MITSUBISHI ELECTRIC RESEARCH LABORATORIES
http://www.merl.com

An Alternating Direction Method of Multipliers Algorithm for Symmetric MPC

Danielson, C.

TR2018-118 August 25, 2018

Abstract

This paper presents an alternating-direction method of multipliers (admm) algorithm for solving large-scale symmetric model predictive control (mpc) problems in real-time on embedded computers with limited computational and memory resources. Symmetry was used to find transformations of the states, inputs, and constraints of the mpc problem that decompose the dynamics and cost. We prove a key-property of the symmetric group that allows us to efficiently transform between the original and decomposed symmetric domains. This allows us to solve different sub-problems of a baseline admm algorithm in different domains where the computations are less expensive. This reduces the computational cost of each iteration from quadratic in problem size to linear. In addition, we show that our admm algorithm requires a constant amount of memory regardless of the problem size. We demonstrate our algorithm for a battery balancing problem which results in a reduction of computation-times from hours to seconds and a reduction in memory from hundreds of megabytes to tens of kilobytes.

IFAC Nonlinear Model Predictive Control Conference (NMPC)

An Alternating Direction Method of Multipliers Algorithm for Symmetric MPC

Claus Danielson *

* *Mitsubishi Electric Research Laboratories, Cambridge MA*

Abstract: This paper presents an alternating-direction method of multipliers (ADMM) algorithm for solving large-scale symmetric model predictive control (MPC) problems in real-time on embedded computers with limited computational and memory resources. Symmetry was used to find transformations of the states, inputs, and constraints of the MPC problem that decompose the dynamics and cost. We prove a key-property of the symmetric group that allows us to efficiently transform between the original and decomposed symmetric domains. This allows us to solve different sub-problems of a baseline ADMM algorithm in different domains where the computations are less expensive. This reduces the computational cost of each iteration from quadratic in problem size to linear. In addition, we show that our ADMM algorithm requires a constant amount of memory regardless of the problem size. We demonstrate our algorithm for a battery balancing problem which results in a reduction of computation-times from hours to seconds and a reduction in memory from hundreds of megabytes to tens of kilobytes.

1. INTRODUCTION

Model predictive control for large-scale systems is inherently computationally challenging since it requires solving large optimization problems in real-time on embedded hardware with limit computational resources. However, large-scale man-made systems tend to exhibit substantial repetition of mass-produced components that are organized in regular patterns. For instance, a battery pack in an electric vehicle will have hundreds of nearly identical cells wired together in a regular pattern (see Danielson et al. (2012, 2013); Preindl et al. (2013)). Likewise, a heating, ventilation, and air-conditioning (HVAC) system for a large building will typically use the same heat-exchangers, fans, and thermocouples in each thermal zone (see Burns et al. (2018); Danielson (2017)). This organized repetition of components leads to patterns in the resulting control problem called symmetries. Intuitively, it is obvious that these symmetries can be exploited to reduce the computational burden of MPC for large-scale systems. However, it is *not* obvious how to exploit these symmetries, especially when the patterns are obscured by high-dimensional and highly-coupled dynamics, costs, and constraints.

This paper presents an ADMM algorithm that exploits symmetry to reduce the computational and memory burden of solving large-scale MPC problems. We use symmetry to find transformations of the inputs, outputs, and states that decompose the dynamics and cost of the MPC problem. We use this symmetric decomposition to simplify two of the sub-problems solved in each iteration of a baseline ADMM algorithm. The third sub-problem is solved in the original input, output, and state domain. We exploit a key-property of the symmetric group that allows us to perform these transformations with linear complexity. As a result, we are able to show that the computational cost of each iteration of our ADMM algorithm grows linearly with problem size (number of states, inputs, constraints) rather than typical quadratic or cubic complexity growth. Furthermore, since the symmetric transformations are orthogonal they do not adversely affect the convergence rate of the baseline algorithm, resulting in an overall reduction in computational complexity. The memory benefits of our algorithm are even more impressive. We show that the amount of read-only memory required by the algorithm is constant regardless of the problem size. This is a manifestation of our intuition that adding more identical components to a system should not increase the amount of memory needed to describe the system nor the resulting MPC problem. For example, adding another identical cell to a battery pack should not increase the amount of memory needed to describe or control the battery pack.

