

ALADIN- α —An open-source MATLAB toolbox for distributed non-convex optimization

Alexander Engelmann¹  | Yuning Jiang²  | Henrieke Benner¹ | Ruchuan Ou¹ | Boris Houska² | Timm Faulwasser¹ 

¹Institute for Automation and Applied Informatics, Karlsruhe Institute of Technology, Karlsruhe, Germany

²School of Information Science and Technology, ShanghaiTech University, Shanghai, China

Correspondence

Alexander Engelmann, Institute of Energy Systems, Energy Efficiency and Energy Economics, TU Dortmund University, Emil-Figge-Straße 70, 4227 Dortmund, Germany.
Email: alexander.engelmann@ieee.org

Present address

Alexander Engelmann, Ruchuan Ou and Timm Faulwasser are currently with the Institute of Energy Systems, Energy Efficiency and Energy Economics, TU Dortmund University, Dortmund, Germany

Funding information

Baden-Württemberg Stiftung

Abstract

This article introduces an open-source software for *distributed* and *decentralized* non-convex optimization named ALADIN- α . ALADIN- α is a MATLAB implementation of tailored variants of the Augmented Lagrangian Alternating Direction Inexact Newton (ALADIN) algorithm. Its user interface is convenient for rapid prototyping of non-convex distributed optimization algorithms. An improved version of the recently proposed bi-level variant of ALADIN is included enabling *decentralized* non-convex optimization with reduced information exchange. A collection of examples from different applications fields including chemical engineering, robotics, and power systems underpins the potential of ALADIN- α .

KEYWORDS

ADMM, ALADIN, decentralized optimization, distributed model predictive control, distributed optimization, nonconvex optimization, optimal power flow

1 | INTRODUCTION

Distributed *non-convex* optimization is of significant interest in various engineering domains. These domains range from electrical power systems,^{1–4} transportation problems,⁵ via machine learning,⁶ to distributed control,^{5,7–9} and distributed estimation.^{10–13} However, only few software toolboxes for distributed optimization are currently available. Moreover, these toolboxes are typically tailored to specific applications and often focus on *convex* problems. Examples comprise implementations of the Alternating Direction of Multipliers Method (ADMM) from Boyd et al.;^{6,*} an implementation of ADMM for consensus problems;[†] and a tailored implementation of ADMM for Optimal Power Flow (OPF) problems in Guo et al.^{14,‡} However, there is a lack of multi-purpose software tools for distributed optimization and, to the best of the authors' knowledge, there are no generic toolboxes for both distributed and decentralized non-convex optimization.

Notice that we distinguish *parallel* and *distributed* optimization. In *parallel* optimization, the main motivations are computational speed-up or computational tractability, while reducing the amount of communication and the amount of central coordination is typically of secondary importance (due to shared memory architectures). In *distributed* optimization, the main goal is to minimize central coordination and communication (distributed memory architectures).

This is an open access article under the terms of the Creative Commons Attribution License, which permits use, distribution and reproduction in any medium, provided the original work is properly cited.

© 2021 The Authors. *Optimal Control Applications and Methods* published by John Wiley & Sons Ltd.

Decentralized optimization additionally requires communication purely on a neighbor-to-neighbor basis. This is especially relevant in multi-agent settings, where individual entities cooperate to the end of optimization, control, or estimation—for example, in the context of cyber-physical systems, IoT, or embedded control. *Essentially decentralized* optimization softens the requirement of pure neighbor-to-neighbor communication by allowing the global summation of scalars.¹⁵

For *parallel* optimization efficient structure-exploiting tools exist. Classical tools include the GALAHAD software collection and in particular the LANCELOT algorithm, which is based on augmented Lagrangians and efficiently solves problems on shared-memory architectures.¹⁶ A closed-source parallel interior point software is OOPS.¹⁷ The open-source package qpDUNES is tailored towards the time-wise decomposition of Quadratic Programs (QPs) arising in Model Predictive Control (MPC).¹⁸ PIPS is a collection of algorithms solving structured linear programs, QPs, and general Nonlinear Programming Problems (NLPs) in parallel.^{19,20} The software HiOp is tailored towards structured and very large-scale NLPs with few nonlinear constraints. It is based on interior point methods.^{21,22} Moreover, combining parallel linear algebra routines (e.g., PARDISO)²³ with standard nonlinear programming solvers (e.g., IPOPT)²⁴ also leads to partially parallel algorithms.^{25,26} The tools mentioned above are implemented in low-level languages such as C or C++ leading to a high computational performance. On the other hand, their focus is mainly computational speedup via parallel computing rather than distributed and decentralized optimization in a multi-agent setting.

Classical distributed and decentralized optimization algorithms based on Lagrangian relaxation such as dual decomposition or ADMM are guaranteed to converge only for very specific non-convexities typically appearing in the objective function of the optimization problems commonly at a sublinear/linear rate.^{27–29} In many multi-agent applications, however, the non-convexities occur in the constraints. This implies that classical algorithms are not guaranteed to converge.^{2,8} One of the few algorithms exhibiting fast convergence guarantees in the non-convex case is the Augmented Lagrangian Alternating Direction Inexact Newton (ALADIN) algorithm.³⁰ Yet—up to now—a publicly available software implementation of ALADIN is missing.

The present article introduces an open-source MATLAB implementation of different ALADIN variants in the toolbox ALADIN- α . It is intended for rapid prototyping and aims at user-friendliness. The only user-provided information are objective and constraint functions—derivatives and numerical solvers are generated automatically using algorithmic differentiation routines and external state-of-the-art NLP solvers. A rich set of examples covering problems from robotics, power systems, sensor networks, and chemical engineering underpins the application potential of ALADIN- α . Besides the vanilla ALADIN algorithm, ALADIN- α covers recent extensions including:

- improved⁸ decentralization of bi-level ALADIN with essentially decentralized, respectively, decentralized variants of the Conjugate Gradient method (d-CG) and the Alternating Direction of Multipliers Method (d-ADMM) as inner algorithms;^{1,15}
- the nullspace ALADIN variant reducing communication and coordination;¹
- a parametric implementation enabling distributed MPC, and
- heuristics for Hessian regularization and parameter tuning for improving performance.

Moreover, we provide an implementation of ADMM based on the formulation of Houska et al.,³⁰ which uses the same interface as ALADIN. This way, comparisons between ALADIN and ADMM are fostered. Moreover, ALADIN- α can be executed in parallel mode via the MATLAB parallel computing toolbox. This often leads to a substantial speed-up, for example, in distributed estimation problems. A documentation and many application examples of ALADIN- α are available under <https://alexe15.github.io/ALADIN.m/>. We remark that ALADIN- α intends to be a *rapid prototyping* environment to enable testing of distributed and decentralized algorithms for non-convex optimization based on ALADIN. At this stage, computational speed or real-time feasibility are beyond the scope of the toolbox.

The remainder of the article is organized as follows: Section 2 recalls the main ideas of ALADIN and bi-level ALADIN. In Section 3 we comment on the code structure and data structures and present a simple tutorial example. Numerical examples from chemical engineering, power systems, and sensor networks illustrate how to use ALADIN- α in different application domains in Section 4. The Appendix provides implementation details.

2 | PRELIMINARIES

We start with a problem formulation amenable for distributed and decentralized optimization.

2.1 | Problem formulation

The ALADIN- α toolbox solves structured optimization problems of the form

$$\min_{x_1, \dots, x_{n_s}} \sum_{i \in S} f_i(x_i, p_i) \quad (1a)$$

$$\text{subject to } g_i(x_i, p_i) = 0 \quad |\kappa_i, \forall i \in S, \quad (1b)$$

$$h_i(x_i, p_i) \leq 0 \quad |\gamma_i, \forall i \in S, \quad (1c)$$

$$\underline{x}_i \leq x_i \leq \bar{x}_i \quad |\eta_i, \forall i \in S, \quad (1d)$$

$$\sum_{i \in S} A_i x_i = b \quad |\lambda, \quad (1e)$$

where $S = \{1, \dots, n_s\}$ is a set of subproblems, $f_i : \mathbb{R}^{n_{xi}} \times \mathbb{R}^{n_{pi}} \rightarrow \mathbb{R}$ are objective functions, $g_i : \mathbb{R}^{n_{xi}} \times \mathbb{R}^{n_{pi}} \rightarrow \mathbb{R}^{n_{gi}}$ and $h_i : \mathbb{R}^{n_{xi}} \times \mathbb{R}^{n_{pi}} \rightarrow \mathbb{R}^{n_{hi}}$ are constraint functions of the subproblems $i \in S$. Upper and lower bounds $\underline{x}_i, \bar{x}_i \in \mathbb{R}^{n_{xi}}$ are considered separately for numerical efficiency reasons. The matrices $A_i \in \mathbb{R}^{n_c \times n_{xi}}$ combined with $b \in \mathbb{R}^{n_c}$ model affine coupling constraints between the subproblems. The Lagrange multipliers κ assigned to the constraint g are denoted by $g(x) = 0 \quad |\kappa$. The partially separable formulation of (1) is generic: it contains several problem formulations as special cases such as consensus or sharing problems. Most NLPs can be reformulated in form of (1) by introducing auxiliary variables.⁶ We discuss a particular reformulation example in Section 4. Note that problem (1) allows for parametric problem data captured in $p_i \in \mathbb{R}^{n_{pi}}$. This can be useful in MPC or if one would like to solve the same problem for varying parameters.

2.2 | Standard and bi-level ALADIN

ALADIN solves convex and non-convex optimization problems (1) in a distributed fashion. A simplified flow chart of standard ALADIN is sketched in Figure 1. ALADIN combines ideas from ADMM and Sequential Quadratic Programming (SQP) combining distributed computation from ADMM with fast convergence properties and guarantees from SQP.³⁰ Similar to ADMM, ALADIN adopts a parallel step—that is, several NLPs are solved locally and in parallel to minimize local objective functions f_i together with augmentation terms

$$a_{z_i, \lambda}(x_i) = \lambda^\top A_i x_i + \|x_i - z_i\|_{\Sigma_i^k}^2.$$

These terms account for the coupling between the subproblems. Here, $\Sigma_i \in \mathbb{R}^{n_{xi} \times n_{xi}} > 0$ are scaling matrices and $z_i \in \mathbb{R}^{n_{xi}}$ encodes the influence of other subproblems. Moreover, local non-convex constraints

$$\mathcal{X}_i = \{x_i \in \mathbb{R}^{n_{xi}} \mid g_i(x_i) = 0, h_i(x_i) \leq 0\}$$

are considered in each subproblem $i \in S$. Since these subproblem-specific NLPs are similar in ALADIN and ADMM, both algorithms share the same computational complexity in the local step. Sensitivities such as the gradients of the local objective $\nabla f_i(x_i)$, Hessian approximations \tilde{B}_i and Jacobian matrices $(\nabla g_i, \nabla h_i)$ of the local constraints are evaluated locally. These sensitivities are combined in a sparse coordination QP adopted from SQP methods. Note that the coordination QP is equality-constrained and strongly convex (under certain regularity assumptions)—thus it can be reformulated as a system of linear equations. The primal and dual solution vectors of this coordination QP are broadcasted to the local subproblems and the next ALADIN iteration starts. The algorithm terminates once the norm of the violation of the constraint (1e) and the stepsize are both sufficiently small.

