Smart Grid 8(6), 2941–2962 (2017)
- Boyd, S., Parikh, N., Chu, E., Peleato, B., Eckstein, J.: Distributed optimization and statistical learning via the alternating direction method of multipliers. Foundations and Trends<sup>®</sup> in Machine Learning 3(1), 1–122 (2011)
- Bertsekas, D., Tsitsiklis, J.: Parallel and distributed computation: numerical methods, vol. 23. Prentice hall Englewood Cliffs, NJ (1989)
- Shi, W., Ling, Q., Yuan, K., Wu, G., Yin, W.: On the linear convergence of the ADMM in decentralized consensus optimization. IEEE Transactions on Signal Processing 62(7), 1750–1761 (2014)
- Houska, B., Frasch, J., Diehl, M.: An augmented Lagrangian based algorithm for distributed nonconvex optimization. SIAM Journal on Optimization 26(2), 1101–1127 (2016)
- Gondzio, J., Grothey, A.: Exploiting structure in parallel implementation of interior point methods for optimization. Computational Management Science 6(2), 135–160 (2009)
- Pakazad, S., Hansson, A., Andersen, M.: Distributed primal-dual interior-point methods for solving tree-structured coupled problems using message passing. Optimization Methods & Software 32(3), 401–435 (2017)
- Zavala, V., Laird, C., Biegler, L.: Interior-point decomposition approaches for parallel solution of large-scale nonlinear parameter estimation problems. Chemical Engineering Science 63(19), 4834–4845 (2008)
- Hamdi, A.: Two-level primal-dual proximal decomposition technique to solve large scale optimization problems. Applied mathematics and computation 160(3), 921–938 (2005)
- Hamdi, A., Mishra, S.K.: Decomposition methods based on augmented Lagrangians: a survey. In: Topics in nonconvex optimization, pp. 175–203. Springer (2011)
- Bertsekas, D.: Convexification procedures and decomposition methods for nonconvex optimization problems. Journal of Optimization Theory and Applications 29(2), 169– 197 (1979)
- Hong, M., Luo, Z.Q., Razaviyayn, M.: Convergence analysis of alternating direction method of multipliers for a family of nonconvex problems. SIAM Journal on Optimization 26(1), 337–364 (2016)
- Khoshfetrat Pakazad, S., Hansson, A., Andersen, M.S., Nielsen, I.: Distributed primal– dual interior-point methods for solving tree-structured coupled convex problems using message-passing. Optimization Methods and Software **32**(3), 401–435 (2017)
- Bertsekas, D.: Constrained optimization and Lagrange multiplier methods. Academic press (2014)
- Terelius, H., Topcu, U., Murray, R.M.: Decentralized multi-agent optimization via dual decomposition. IFAC proceedings volumes 44(1), 11,245–11,251 (2011)
- Makhdoumi, A., Ozdaglar, A.: Convergence rate of distributed ADMM over networks. IEEE Transactions on Automatic Control 62(10), 5082–5095 (2017)
- Nesterov, Y.: Introductory lectures on convex optimization: A basic course, vol. 87. Springer Science & Business Media (2013)
- 23. Bellman, R.: Dynamic programming. Science 153(3731), 34-37 (1966)
- Rawlings, J., Mayne, D., Diehl, M.: Model predictive control: Theory and design, 2nd Edition. Madison, WI: Nob Hill Publishing (2017)
- Bernardini, D., Bemporad, A.: Stabilizing model predictive control of stochastic constrained linear systems. IEEE Transactions on Automatic Control 57(6), 1468–1480 (2011)

- Kouzoupis, D., Klintberg, E., Diehl, M., Gros, S.: A dual Newton strategy for scenario decomposition in robust multistage MPC. International Journal of Robust and Nonlinear Control 28(6), 2340–2355 (2018)
- Lucia, S., Andersson, J.A., Brandt, H., Diehl, M., Engell, S.: Handling uncertainty in economic nonlinear model predictive control: A comparative case study. Journal of Process Control 24(8), 1247–1259 (2014)
- Kekatos, V., Giannakis, G.B.: Distributed robust power system state estimation. IEEE Transactions on Power Systems 28(2), 1617–1626 (2012)
- Peng, Q., Low, S.: Distributed algorithm for optimal power flow on a radial network. In: 53rd IEEE Conference on decision and control, pp. 167–172. IEEE (2014)
- Bertsekas, D.P.: Dynamic Programming and Optimal Control, 3rd ed. Athena Scientific Belmont, MA (2007)
- Bertsekas, D.P.: Dynamic programming and suboptimal control: A survey from ADP to MPC. European Journal of Control 11(4-5), 310–334 (2005)
- Keshavarz, A., Boyd, S.: Quadratic approximate dynamic programming for input-affine systems. International Journal of Robust and Nonlinear Control 24(3), 432–449 (2014)
- Wang, Y., O'Donoghue, B., Boyd, S.: Approximate dynamic programming via iterated Bellman inequalities. International Journal of Robust and Nonlinear Control 25(10), 1472–1496 (2015)
- Kellerer, A., Steinke, F.: Scalable economic dispatch for smart distribution networks. IEEE Transactions on Power Systems 30, 1739–1746 (2014)
- Kellerer, A., Steinke, F.: An approximate min-sum algorithm for smart grid dispatch with continuous variables. IFAC-PapersOnLine 49, 307–312 (2016)
- 36. Nocedal, J., Wright, S.J.: Numerical optimization. Springer, 2nd edition (2006)
- 37. Robinson, S.: Strongly regular generalized equations. Mathematics of Operations Research  $\mathbf{5}(1),\,43\text{--}62$  (1980)
- Houska, B., Diehl, M.: Nonlinear robust optimization via sequential convex bilevel programming. Mathematical Programming, Series A 142, 539–577 (2013)
- Kouzoupis, D., Quirynen, R., Garcia, J., Erhard, M., Diehl, M.: A quadratically convergent primal decomposition algorithm with soft coupling for nonlinear parameter estimation. In: 2016 IEEE 55th Conference on Decision and Control (CDC), pp. 1086–1092 (2016)
- 40. Kouzoupis, D.: Structure-exploiting numerical methods for tree-sparse optimal control problems. Ph.D. thesis, University of Freiburg (2019)
- 41. Bertsekas, D.P.: Abstract dynamic programming. Athena Scientific Belmont, MA (2013)
- Grüne, L., Semmler, W.: Using dynamic programming with adaptive grid scheme to solve nonlinear dynamic models in economics. Computing in Economics and Finance 2002 (99) (2002)
- Luss, R.: Optimal control by dynamic programming using systematic reduction in grid size. Int. J. Control 51(5), 995–1013 (1990)
- Nesterov, Y., Polyak, B.T.: Cubic regularization of Newton method and its global performance. Mathematical Programming 108(1), 177–205 (2006)
- Zimmerman, R.D., Murillo-Sánchez, C.E., Thomas, R.J.: Matpower: Steady-state operations, planning, and analysis tools for power systems research and education. IEEE Transactions on power systems 26(1), 12–19 (2011)
- 46. Du, X., Engelmann, A., Jiang, Y., Faulwasser, T., Houska, B.: Distributed state estimation for AC power systems using Gauss-Newton ALADIN. In: In Proceedings of the 58th IEEE Conference on Decision and Control, pp. 1919–1924 (2019)
- Wächter, A., Biegler, L.T.: On the implementation of an interior-point filter line-search algorithm for large-scale nonlinear programming. Mathematical programming 106(1), 25–57 (2006)