Symmetry has been exploited for large-scale optimization. For instance, Boyd et al. (2009); Margot (2010); Bodi et al. (2011) exploited the fact that the optimal solution of a symmetric convex optimization problem lies in a lower-dimensional sub-space called the fixed-space. Thus, symmetry can be used to reduce a high-dimensional optimization problem into a low-dimensional problem. Unfortunately, those papers used a stronger definition of symmetry that is extremely restrictive for control applications. In the context of control theory, their definition of symmetry requires that the states, references, and disturbances are symmetric, as well as the dynamics, cost, and constraints. For example, for an HVAC system this requires that all thermal-zones are at the same temperature, have the same temperature set-points, and are subject to the same heat-loads. In contrast, this paper uses a weaker definition of symmetry that does not restrict the states, references, and disturbances of the system. Instead, the symmetries appear in the system behavior. As shown in Chuang et al. (2015); Danielson and Bauer (2015), this symmetric behavior need only approximate e.g. in terms of the system's $\mathcal{H}_\infty$ norm.

Symmetry has also been applied to control theory, see Cogill et al. (2008); Lin et al. (2012); Danielson and Borrelli (2014); Chuang et al. (2015); Danielson and Di Cairano (2015). Most relevant to this paper, are the works Danielson and Borrelli (2012, 2015b,a) on exploiting symmetry for explicit MPC. In those papers, symmetry was used to discard symmetrically redundant portions of the optimal control-law resulting in an exponential decrease in the memory required to store the piecewise affine on polyhedra controller. This paper exploits the same symmetric structure, but for *implicit* rather than *explicit* MPC. Furthermore, this paper uses a different mathematical framework than the previous work; linear representations rather than orbits of finite groups. However, we concentrate on the properties and advantages of a specific symmetry group in order to avoid becoming lost in the abstract algebra of group theory. Details on representation theory can be found in the famous text Serre (1977) and its application to the decomposition of dynamic systems can be found in Cogill et al. (2008); Danielson and Bauer (2015).

This paper is organized as follows. In Section 2 we describe the baseline ADMM algorithm on which our symmetric algorithm is based. In Section 3 we formally define symmetry and describe the symmetric decomposition. In Section 4 we present our symmetry exploiting ADMM algorithm and study its computational and memory benefits. Finally, in Section 5 we apply our ADMM algorithm to the problem of balancing cells in a battery pack. For space reasons proof are omitted. All proofs can be found in the extended version of this paper Danielson (2018).

Notation and Definitions We will use the short-hand $\Theta_{x,u,y} \in \mathbb{R}^{n_{x,u,y} \times n_{x,u,y}}$ to denote a triple of transformations acting on the states $\Theta_x \in \mathbb{R}^{n_x \times n_x}$, inputs $\Theta_u \in \mathbb{R}^{n_u \times n_u}$, and outputs $\Theta_y \in \mathbb{R}^{n_y \times n_y}$ of a system. The notation $[x]_i$ will be used to denote the i-th component of a vector $x \in \mathbb{R}^n$. The Kronecker product $A \otimes B$ of matrices $A \in \mathbb{R}^{n_1 \times m_1}$ and $B \in \mathbb{R}^{n_2 \times m_2}$ is defined as

$$A \otimes B = \begin{bmatrix} [A]_{11}B & \dots & [A]_{1m_1}B \\ \vdots & & \vdots \\ [A]_{n_1 1}B & \dots & [A]_{n_1 m_1}B \end{bmatrix} \in \mathbb{R}^{n_1 n_2 \times m_1 m_2}.$$

The singular value decomposition (SVD) decomposes a matrix $M = U\Sigma V^* \in \mathbb{R}^{n \times m}$ into orthogonal matrices $U \in \mathbb{R}^{n \times n}$ and $V \in \mathbb{R}^{m \times m}$ and a diagonal matrix $\Sigma \in \mathbb{R}^{n \times m}$. The column-vectors of V and U are called in the input and output channels respectively and the diagonal elements of Σ are called the singular values.