The main advantage of standard ALADIN over other existing approaches are convergence guarantees and fast local convergence.³⁰ On the other hand, the coordination QP makes ALADIN distributed but *not* decentralized. Furthermore, the coordination step in standard ALADIN is quite heavy and communication intense compared with other algorithms such as ADMM. Bi-level ALADIN overcomes these drawbacks by constructing a coordination QP of smaller dimension lowering communication.¹ Here, the sensitivities are “condensed” by computing the Schur-complement of the KKT

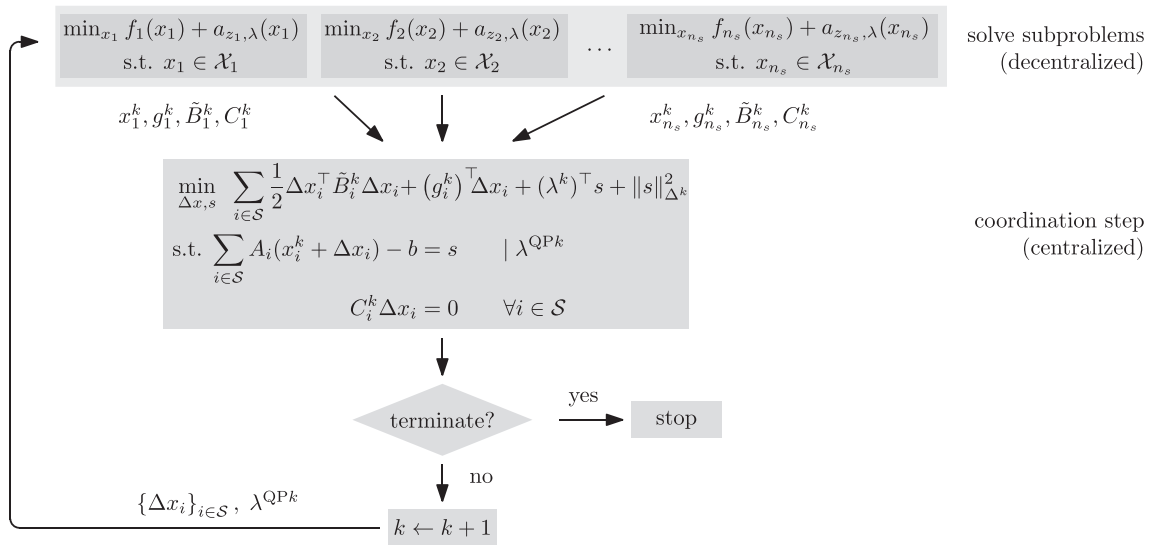


FIGURE 1 Simplified flow chart of standard ALADIN

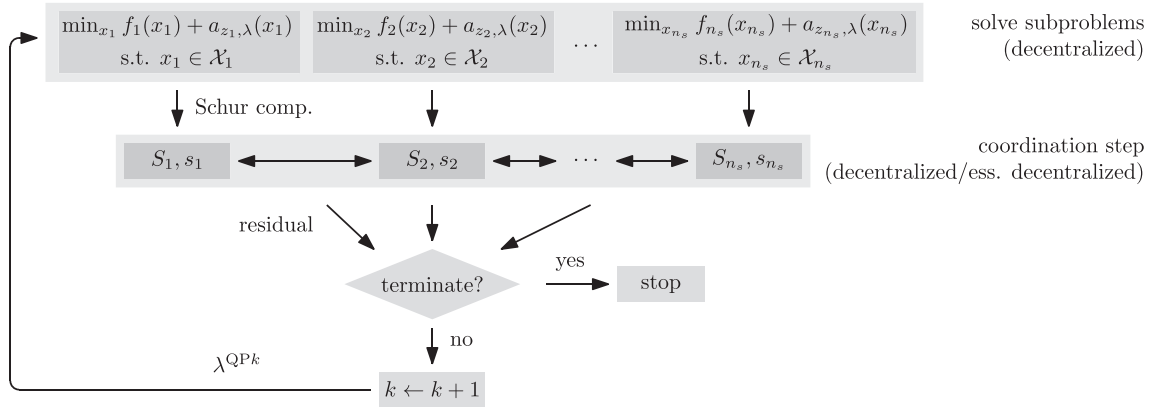


FIGURE 2 Simplified flow chart of bi-level ALADIN

systems leading to $\{S_i\}, \{s_i\}$, which are of dimension n_c —the number of the coupling variables. The number of coupling variables is typically much smaller than the total number of variables. These Schur-complements are combined in a lower-dimensional QP, which is solved in a decentralized fashion purely based on neighborhood communication leading to an overall decentralized algorithm. A simplified flow chart of bi-level ALADIN is shown in Figure 2. Observe that—in contrast to standard ALADIN (Figure 1)—bi-level ALADIN solves the coordination QP in a decentralized fashion based on decentralized inner algorithms. ALADIN- α comes with two of these inner algorithms: an essentially decentralized version of the conjugate gradient (d-CG) method and a decentralized version of ADMM (d-ADMM).¹⁵ The variables, which have to be exchanged in the solution process of the lower-dimensional QP depend on the particular decentralized algorithm at hand. Although these decentralized inner algorithms do not solve the coordination problem exactly, bi-level ALADIN is still guaranteed to converge locally under certain bounds on the numerical precision.¹ A detailed description of ALADIN is given in Appendix A.1.

3 | THE ALADIN- α TOOLBOX

This section presents the main contribution of this article: the ALADIN- α toolbox implementing different ALADIN variants. We comment on its code structure and data structures. Moreover, we illustrate the usage of ALADIN- α on a tutorial example.

3.1 | Code structure

In order to simplify algorithm development and testing we choose a procedural/functional programming style. All core features are implemented in MATLAB enabling easy rapid-prototyping. The overall structure of `run_ALADIN()`—the main function of ALADIN- α —is shown in Figure 3. First, a preprocessing step performs a consistency check of the input data and provides default options. The `createLocSolAndSens()` function initializes the local NLPs and sensitivities for all subproblems $i \in S$. We use CasADi³¹ for algorithmic differentiation and as an interface to many state-of-the-art NLP solvers such as IPOPT.²⁴ CasADi itself relies on pre-compiled code making function and derivative evaluation fast. A `reuse` option avoids the reconstruction of the CasADi problem setup, which enables the use of saved problem formulations. When the `reuse` mode is activated (e.g., when ALADIN- α is used within an MPC loop), `createLocSolAndSens()` is skipped, which results in a speed-up especially for large problems.

In the main loop `iterateAL()`, the function `parallelStep()` solves the local NLPs and evaluates the Hessian of the Lagrangian (or its approximation e.g., when BFGS is used), the gradient of the objective, and the Jacobian of the active constraints (sensitivities) at the NLP's solution. The set of active constraints is determined by primal active set detection described in Appendix A.1. Furthermore, a regularization procedure is executed if needed. Moreover, in case the nullspace method or bi-level ALADIN is used, the computation of a nullspace basis and the computation of the Schur-complement is performed locally shifting substantial computational burden from the centralized coordination step to `parallelStep()`. The function `updateParam()` computes dynamically changing ALADIN parameters for numerical stability and speedup.

The coordination QP is constructed in the function `createCoordQP()`. Different QP formulations are possible: here we use a variant considering slack variables from Houska et al. for numerical stability.³⁰ Different dense and sparse solvers for solving the coordination QP are available in `solveQP()`. Most of them are based on solving the first-order necessary conditions which is a system of linear equations. Available solvers are the MATLAB linear algebra routines `linsolve()`, `pinv()` and MA57.[†] Using sparse solvers can speed up the computation time substantially. Note that only MA57 supports sparse matrices. The solver can be specified by setting the `solveQP` option. In case of convergence problems from remote starting points it can help to reduce the primal-dual stepsize of the QP step by setting the `stepSize` in the options to a value smaller than 1. More advanced step-size selection rules are subject to ongoing and future work.

3.2 | Data structures

The data structure for defining problems in form of (1) is a struct called `sProb`. This data structure collects the objective functions $\{f_i\}_{i \in S}$ and constraint functions $\{g_i\}_{i \in S}$ and $\{h_i\}_{i \in S}$ in cells, which are contained in a nested struct called `locFuns` (Figure 4). Furthermore, `sProb` collects lower/upper bounds (1d) in cells `llbx` and `uubx`. The coupling matrices $\{A_i\}_{i \in S}$ are summarized in `AA`. One can provide NLP solvers and sensitivities optionally—in this case the problem construction in `createLocSolAndSens()` is

a) preprocessing	<code>checkInput()</code> , <code>setDefaultOpts()</code>
b) problem/sensitivity setup	<code>createLocSolAndSens()</code>
ALADIN main loop	<code>iterateAL()</code>
<div style="display: flex; align-items: center;"> <div style="writing-mode: vertical-rl; transform: rotate(180deg); margin-right: 5px;">in parallel</div> <div style="flex-grow: 1;"> <div style="display: flex; align-items: center;"> <div style="margin-right: 10px;">c) solve local NLPs</div> <div style="font-size: 3em; margin-right: 10px;">}</div> <div>parallelStep(), BFGS(),</div> </div> <div style="display: flex; align-items: center;"> <div style="margin-right: 10px;">d) evaluate sensitivities</div> <div style="font-size: 3em; margin-right: 10px;">}</div> <div>parallelStepInnerLoop(),</div> </div> <div style="display: flex; align-items: center;"> <div style="margin-right: 10px;">e) Hessian approx./regularization</div> <div style="font-size: 3em; margin-right: 10px;">}</div> <div>updateParam(), regularizeH()</div> </div> </div> </div>	
f) solve the coordination QP	<code>createCoordQP()</code> , <code>solveQP()</code> , <code>solveQPdec()</code>
g) compute primal/dual step	<code>computeALstep()</code>
h) postprocessing	<code>displaySummary()</code> , <code>displayTimers()</code>

FIGURE 3 Structure of `run_ALADIN()` in ALADIN- α

skipped leading to a speedup for large problems. This way, problem setups can be saved and reused. For a minimal working example of ALADIN- α , one only needs to specify `ffi` and `AA`. Optionally one can provide initial guesses `zz0` and initial Lagrange multipliers `lam0`. The second ingredient for ALADIN- α is an `opts` struct, which specifies the ALADIN variant and algorithm parameters. A full list of options with descriptions can be found in the code documentation.[#]

ALADIN- α returns a struct as output. This struct contains the cell `xxOpt` with local minimizers $\{x_i^*\}_{i \in S}$ and the optimal Lagrange multipliers λ^* of the consensus constraints (1e). Moreover the field `iter` contains information about the ALADIN iterates such as primal/dual iterates and `timers` collects timing information. Note that `run_ALADIN()` and `run_ADMM()` have the same function signature in terms of `sProb`—only the options differ.

3.3 | Further features

We describe selected features of ALADIN- α —a full description of all features can be found under^{||}.

Hessian approximations Instead of exact Hessians, approximations such as the Broyden–Fletcher–Goldfarb–Shanno (BFGS) update can be used either to reduce communication and/or to reduce computational complexity in sensitivity computation. The BFGS Hessian is activated by setting the `Hess` option either to `BFGS` for standard BFGS or to `DBFGS` for damped BFGS. For details on BFGS we refer to the book of Nocedal and Wright.³²

Parametric NLP setup A parametric problem setup, where the objective functions f_i and the equality/inequality constraints g_i/h_i depend on parameters p_i is possible. This feature is useful in combination with the `reuse` option which returns the internally constructed CasADi solvers and derivatives. If one provides a previously constructed NLP as input argument when calling `run_ALADIN()`, the problem construction is skipped, which can lead to a substantial speedup. In an MPC setting, for example, the parameter p_i models the changing initial condition in the MPC loop. Moreover, parametric problem data might be useful for large-scale problems where one would like to solve an optimization problem for a wide range of parameters. This feature is activated by adding a parameter cell `p` to `sProb` and defining the objective/constraints in terms of two inputs, x_i and p_i . An example illustrating how to use these features for distributed predictive control of two mobile robots is given in the code repository.^{**}

Parallelization ALADIN- α also supports parallel computing on multiple processors via the MATLAB parallel computing toolbox. Here, we exploit the fact that the local NLPs are independent from each other, that is, they can be solved in parallel. An example for distributed nonlinear estimation with mobile sensor networks can be found in Subsection 4.3. Parallel computing can be activated by setting the `parfor` option to `true`.

Application examples We provide numerical examples highlighting applicability of ALADIN- α to a wide range of problems. The code for all these examples is available in the `examples\` folder of ALADIN- α . Furthermore, we provide descriptions of these examples in the documentation online.^{††} Beyond the examples of this section, we consider

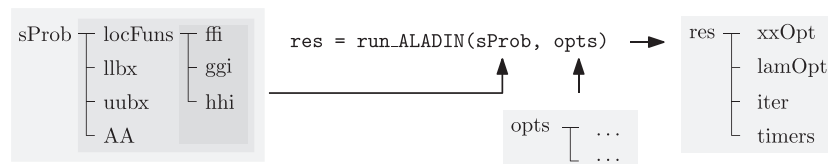


FIGURE 4 The `sProb` data structure for defining problems in form of (1)

TABLE 1 Application examples coming with ALADIN- α

Example	Field	examples/...	Docs
Chemical reactors	Chemical engineering/control	chemical_reactor	
Mobile robots	Robotics/control	robots	https://alexe15.github.io/ALADIN.m/robotEx/
Optimal power flow	Power systems	optimal_power_flow	https://alexe15.github.io/ALADIN.m/redComm/
Sensor network	Estimation	sensor_network	https://alexe15.github.io/ALADIN.m/ParallelExample/

distributed optimal control and the application of ALADIN- α to test problems from the Hock–Schittkowski test collection in the online repository.^{1,33,34} A list of all examples is given in Table 1.

3.4 | A tutorial example

Consider the non-convex NLP

$$\min_{x_1, x_2 \in \mathbb{R}} f(x) = 2(x_1 - 1)^2 + (x_2 - 2)^2 \quad (2a)$$

$$\text{subject to} \quad -1 \leq x_1 \cdot x_2 \leq 1.5. \quad (2b)$$

In order to apply ALADIN- α , we reformulate problem (2) in form of (1). We introduce auxiliary variables y_1, y_2 with $y_1 \in \mathbb{R}$ and $y_2 = (y_{21} \ y_{22})^\top \in \mathbb{R}^2$. We couple these variables again by introducing a consensus constraint $\sum_i A_i y_i = 0$ with $A_1 = 1$ and $A_2 = (-1 \ 0)$. Furthermore, we reformulate the objective function f by local objective functions $f_1(y_1) := 2(y_1 - 1)^2$ and $f_2(y_2) = (y_{22} - 2)^2$ with $f = f_1 + f_2$. Moreover, reformulate the global inequality constraint (2b) by a local two dimensional constraint $h_2 = (h_{21} \ h_{22})^\top$ with $h_{21}(y_2) = -1 - y_{21} y_{22}$ and $h_{22}(y_2) = -1.5 + y_{21} y_{22}$. Combining these reformulations yields

$$\min_{y_1 \in \mathbb{R}, y_2 \in \mathbb{R}^2} 2(y_1 - 1)^2 + (y_{22} - 2)^2 \quad (3a)$$

$$\text{subject to} \quad -1 - y_{21} y_{22} \leq 0, \quad -1.5 + y_{21} y_{22} \leq 0, \quad (3b)$$

$$y_1 + (-1 \ 0) y_2 = 0, \quad (3c)$$

which is in form of (1). Note that the solutions to (2) and (3) coincide but (3) is of higher dimension. This reformulation reveals a general strategy for reformulating problems in form of (1): if there is nonlinear coupling in the objective functions or the constraints, introduce auxiliary variables and require them to coincide by an additional consensus constraint in form of (1e).

Solution with ALADIN- α Next, we transcribe (3) in the struct `sProb` as illustrated in Subsection 3.2. To highlight different possibilities of problem setup, we construct the problem in three different ways: a) via the MATLAB symbolic toolbox, b) via the CasADi symbolic framework, and, c) directly via function handles, compare Figure 5.

After defining objective and constraint functions, all function handles and the coupling matrices A_i are collected in `sProb`. We call `run_ALADIN()` with an empty options struct leading to computation with default parameters. The code and the resulting ALADIN- α report after running `run_ALADIN()` are shown in Figure 6. In the ALADIN- α report, the reason for termination and timing information is displayed. Figure 7 shows the output of ALADIN- α while it is running. The figures show (in this order) the consensus violation $\|Ax - b\|_\infty$, the local step sizes $\|x^k - z^k\|_\infty$, the step size in the coordination step $\|\Delta x^k\|_\infty$, and the changes in the active set. Note that online plotting may consume a substantial amount of time—hence it is advisable to deactivate online plotting if there is not required for diagnostic reasons.

4 | NUMERICAL CASE STUDIES

We present three case studies to shed light on the differences of the implemented algorithms. We consider an optimal control problem for a chemical reactor, an OPF problem, and a sensor localization example.

4.1 | Distributed optimal control of a chemical process system

We consider a discrete-time Optimal Control Problem (OCP) for a chemical process system. This OCP can serve as a basis for distributed MPC.³⁵⁻³⁷ The process consists of two Continuous Stirred-Tank Reactors (CSTRs) and a flash separator shown in Figure 8.^{38,39} The goal is to steer the system to the optimal setpoint

<pre> % define symbolic variables y1 = sym('y1',[1,1],'real'); y2 = sym('y2',[2,1],'real'); % define symbolic objectives f1s = 2*(y1-1)^2; f2s = (y2(2)-2)^2; % define symbolic ineq. constraints h2s = [-1-y2(1)*y2(2); ... -1.5+y2(1)*y2(2)]; % convert symbolic variables to MATLAB fuctions f1 = matlabFunction(f1s,'Vars',{y1}); f2 = matlabFunction(f2s,'Vars',{y2}); h1 = @(y1)[]; h2 = matlabFunction(h2s,'Vars',{y2}); </pre>	<pre> % define symbolic variables y_1 = SX.sym('y_1', 1); y_2 = SX.sym('y_2', 2); % define symbolic objectives f1s = 2 * (y_1 - 1)^2; f2s = (y_2(2) - 2)^2; % define symbolic ineq. constraints h1s = []; h2s = [-1 - y_2(1)*y_2(2); ... -1.5 + y_2(1)*y_2(2)]; % convert symbolic variables to MATLAB fuctions f1 = Function('f1', {y_1}, {f1s}); f2 = Function('f2', {y_2}, {f2s}); h1 = Function('h1', {y_1}, {h1s}); h2 = Function('h2', {y_2}, {h2s}); </pre>	<pre> % define objectives f1 = @(y1) 2 * (y1 - 1)^2; f2 = @(y2) (y2(2) - 2)^2; % define inequality constraints h1 = @(y1) [] h2 = @(y2) [-1 - y2(1) * y2(2);... -1.5 + y2(1) * y2(2)]; </pre>
---	---	--

FIGURE 5 Problem setup via MATLAB symbolic (left), CasADi (middle), and function handles (right)

<pre> % define coupling matrices A1 = 1; A2 = [-1, 0]; % collect problem data in sProb struct sProb.locFuns.ffi = {f1, f2}; sProb.locFuns.hhi = {h1, h2}; % handing over of coupling matrices to problem sProb.AA = {A1, A2}; % start solver with default options sol = run_ALADINnew(sProb); </pre>	<pre> ===== == This is ALADIN-alpha v0.1 == ===== QP solver: MA57 Local solver: ipopt Inner algorithm: none No termination criterion was specified. Consensus violation: 6.6531e-12 Maximum number of iterations reached. ----- ALADIN-alpha timing ----- t[s] %tot %iter Tot time.....: 3.92 Prob setup....: 0.19 4.8 Iter time.....: 3.72 95 ----- NLP time.....: 1.1 29.7 QP time.....: 0.11 2.8 Reg time.....: 0.02 0.6 Plot time.....: 2.27 60.8 ===== </pre>
---	---