2. PROBLEM STATEMENT

In this section we describe the baseline ADMM algorithm on which our symmetric algorithm is based. The baseline ADMM algorithm is based on the design from Raghunathan and Di Cairano (2014a,b).

2.1 Model Predictive Control Problem

This paper studies algorithms for solving the following constrained finite-time optimal control (CFTOC) problem

$$\min \sum_{k=0}^{N-1} \begin{bmatrix} x_k \\ u_k \end{bmatrix}^\mathsf{T} \begin{bmatrix} Q & S \\ S^\mathsf{T} & R \end{bmatrix} \begin{bmatrix} x_k \\ u_k \end{bmatrix} \tag{1a}$$

$$\text{s.t. } x_{k+1} = Ax_k + Bu_k, x_0 = x(t) \tag{1b}$$

$$y_k = Cx_k + Du_k \tag{1c}$$

$$\underline{y} \leq y_k \leq \overline{y} \tag{1d}$$

where x_k is the predicted state over the horizon N under the control inputs u_k initialized $x_0 = x(t)$ at the current state $x(t)$ of the plant. The predicted outputs y_k are signals to be constrained (1d) by the controller. Note that, polytopic state and input constraints can be describe by the bounds (1d) using properly defined matrices C and D in the constrained-outputs (1c). We assume the pairs (A, B) and $(A, Q{-}SRS^\mathsf{T})$ are controllable and observable respectively and $R \succ 0$ positive definite so that (1) has a unique optimal solution.

The customary terminal cost and constraints were omitted from (1) to keep the notion compact and since these features are not relevant to the presented algorithms. Nonetheless, a terminal cost and constraint set can be added to the CFTOC (1) without affecting the results of this paper.

2.2 Baseline ADMM

The ADMM algorithm moves the equality constraints (1c) that define the constrained-outputs y_k into the cost-function to produce the following augmented CFTOC

$$\min \sum_{k=0}^{N-1} \begin{bmatrix} x_k \\ u_k \end{bmatrix}^\mathsf{T} \begin{bmatrix} Q & S \\ S^\mathsf{T} & R \end{bmatrix} \begin{bmatrix} x_k \\ u_k \end{bmatrix} + \rho \|Cx_k + Du_k - y_k - \gamma_k\|^2 \tag{2a}$$

$$\text{s.t. } x_{k+1} = Ax_k + Bu_k, x_0 = x(t) \tag{2b}$$

$$\underline{y} \leq y_k \leq \overline{y} \tag{2c}$$

where γ_k for $k = 0, \dots, N-1$ are dual-variables that ensure the equality constraints (1c) hold at optimality and the step-size ρ is a design parameter for the ADMM algorithm.

The baseline ADMM algorithm is described by Algorithm 1. During each iteration, Algorithm 1 alternately solves (2) with respect to three sets of decision variables; the state $\{x_k\}_{k=0}^{N-1}$ and input $\{u_k\}_{k=0}^{N-1}$ trajectories, the constrained output trajectory $\{y_k\}_{k=0}^{N-1}$, and the dual variable trajectory $\{\gamma_k\}_{k=0}^{N-1}$.