FIGURE 6 Collection of variables (left) and output of ALADIN- α (right)

$$u_s^T = (0 \ 0 \ 0) \text{ and } x_s^T = (369.53 \ 3.31 \ 0.17 \ 0.04 \ 435.25 \ 2.75 \ 0.45 \ 0.11 \ 435.25 \ 2.88 \ 0.50 \ 0.12)$$

from $x(0)^T = (360.69 \ 3.19 \ 0.15 \ 0.03 \ 430.91 \ 2.76 \ 0.34 \ 0.08 \ 430.42 \ 2.79 \ 0.38 \ 0.08)$. After applying a fourth-order Runge–Kutta scheme for discretization, the dynamics of all CSTRs and the flash separator are given by

$$x_i^{k+1} = q_i(x_i^k, u_i^k, z_i^k) \text{ for } i \in S,$$

where $q_i : \mathbb{R}^{n_{xi}} \times \mathbb{R}^{n_{ui}} \times \mathbb{R}^{n_{zi}} \rightarrow \mathbb{R}^{n_{xi}}$ are the dynamics of the i th vessel with $S := \{1, 2, 3\}$ being the set of vessels. Here, $x_i^T = (x_{Ai}, x_{Bi}, x_{Ci}, T_i)$ are the states, x_{Ai}, x_{Bi}, x_{Ci} are the concentrations of the reactants A, B and C , and T is the temperature.

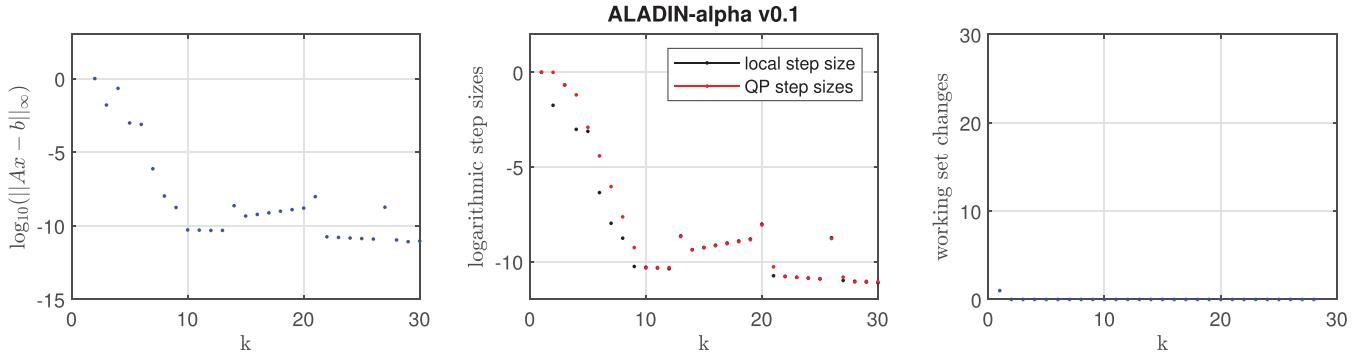


FIGURE 7 ALADIN- α iteration plot for tutorial problem (3)

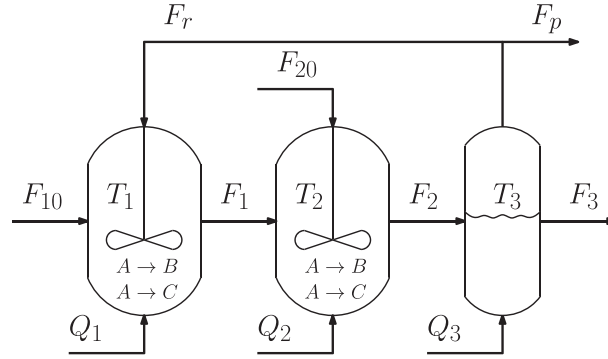


FIGURE 8 Reactor-separator process

The input $u_i = Q_i$ denotes the heat-influx of the i th vessel and $z_i := (x_j)_{j \in N(i)}$ are copied states of all neighbors $N(i) \subseteq S$. Note that the feed-stream flow rates F_{10}, F_{20}, F_3, F_R , and F_p are fixed and given. A detailed description of the system dynamics is given in Christofides et al.³⁹ With the above, we formulate a discrete-time optimal control problem

$$\min_{\substack{(x_i^k, z_i^k, u_i^k)_{i \in S} \\ k \in \mathbb{I}_{[1, T]}}} \sum_{i \in S} \sum_{k \in \mathbb{I}_{[1, T]}} \frac{1}{2} (x_i^k - x_{is}^k)^\top Q_i (x_i^k - x_{is}^k) + \frac{1}{2} (u_i^k - u_{is}^k)^\top R_i (u_i^k - u_{is}^k) \quad (4a)$$

$$\text{subject to } x_i^{k+1} - q_i(x_i^k, u_i^k, z_i^k) = 0, \quad x_i^0 = x_i(0) \text{ for all } k \in \mathbb{I}_{[1, T]} \text{ and for all } i \in S, \quad (4b)$$

$$\underline{u}_i \leq u_i^k \leq \bar{u}_i, \quad \underline{x}_i \leq x_i^k, \quad \text{for all } k \in \mathbb{I}_{[1, T]} \text{ and for all } i \in S, \quad (4c)$$

$$\sum_{i \in S} A_i (x_i^{k\top} z_i^{k\top} u_i^{k\top})^\top = 0 \quad \text{for all } k \in \mathbb{I}_{[1, T]}, \quad (4d)$$

with lower/upper bounds on the inputs $\bar{u} = -\underline{u} = (5 \cdot 10^4 \ 1.5 \cdot 10^5 \ 2 \cdot 10^5)^\top$, and lower bounds on the states $\underline{x}_i^k = 0$ for all times $k \in \mathbb{I}_{[1, T]}$ and all vessels $i \in S$. The weighting matrices are $Q_i = \text{diag}(20 \ 10^3 \ 10^3 \ 10^3)$ and $R_i = 10^{-10}$. The matrices A_i are constructed to model the constraint $z_i := (x_j)_{j \in N(i)}$. The sampling time is $\Delta h = 0.01h$ and the horizon is $T = 0.1$ h. By defining $\tilde{x}_i^\top := (x_i^{k\top} z_i^{k\top} u_i^{k\top})_{k \in \mathbb{I}_{[1, T]}}$, $f_i(\tilde{x}_i) := \sum_{k \in \mathbb{I}_{[1, T]}} \frac{1}{2} (x_i^k - x_{is}^k)^\top Q_i (x_i^k - x_{is}^k) + \sum_{k \in \mathbb{I}_{[1, T-1]}} \frac{1}{2} (u_i^k - u_{is}^k)^\top R_i (u_i^k - u_{is}^k)$, $g_i(\tilde{x}_i) := (x_i^{k+1} - q_i(x_i^k, u_i^k, z_i^k))_{k \in \mathbb{I}_{[1, T-1]}}$, and $h_i(\tilde{x}_i) := ((\underline{u}_i - u_i^k \ u_i^k - \bar{u}_i \ \underline{x}_i - x_i^k)^\top)_{k \in \mathbb{I}_{[1, T]}}$ the OCP (4) is in form of (1), where \tilde{x}_i corresponds to x_i in (1).

Numerical results Figure 10 shows the convergence behavior of standard ALADIN, of bi-level ALADIN with d-CG as inner algorithm, of bi-level ALADIN with d-ADMM as inner algorithm, and of ADMM over the iteration index k . Specifically, we depict the distance to a minimizer $\|x^k - x^*\|_\infty$, the consensus violation $\|Ax^k - b\|_\infty$, and the optimality gap $|f(x^k) - f(x^*)|$. Note that for the considered problem, ADMM is not guaranteed to converge because of the nonlinear dynamics. However, since ADMM is nevertheless used in many works, we use

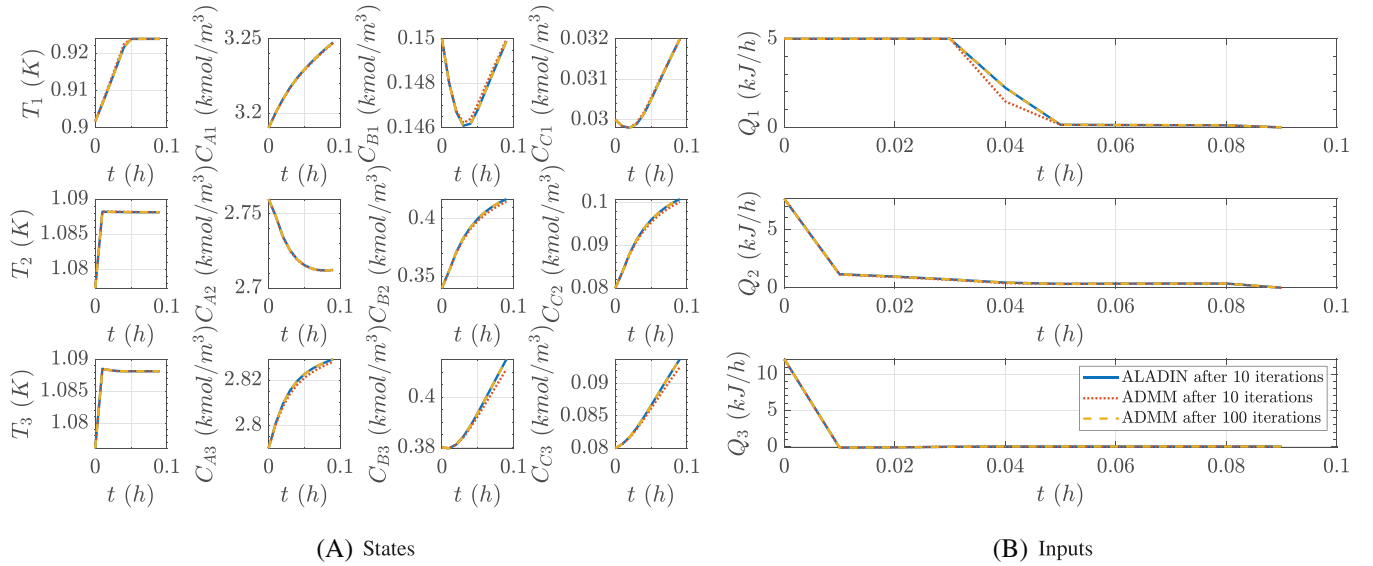


FIGURE 9 Optimal state/input trajectories computed by ALADIN and ADMM

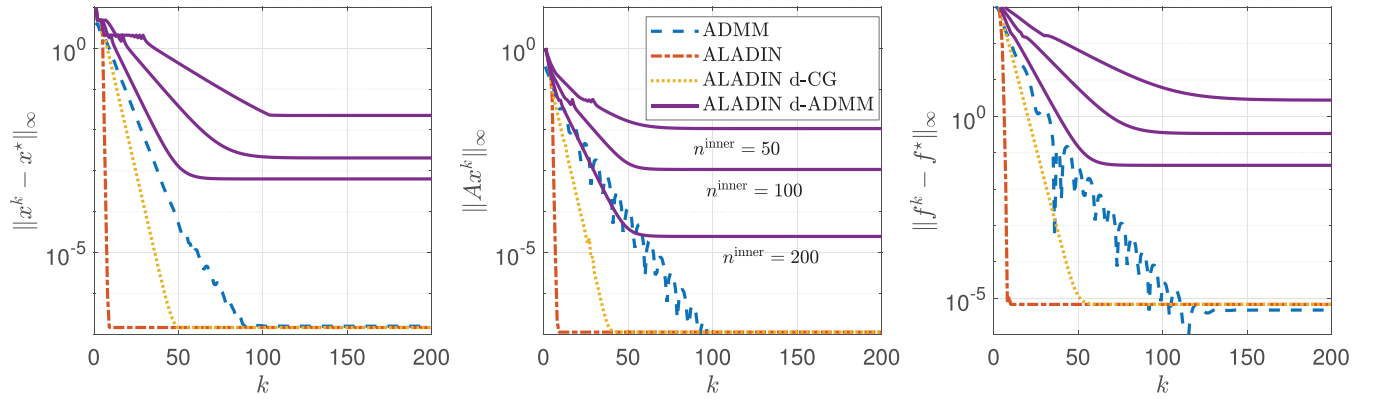


FIGURE 10 Numerical performance of ALADIN, bi-level ALADIN, and ADMM for OCP (4)

it as a baseline for comparison.^{40–42} Bi-level ALADIN with d-ADMM is executed with inner d-ADMM iterations $n^{\text{inner}} \in \{50, 100, 200\}$ and bi-level ALADIN with d-CG is executed with 200 inner d-CG iterations. One can see that ADMM converges fast and there seems to be no benefit when using bi-level ALADIN with ADMM as an inner algorithm. Basic ALADIN and ALADIN with conjugate gradients converges faster, but one has to solve an expensive coordination step in case of basic ALADIN or to perform many inner iterations in case of bi-level ALADIN with d-CG.

Figure 9 shows the resulting open-loop input and state trajectories for OCP (4) for ALADIN and ADMM after 20 iterations, and for ADMM after 100 iterations. At first-glance, all trajectories look quite similar. However, small differences in the input trajectories can be observed. Close inspection of Figure 10 shows that in logarithmic scale the differences can be large. For example the consensus gap $\|Ax - b\|_\infty$ is in an order of 10^{-1} after 20 iterations, which means that the physical values at the interconnection points have a maximum mismatch of 10^{-1} .

4.2 | Distributed optimal power flow

Next, we consider an OPF problem, which is one of the most important optimization problems in power systems.⁴³ Distributed optimization is particularly important here due to large problem sizes and due to the necessity of a reduced information exchange between subsystems.

Numerical results Figure 12 shows the performance of all distributed and decentralized optimization algorithms coming with ALADIN- α . Bi-level ALADIN with d-ADMM is executed with inner d-ADMM iterations $n^{\text{inner}} \in \{50, 100, 200\}$ and bi-level ALADIN with d-CG is executed with 70 inner iterations. One can see that in contrast to the chemical reactor from the previous subsection ADMM converges quite slowly and requires about 1500 iterations to converge to an acceptable level of accuracy. This underlines that the performance of ADMM is problem dependent—especially in a setting with non-convex constraints. Basic ALADIN and bi-level ALADIN with d-CG on the

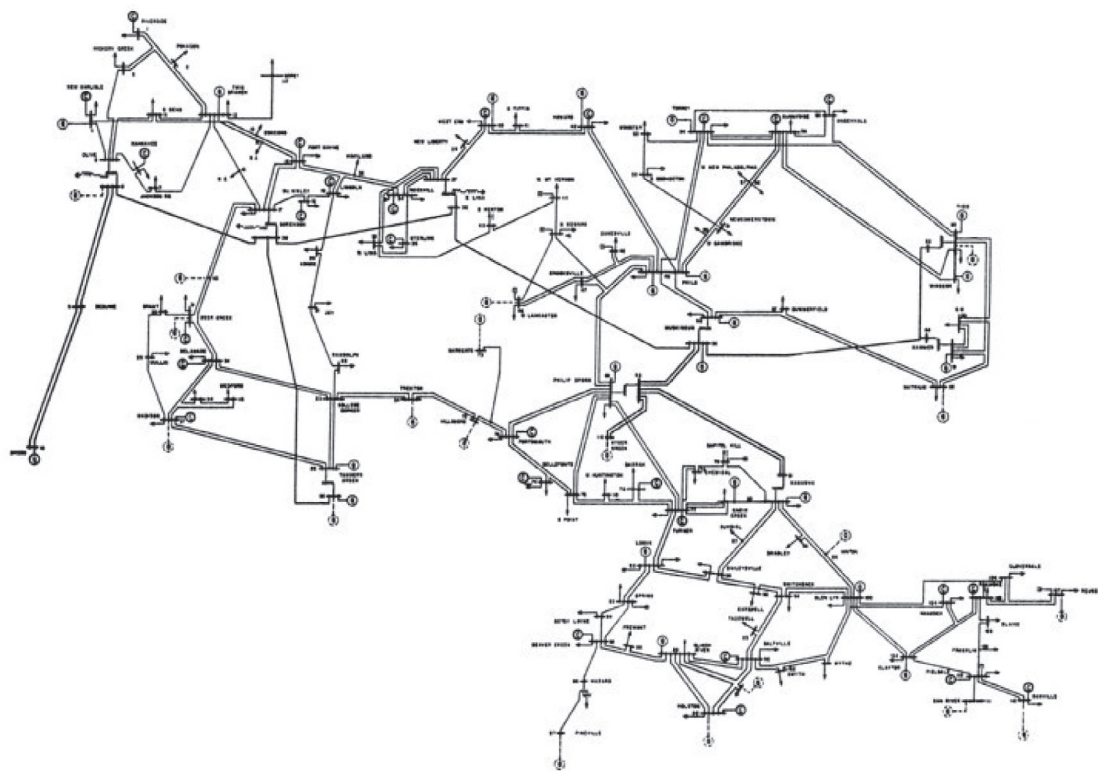


FIGURE 12 Numerical performance of ALADIN, bi-level ALADIN and ADMM for the 118-bus OPF problem

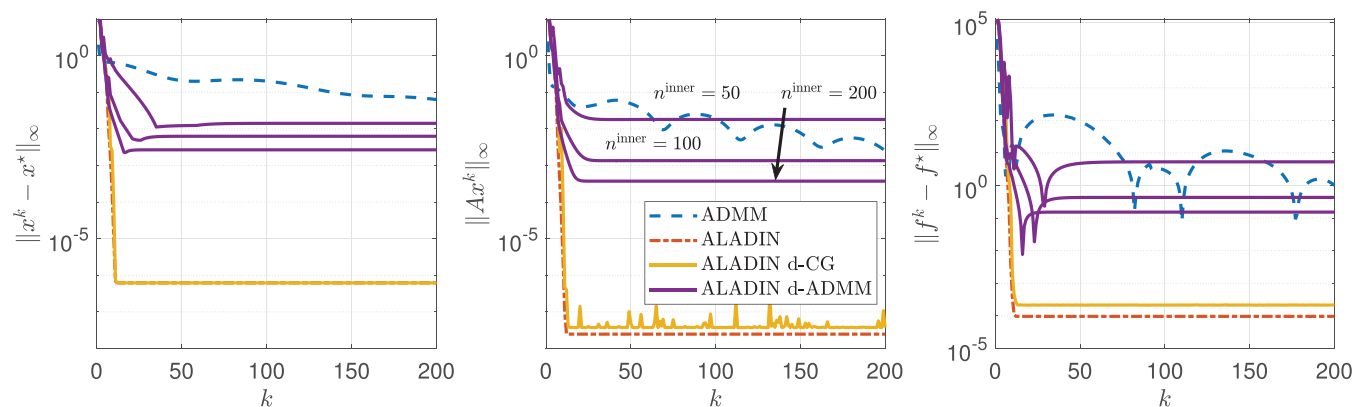


TABLE 2 Timings for different algorithms converging to $\|Ax^k\|_\infty < 10^{-3}$

Basic ALADIN	Bi-level ALADIN		ADMM
	d-CG (70)	d-ADMM (200)	
2,5 s	2,8 s	4,5 s	16,2 s

other hand converge rapidly and to a high accuracy. For bi-level ALADIN with d-ADMM, the achievable accuracy depends on the number of inner d-ADMM iterations.

Table 2 shows timing information for all algorithms converging to $\|Ax^k\|_\infty < 10^{-3}$. We use a computer with an Intel Core i7-8550U processor with 4 cores, 16 GiB of memory, and MATLAB R2020a running Arch Linux with parallel computing disabled. The initialization phase of the sensitivities is not considered. One can see that there is not much difference between the ALADIN variants since most of the time is spent in solving the local NLPs and this step is the same. ADMM is about five times slower since it requires many more iterations and thus many more NLP solutions are computed.

4.3 | Parallel computing for sensor localization

To illustrate parallel computing capabilities of ALADIN- α , we consider a sensor localization problem.³⁰ Let N be the number of sensors, let $\chi_i = (x_i, y_i)^T \in \mathbb{R}^2$ be the unknown position of the i th sensor, let η_i be its estimated position, let ξ_i be the position of sensor $(i + 1)$ estimated by sensor i , and let $\bar{\eta}_i$ be the estimated distance between sensor i and its neighbors. The measurement error is given by $\eta_i - \chi_i$ and is assumed to be Gaussian with covariance $\sigma_i^2 I_{2 \times 2}$ with $\sigma = 1.5$ and zero mean. We define local decision variables $x_i = (\chi_i^T, \xi_i^T)^T \in \mathbb{R}^4$ and obtain the estimation problem

$$\begin{aligned} \min_x \quad & \sum_{i \in S} f_i(x_i) \\ \text{s.t.} \quad & h_i(x_i) \leq 0 \quad \forall i \in \{1, \dots, N\}, \\ & \xi_i = \chi_{i+1} \quad \forall i \in \{1, \dots, N\}, \end{aligned}$$

with

$$f_i(x_i) = \frac{1}{4\sigma_i^2} \|\chi_i - \eta_i\|_2^2 + \frac{1}{4\sigma_{i+1}^2} \|\xi_i - \eta_{i+1}\|_2^2 + \frac{1}{2\sigma_{i+1}^2} (\|\chi_i - \xi_i\|_2^2 - \bar{\eta}_i)^2,$$

and

$$h_i(x_i) = (\|\chi_i - \xi_i\|_2 - \bar{\eta}_i)^2 - \bar{\sigma}_i^2,$$

which is in form of problem (1).

Numerical results We investigate how the runtime of ALADIN- α can be improved via parallel computing. The timings are shown in Figure 13 for different numbers of sensors S . One can see that for a very small number of sensors (less than 10), parallel computing is not very useful as the introduced additional overhead outweighs the benefits of parallelization. However, for larger number of sensors a speedup factor of more than two can be achieved when running on four processors.

5 | SUMMARY AND FUTURE WORK

This article has introduced one of the first open source toolboxes for distributed non-convex optimization: ALADIN- α . It is based on the ALADIN algorithm and implements various extensions mostly aiming at reducing communication and

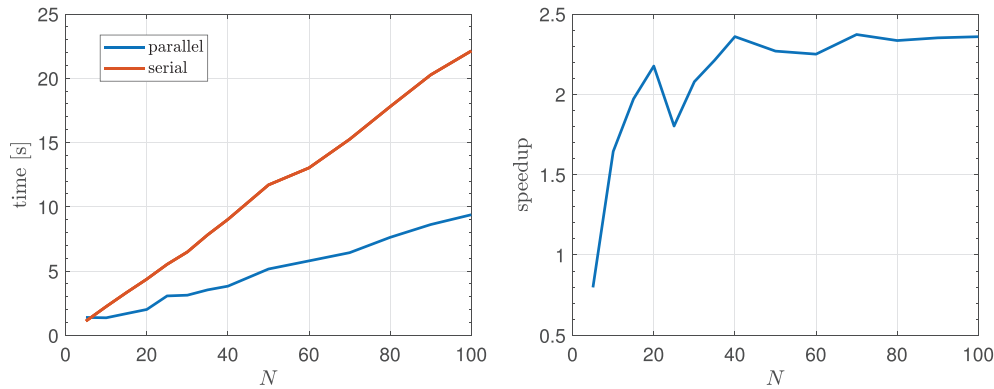


FIGURE 13 Computation time for N sensors computed on four cores

coordination overhead. Moreover, ALADIN- α comes with a rich set of examples from different engineering fields reaching from power systems over non-linear control to mobile sensor networks.

Although ALADIN- α performs well for many small to medium-sized problems, we aim at further improving numerical stability in future work by developing more advanced internal auto-tuning routines. Furthermore, developing distributed globalization strategies for enlarging the set of possible initializations seems important and promising. Code generation for distributed optimization on embedded devices is another interesting research direction. A possible alternative to ALADIN-based schemes for distributed non-convex optimization seem essentially decentralized interior point methods.^{15,45}

ACKNOWLEDGMENT

We would like to thank Tillmann Mühlfordt for very helpful discussions and suggestions, and Veit Hagenmeyer for supporting the development of ALADIN- α . Moreover, Timm Faulwasser acknowledges financial support by the Elite Program for Postdocs of the Baden-Württemberg Stiftung.

DATA AVAILABILITY STATEMENT

The data that support the findings of this study are openly available in the repository <https://github.com/alexe15/ALADIN.m/tree/b156d42c64d837865644e2f212d9ae5ead2143d4>.

ENDNOTE

^{*}<https://web.stanford.edu/~boyd/papers/admm/>

[†]<http://users.isr.ist.utl.pt/~jmota/DADMM/>

[‡]<https://github.com/guojunyao419/OPF-ADMM>

[§]The version of bi-level ALADIN given here is improved in the sense that we use improved versions of d-CG and d-ADMM from Engelmann et al.¹⁵ In contrast to a previous version,¹ these two algorithms rely on a unified sparsity framework and do not require a precomputation phase lowering communication demand.

[¶]MA57 is interfaced indirectly—we employ the MATLAB LDL factorization, which is based on MA57.

[#]<https://alexe15.github.io/ALADIN.m/options/>

^{||}<https://alexe15.github.io/ALADIN.m/options/>

^{**}<https://alexe15.github.io/ALADIN.m/robotEx/>

^{††}<https://alexe15.github.io/ALADIN.m/>


^{‡‡}We use scaled 2-norms $\|x\|_{\Sigma} := \sqrt{x^T \Sigma x}$ for $\Sigma > 0$ here.

^{§§}Some numerical solvers for (A1) can for example treat box constraints in an efficient way by using projection methods.

^{¶¶}Note that in contrast to Houska et al.,³⁰ we omit the term $\rho/2$ in front of the penalization term in (A1) avoiding redundancy. The setting from Houska et al.³⁰ can be recovered by choosing $\Sigma_i^k = \rho^k/2 I$.

^{**}If one would use (A3) regardless, the coordination step (A2) would not necessarily produce descent directions destroying the local convergence properties of ALADIN. In case of zero eigenvalues, B_i^k is singular and the coordination step can not be solved by a standard solver for linear systems of equations.

ORCID

Alexander Engelmann  <https://orcid.org/0000-0002-8545-0013>

Yuning Jiang  <https://orcid.org/0000-0002-7145-0995>

Timm Faulwasser  <https://orcid.org/0000-0002-6892-7406>

REFERENCES

- Engelmann A, Jiang Y, Houska B, Faulwasser T. Decomposition of nonconvex optimization via bi-level distributed ALADIN. *IEEE Trans Control Netw Syst.* 2020;7(4):1848-1858. doi:10.1109/TCNS.2020.3005079
- Erseghe T. Distributed optimal power flow using ADMM. *IEEE Trans Power Syst.* 2014;29(5):2370-2380. doi:10.1109/TPWRS.2014.2306495
- Kim BH, Baldick R. A comparison of distributed optimal power flow algorithms. *IEEE Trans Power Syst.* 2000;15(2):599-604. doi:10.1109/59.867147
- Dall'Anese E, Zhu H, Giannakis GB. Distributed optimal power flow for smart microgrids. *IEEE Trans Smart Grid.* 2013;4(3):1464-1475. doi:10.1109/TSG.2013.2248175
- Jiang Y, Zanon M, Hult R, Houska B. Distributed algorithm for optimal vehicle coordination at traffic intersections. *IFAC-PapersOnLine.* 2017;50(1):11577-11582.
- Boyd S, Parikh N, Chu E, Peleato B, Eckstein J. Distributed optimization and statistical learning via the alternating direction method of multipliers. *Found Trends Mach Learn.* 2011;3(1):1-122.
- Tippett MJ, Bao J. Distributed model predictive control based on dissipativity. *AIChE J.* 2013;59(3):787-804. doi:10.1002/aic.13868
- Christofides PD, Scattolini R, de la Pena DM, Liu J. Distributed model predictive control: a tutorial review and future research directions. *Comput Chem Eng.* 2013;51:21-41. doi:10.1016/j.compchemeng.2012.05.011
- Stewart BT, Venkat AN, Rawlings JB, Wright SJ, Pannocchia G. Cooperative distributed model predictive control. *Syst Control Lett.* 2010;59(8):460-469. doi:10.1016/j.sysconle.2010.06.005
- Shi Q, He C, Chen H, Jiang L. Distributed wireless sensor network localization via sequential greedy optimization algorithm. *IEEE Trans Signal Process.* 2010;58(6):3328-3340. doi:10.1109/TSP.2010.2045416
- Du X, Engelmann A, Jiang Y, Faulwasser T, Houska B. Distributed state estimation for AC power systems using Gauss-Newton ALADIN. Proceedings of the 2019 IEEE 58th Conference on Decision and Control (CDC); December 2019:1919-1924; IEEE.
- Rabbat M., Nowak R.. Distributed optimization in sensor networks. Proceedings of the 3rd International Symposium on Information Processing in Sensor Networks; 2004:20-27.
- Kar S, Moura José MF, Ramanan K. Distributed parameter estimation in sensor networks: nonlinear observation models and imperfect communication. *IEEE Trans Inf Theory.* 2012;58(6):3575-3605. doi:10.1109/TIT.2012.2191450
- Guo J, Hug G, Tonguz OK. A case for nonconvex distributed optimization in large-scale power systems. *IEEE Trans Power Syst.* 2017;32(5):3842-3851. doi:10.1109/TPWRS.2016.2636811
- Engelmann A, Faulwasser T. Essentially decentralized conjugate gradients; 2021.
- Gould NI, Orban D, Toint PL. GALAHAD, a library of thread-safe Fortran 90 packages for large-scale nonlinear optimization. *ACM Trans Math Softw.* 2003;29(4):353-372. doi:10.1145/962437.962438
- Gondzio J, Grothey A. Parallel interior-point solver for structured quadratic programs: application to financial planning problems. *Ann Operat Res.* 2007;152(1):319-339. doi:10.1007/s10479-006-0139-z
- Frasch JV, Sager S, Diehl M. A parallel quadratic programming method for dynamic optimization problems. *Math Program Comput.* 2015;7(3):289-329. doi:10.1007/s12532-015-0081-7
- Chiang N, Petra CG, Zavala VM. Structured nonconvex optimization of large-scale energy systems using PIPS-NLP. Proceedings of the 2014 Power Systems Computation Conference 2014:1-7
- Lubin M, Petra CG, Anitescu M, Zavala V. Scalable stochastic optimization of complex energy systems. Proceedings of 2011 International Conference for High Performance Computing, Networking, Storage and Analysis SC'11; 2011:1-10; IEEE.
- Petra CG. A memory-distributed quasi-newton solver for nonlinear programming problems with a small number of general constraints. *J Parallel Distrib Comput.* 2019;133:337-348. doi:10.1016/j.jpdc.2018.10.009
- Petra CG, Chiang N, Anitescu M. A structured quasi-newton algorithm for optimizing with incomplete Hessian information. *SIAM J Optim.* 2019;29(2):1048-1075. doi:10.1137/18M1167942
- Schenk O, Gärtner K, Fichtner W, Stricker A. PARDISO: a high-performance serial and parallel sparse linear solver in semiconductor device simulation. *Future Gener Comput Syst.* 2001;18(1):69-78. doi:10.1016/S0167-739X(00)00076-5
- Andreas W, Biegler Lorenz T. On the implementation of an interior-point filter line-search algorithm for large-scale nonlinear programming. *Math Program.* 2006;106(1):25-57. doi:10.1007/s10107-004-0559-y
- Curtis FE, Huber J, Schenk O, Wächter A. A note on the implementation of an interior-point algorithm for nonlinear optimization with inexact step computations. *Math Program.* 2012;136(1):209-227. doi:10.1007/s10107-012-0557-4
- Kourounis D, Fuchs A, Schenk O. Toward the next generation of multiperiod optimal power flow solvers. *IEEE Trans Power Syst.* 2018;33(4):4005-4014. doi:10.1109/TPWRS.2017.2789187
- Wang Y, Yin W, Zeng J. Global convergence of ADMM in nonconvex nonsmooth optimization. *J Sci Comput.* 2019;78(1):29-63. doi:10.1007/s10915-018-0757-z
- Hong M, Luo Z-Q, Razaviyayn M. Convergence analysis of alternating direction method of multipliers for a family of nonconvex problems. *SIAM J Optim.* 2016;26(1):337-364. doi:10.1137/140990309

29. He B, Yuan X. On the $O(1/n)$ convergence rate of the Douglas–Rachford alternating direction method. *SIAM J Numer Anal.* 2012;50(2):700–709. doi:10.1137/110836936
30. Houska B, Frasch J, Diehl M. An augmented Lagrangian based algorithm for distributed non-convex optimization. *SIAM J Optim.* 2016;26(2):1101–1127. doi:10.1137/140975991
31. Andersson JAE, Gillis J, Horn G, Rawlings JB, Diehl M. CasADi: a software framework for nonlinear optimization and optimal control. *Math Program Comput.* 2019;11(1):1–36. doi:10.1007/s12532-018-0139-4
32. Nocedal J, Wright S. *Numerical Optimization*. Springer Science & Business Media; 2006.
33. Hock W, Schittkowski K. Test examples for nonlinear programming codes. *J Optim Theory Appl.* 1980;30(1):127–129.
34. Mehrez MW, Sprodowski T, Worthmann K, et al. Occupancy grid based distributed MPC for mobile robots. Proceedings of the 2017 IEEE/RIS International Conference on Intelligent Robots and Systems (IROS); 2017:4842–4847; IEEE.
35. Rawlings JB, Mayne DQ, Diehl M. *Model Predictive Control: Theory, Computation, and Design*. 2nd ed. Nob Hill Publishing; 2019.
36. Stewart BT, Wright SJ, Rawlings JB. Cooperative distributed model predictive control for nonlinear systems. *J Process Control.* 2011;21(5):698–704. doi:10.1016/j.jprocont.2010.11.004
37. Müller MA, Allgöwer F. Economic and distributed model predictive control: recent developments in optimization-based control. *SICE J Control Measur Syst Integr.* 2017;10(2):39–52. doi:10.9746/jcmsi.10.39
38. Cai X, Tippet M, Xie L, Bao J. Fast distributed MPC based on active set method. *Comput Chem Eng.* 2014;71:158–170.
39. Christofides PD, Liu J, De La Pena DM. *Networked and Distributed Predictive Control: Methods and Nonlinear Process Network Applications*. Springer Science & Business Media; 2011.
40. Bestler A, Graichen K. Distributed model predictive control for continuous-time nonlinear systems based on suboptimal ADMM. *Opt Control Appl Methods.* 2019;40(1):1–23. doi:10.1002/oca.2459
41. Tang W, Daoutidis P. Distributed nonlinear model predictive control through accelerated parallel ADMM. Proceedings of the 2019 American Control Conference (ACC); 2019:1406–1411; Philadelphia.
42. Farokhi F, Shames I, Johansson KH. Distributed MPC via dual decomposition and alternative direction method of multipliers. In: Maestre JM, Negenborn RR, eds. *Distributed Model Predictive Control Made Easy*. Springer; 2014:115–131.
43. Frank S, Rebennack S. An introduction to optimal power flow: theory, formulation, and examples. *IEEE Trans.* 2016;48(12):1172–1197. doi:10.1080/0740817X.2016.1189626
44. Engelmann A, Jiang Y, Mühlfordt T, Houska B, Faulwasser T. Toward distributed OPF using ALADIN. *IEEE Trans Power Syst.* 2019;34(1):584–594. doi:10.1109/TPWRS.2018.2867682
45. Engelmann A, Stomberg G, Faulwasser T. Toward decentralized interior point methods for control; 2021.
46. Bertsekas DP. *Nonlinear Programming*. Athena Scientific; 1999.

How to cite this article: Engelmann A, Jiang Y, Benner H, Ou R, Houska B, Faulwasser T. ALADIN- α —An open-source MATLAB toolbox for distributed non-convex optimization. *Optim Control Appl Meth.* 2022;43(1):4–22. doi: 10.1002/oca.2811

APPENDIX A. IMPLEMENTATION DETAILS

The main ALADIN algorithm is based on Houska et al.,³⁰ but additional practical considerations have been taken into account improving efficiency and numerical stability.

A.1 ALADIN in Detail

Standard ALADIN is summarized in Algorithm 1. Each ALADIN iteration executes three main steps: Step 1 solves local NLPs (A1) for fixed and given values for primal iterates z_i^k and dual iterates λ^k in parallel. The parameter sequences $\{\Sigma_i^k\} > 0$ and $\{\Delta^k\} > 0$ are user-defined—details are described in Subsection A.4.^{‡‡} Note that the equality constraints (1b) and box constraints (1d) are not explicitly detailed in Houska et al.³⁰—we consider them separately here for numerical efficiency reasons.^{§§} Step 2 of Algorithm 1 computes sensitivities such as the gradients of the objective functions $g_i^k = \nabla f_i(x_i^k)$ and positive definite approximations of the local Hessian matrices

Algorithm 1. Augmented Lagrangian Alternating Direction Inexact Newton (ALADIN)

Initialization: Initial guess $(\{z_i^0\}_{i \in S}, \lambda^0)$, choose $\{\Sigma_i^k\} > 0, \{\Delta^k\} > 0, \tau > 0, \epsilon > 0$.

Repeat:

1. *Parallelizable step:* For each $i \in S$, solve

$$\min_{x_i} f_i(x_i) + (\lambda^k)^\top A_i x_i + \|x_i - z_i^k\|_{\Sigma_i^k}^2 \quad \text{subject to} \quad g_i(x_i) = 0, \quad h_i(x_i) \leq 0, \quad \text{and} \quad \underline{x}_i \leq x_i \leq \bar{x}_i. \quad (\text{A1})$$

2. *Termination criterion:* If $\|\sum_{i \in S} A_i x_i^k - b\| \leq \epsilon$ and $\|x^k - z^k\| \leq \epsilon$, return $x^* = x^k$.
3. *Sensitivity evaluation:* Compute and communicate gradients, Hessian approximations and constraint Jacobians according to (A3) and (A5), respectively.
4. *Consensus step:* Solve the coordination QP

$$\begin{aligned} \min_{\Delta x, s} \quad & \sum_{i \in S} \frac{1}{2} \Delta x_i^\top \tilde{B}_i^k \Delta x_i + (g_i^k)^\top \Delta x_i + (\lambda^k)^\top s + \|s\|_{\Delta^k}^2 \\ \text{subject to} \quad & \sum_{i \in S} A_i (x_i^k + \Delta x_i) - b = s \quad | \quad \lambda^{\text{QP}k}, \\ & C_i^k \Delta x_i = 0 \quad \forall i \in S, \end{aligned} \quad (\text{A2})$$

and return Δx^k and $\lambda^{\text{QP}k}$.

5. *Line search:* Update primal and dual variables by

$$z^{k+1} \leftarrow x^k + \alpha^k \Delta x^k \quad \lambda^{k+1} \leftarrow \lambda^k + \alpha^k (\lambda^{\text{QP}k} - \lambda^k),$$

with $\alpha^k = 1$ for a full-step variant. Update Σ_i^k and Δ^k .

$$\tilde{B}_i^k \approx B_i^k = \nabla_{xx}^2 \left\{ f_i(x_i^k) + \kappa_i^{k\top} g_i(x_i) + (\gamma_i^{k\top} \eta_i^{k\top})_{j \in \mathbb{A}_i^k} (\tilde{h}_j(x_i^k))_{j \in \mathbb{A}_i^k} \right\}, \quad (\text{A3})$$

where

$$\mathbb{A}_i^k := \left\{ j \in \{1, \dots, n_{hi} + 2n_{xi}\} \mid (\tilde{h}(x^k))_j > -\tau \right\}$$

is the set of active inequality constraints in subproblems $i \in S$ and $\tau > 0$ is a user-defined parameter which can be specified via the `actMargin` option. Moreover, we define combined inequality constraints

$$\tilde{h}(x)^\top := (h(x)^\top \quad (\underline{x}_i - x_i)^\top \quad (x_i - \bar{x}_i)^\top), \quad (\text{A4})$$

and Jacobians of active constraints

$$C_i^{k\top} := \left(\nabla g_i(x_i^k)^\top \quad (\nabla \tilde{h}_j(x_i^k))_{j \in \mathbb{A}_i^k}^\top \right) \quad (\text{A5})$$

for all $i \in S$. With this information, Step 4 of Algorithm 1 solves an equality constrained quadratic program (A2) serving as a coordination problem. Step 5 of Algorithm 1 updates z^k and λ^k based on the solution to (A2). To achieve global convergence guarantees, the step size parameter $\alpha \in (0, 1]$ has to be properly chosen by a globalization routine. Designing suitable distributed globalization routines is subject of ongoing and future work—we use the full step variant $\alpha = 1$. A smaller stepsize can be specified via the `stepSize` option which might stabilize ALADIN- α for certain problems. Note that time-varying parameter sequences $\{\Delta^k\}$ and $\{\Sigma_i^k\}$ with $\Sigma_i^k, \Delta^k > 0$ might accelerate convergence of ALADIN in practice. Heuristic routines for doing so are described in Subsection A.4. \square

TABLE A1 Centralized QP solvers interfaced in ALADIN- α

Name	MA57	pinv	linsolve
Algorithm	Multifrontal LDL	Based on SVD	LU
Sparse	Yes	Yes	No

A.2 Solving the Coordination QP

The Hessian approximations B_i^k are assumed to be positive definite. Hence, problem (A2) is a strictly convex equality-constrained QP which can be solved via the first order optimality conditions (if C_i^k has full row rank) which is a system of linear equations. There are two possibilities for solving (A2) numerically: either by centralized linear algebra routines or by iterative methods. For centralized computation, several solvers are interfaced in ALADIN- α which can be specified via the `solveQP` option. The available solvers are summarized in Table A1. Note that not all solvers support sparse matrices. MA57 usually performs very well in practice—both in terms of speed and robustness. The second approach to solve (A2) is via iterative and *decentralized* routines such as d-CG and d-ADMM. Details of these decentralized routines are described in Subsection A.6.

A.3 Hessian Approximations

As B_i^k may have zero eigenvalues or may even be indefinite if evaluated via (A3), special care has to be taken.^{##} Here we use a heuristic from Nocedal and Wright³²—other heuristics are possible and might accelerate convergence. Our heuristic “flips” the sign of the negative eigenvalues (if there are any) and modifies the zero eigenvalues to a small positive number δ . The intuition here is that the stepsize in the direction of negative curvature becomes smaller the “more negative” the curvature is. For doing so we compute the eigendecomposition $B_i^k = V_i \Lambda_i V_i^\top$ for each subproblem i locally, where Λ_i is a matrix with the eigenvalues of H_i on its main diagonal and V_i is the matrix eigenvectors. Hence, the regularization reads

$$\tilde{\Lambda}_{jj} := \begin{cases} |\Lambda_{jj}| & \text{if } \Lambda_{jj} < -\delta, \\ \delta & \text{if } |\Lambda_{jj}| \in (-\delta, \delta), \\ \Lambda_{jj} & \text{else,} \end{cases} \quad \text{and} \quad \tilde{B}_i^k := V_i \tilde{\Lambda}_i V_i^\top,$$

with $\delta = 10^{-4}$. Regularization can be activated by the option `reg` and δ can be specified via `regParam`.

As an alternative to exact Hessians with regularization, one can use the BFGS update for successively approximating the exact Hessian based on the gradient of the Lagrangian. This has the advantage that only the gradient of the Lagrangian has to be communicated (which is a vector) instead of the Hessian (which is a matrix). A detailed description on how to use BFGS within ALADIN can be found in Engelmann et al.⁴⁴ The BFGS formula can be activated by the setting the option `Hess` to `BFGS` or to `DBFGS` for damped BFGS. The advantage of damped-BFGS is that it guarantees positive-definiteness of B_i^k regardless of the positive-definiteness of the exact Hessian at the current iterate. Note that in case the nullspace method is used (cf. Subsection A.5), the regularization is done for the reduced Hessian \bar{B}_i^k instead of B_i^k .

A.4 Scaling Matrices

A simple heuristic for the sequences $\{\Sigma_i^k\} > 0$ and $\{\Delta^k\} > 0$ is to start with certain (usually diagonal) initial matrices Σ_i^0, Δ^0 and to multiply them by a factor $r_\Delta, r_\Sigma > 1$ in each iteration, that is,

$$\Sigma_i^{k+1} = \begin{cases} r_\Sigma \Sigma_i^k & \text{if } \|\Sigma_i\|_\infty < \bar{\sigma} \\ \Sigma_i^k & \text{otherwise} \end{cases} \quad \text{and} \quad \Delta^{k+1} = \begin{cases} r_\Delta \Delta^k & \text{if } \|\Delta\|_\infty < \bar{\delta} \\ \Delta^k & \text{otherwise.} \end{cases} \quad (\text{A6})$$

These routines have been successfully used in previous works.^{1,44} An alternative for choosing $\{\Delta^k\}$ is based on the consensus violation for each individual row in (1e). The idea here is to increase the corresponding Δ_{ii}^k to drive the corresponding consensus violation to zero. This technique is common in algorithms based on augmented Lagrangians, compare Bertsekas.^{46(chap. 4.2.2)}. Mathematically this means that we choose

$$\Delta_{cc}^{k+1} = \begin{cases} \beta \Delta_{cc}^k & \text{if } \left| \left(\sum_{i \in S} A_i x_i^k - b \right)_c \right| > \gamma \left| \left(\sum_{i \in S} A_i x_i^{k-1} - b \right)_c \right| \\ \Delta_{cc}^k & \text{if } \left| \left(\sum_{i \in S} A_i x_i^k - b \right)_c \right| \leq \gamma \left| \left(\sum_{i \in S} A_i x_i^{k-1} - b \right)_c \right| \end{cases} \quad \text{for all } c \in 1, \dots, n_c, \quad (\text{A7})$$

with $\gamma \in (0, 1)$ and $\beta > 1$. In ALADIN- α we choose $\beta = 10$ and $\gamma = 0.25$. This rule can be activated by the option `DelUp` and is able to accelerate convergence of ALADIN- α substantially in some cases.

Note that the above heuristics such as regularization or parameter updates do not interfere with the fast local convergence properties of ALADIN- α . They are required for guaranteeing fast local convergence since they ensure that the assumptions made in the local convergence proof of ALADIN such as the positive-definiteness of \bar{B}_i^k are satisfied.³⁰

A.5 The Nullspace Method

The nullspace method can be activated to reduce the dimensionality of the coordination QP (A2), thus reducing communication and computation in the coordination step. The idea is to parameterize the nullspace of the active constraints $\text{null}(C_i^k) := \{x_i \in \mathbb{R}^{n_{xi}} \mid C_i^k x_i = 0\}$ by $\text{null}(C_i^k) = Z_i^k \Delta v_i$, where $Z_i^k \in \mathbb{R}^{n_{xi} \times (n_{xi} - |A_i^k|)}$ is a matrix whose columns are a basis of $\text{null}(C_i^k)$. Note that $C_i^k Z_i^k = 0$ by definition of the nullspace. Using this parametrization, (A2) can be written as

$$\begin{aligned} \min_{\Delta v, s} \quad & \sum_{i \in S} \left\{ \frac{1}{2} \Delta v_i^\top \bar{B}_i^k \Delta v_i + \bar{g}_i^\top \Delta v_i \right\} + (\lambda^k)^\top s + \|s\|_{\Delta^k}^2 \\ \text{subject to} \quad & \sum_{i \in S} \bar{A}_i^k (v_i^k + \Delta v_i) - b = s \quad \mid \lambda^{\text{QP}}, \end{aligned} \quad (\text{A8})$$

where $\bar{B}_i^k = Z_i^{k\top} B_i^k Z_i^k \in \mathbb{R}^{(n_{xi} - |A_i^k|) \times (n_{xi} - |A_i^k|)}$, $\bar{g}_i^k = Z_i^{k\top} g_i^k \in \mathbb{R}^{(n_{xi} - |A_i^k|)}$, and $\bar{A}_i^k = A_i Z_i^k \in \mathbb{R}^{n_c \times (n_{xi} - |A_i^k|)}$. Note that \bar{A}_i^k has an iteration index k and changes during the iterations since Z_i^k changes. Similar to the full-space approach, regularization from Subsection A.3 is used (if it is activated via the option `reg`) yielding a positive definite \bar{B}_i^k . The nullspace method can be used by activating the option `redSpace`. Notice that the required communication between the subproblems and the coordinator is reduced by twice the number of equality constraints and active inequality constraints. Thus, the communication reduction can be large for problems with many constraints. Furthermore, the coordination QP (A2) is in general less expensive to solve since (A8) is of smaller dimension than (A2). Indeed, (A8) is strongly convex under suitable assumptions which (A2) is not necessarily.¹ While computing nullspaces is numerically expensive (due to singular-value decomposition), it is done parallel in our context—thus fostering parallelization.

A.6 Bi-level ALADIN

Bi-level ALADIN is an extension of ALADIN to further reduce dimensionality of the coordination QP (A8). Moreover, it enables the use of decentralized ADMM or essentially decentralized conjugate gradients as inner algorithms leading to an overall (essentially) *decentralized* ALADIN variant.

We briefly recall the main idea of bi-level ALADIN. Under the assumptions from Engelmann et al.,¹ evaluating the KKT conditions for (A8) yields

$$\bar{B}^k \Delta v + \bar{g}^k + \bar{A}^{k\top} \lambda^{\text{QP}} = 0, \quad (\text{A9a})$$

$$\bar{A}^k (v^k + \Delta v) - b - \frac{1}{2} (\Delta^k)^{-1} (\lambda^{\text{QP}} - \lambda^k) = 0, \quad (\text{A9b})$$

where \bar{B}^k , \bar{A}^k , and $\bar{\Delta} v^k$ are block-diagonal concatenations of \bar{B}_i^k , \bar{A}_i^k , and $\bar{\Delta} v_i^k$. Using the *Schur-complement* reveals that (A9) is equivalent to solving the system of linear equations

$$\left(\sum_{i \in S} S_i^k + \frac{1}{2} (\Delta^k)^{-1} \right) \lambda^{\text{QP}} = \sum_{i \in S} s_i^k + \left(\frac{1}{2} (\Delta^k)^{-1} \lambda^k - b \right), \quad (\text{A10})$$

where $S_i^k := \bar{A}_i^k \bar{B}_i^{k-1} \bar{A}_i^{k\top} > 0$ are local Schur-complement matrices and $s_i^k := \bar{A}_i^k \left(v_i^k - \bar{B}_i^{k-1} \bar{g}_i^k \right)$ are local Schur-complement vectors. The key observation for decentralization is that the matrices S_i^k and vectors s_i^k inherit the sparsity pattern of the consensus matrices A_i , that is, zero rows in A_i yield zero rows/columns in S_i^k and s_i^k . Intuitively speaking, each row/column of S_i^k corresponds to one consensus constraint (row of (1e)) and only the subproblems which “participate” in this constraint have non-zero rows in their corresponding S_i . This sparsity can be exploited to solve (A10) in a decentralized fashion. Examples for such algorithms are decentralized ADMM or an essentially decentralized conjugate gradients algorithm presented in Engelmann and Faulwasser.¹⁵