Algorithm 1 Baseline ADMM

Input: State $x(t)$
Output: Input $u(t)$
1: Set $x_0 = x(t)$
2: Set $\{y_k\}_{k=0}^{N-1} = 0$ and $\{\gamma_k\}_{k=0}^{N-1} = 0$
3: **repeat**
4: Update $\{\hat{x}_k^i, \hat{u}_k^i\}_{k=0}^{N-1}$ using (SP1)
5: Update $\{y_k\}_{k=0}^{N-1}$ using (SP2)
6: Update $\{\gamma_k\}_{k=0}^{N-1}$ using (SP3)
7: **until** $\|\gamma_k^+ - \gamma_k\|_\infty < \epsilon, \|y_k^+ - y_k\|_\infty < \epsilon$
8: **return** $u(t) = u_0$

With the constrained-outputs $\{y_k\}_{k=0}^{N-1}$ and dual-variables $\{\gamma_k\}_{k=0}^{N-1}$ fixed, the augmented optimal control problem (2) becomes the following *unconstrained* finite-time optimal control problem

$$(x_k^+, u_k^+) = \underset{x_k, u_k}{\arg\min} \sum_{k=0}^{N-1} \begin{bmatrix} x_k \\ u_k \end{bmatrix}^\mathsf{T} \begin{bmatrix} Q & S \\ S^\mathsf{T} & R \end{bmatrix} \begin{bmatrix} x_k \\ u_k \end{bmatrix} + \rho \left\| Cx_k + Du_k - r_k \right\|^2$$
$$\text{s.t. } x_{k+1} = Ax_k + Bu_k, x_0 = x(t) \tag{SP1}$$

where $r_k = y_k + \gamma_k$. Sub-problem (SP1) trades-off tracking the unconstrained optimal with tracking a constraint satisfying reference $r_k = y_k + \gamma_k$. A solution of sub-problem (SP1) can be found using various standard techniques.

Next, Algorithm 1 solves problem (2) with respect to the constrained-outputs $\{y_k\}_{k=0}^{N-1}$. This produces the following, time-decoupled, constraint-projection problems

$$y_k^+ = \arg\min \rho \|Cx_k^+ + Du_k^+ - y_k - \gamma_k\|^2$$
$$\text{s.t. } \underline{y} \leq y_k \leq \overline{y}$$

for $k = 0, \dots, N$ which has the following closed-form solution

$$y_k^+ = \text{sat}(Cx_k^+ + Du_k^+ - \gamma_k) \tag{SP2}$$

for $k = 0, \dots, N$ where $\text{sat}(\cdot)$ is the saturation function

$$\text{sat}(y) = \begin{cases} y & \text{if } \underline{y} \leq y \leq \overline{y} \\ \underline{y} & \text{if } y < \underline{y} \\ \overline{y} & \text{if } y > \overline{y}. \end{cases}$$

Finally, in each iteration, Algorithm 1 updates the dual-variables using a simple gradient ascent

$$\gamma_k^+ = \gamma_k + \left(Cx_k^+ + Du_k^+ - y_k^+ \right) \tag{SP3}$$

for $k = 0, \dots, N$. The dual-variable update (SP3) can be interpreted as integral-action for sub-problem (SP1) i.e. the constraint violations $v_k = Cx_k + Du_k - y_k$ are integrated (SP3) by the dual-variables γ_k until the reference $r_k = y_k + \gamma_k$ tracked in sub-problem (1) produces an output trajectory y_k that satisfies constraints. In practice, the computation of the constraint-violations $v_k = Cx_k + Du_k - y_k$ are re-used from sub-problem (SP1) to update the dual-variable in linear-time.

Algorithm 1 iterates until the dual-variables $\|\gamma_k^+ - \gamma_k\| = \|Cx_k + Du_k - y_k\| < \epsilon$ and the predicted constrained-outputs $\|y_k^+ - y_k\| < \epsilon$ have converged. For a more detailed description of Algorithm 1 and its properties see Raghunathan and Di Cairano (2014a,b).

3. SYMMETRIC DECOMPOSITION

In this section we formally define symmetry and summarize how symmetry can be used to decompose the CFTOC (1).

3.1 Definition of Symmetry

Intuitively, symmetries are patterns in the CFTOC (1). These patterns can be rigorously defined using linear-operators that map the problem (1) to itself. A symmetry of the CFTOC (1) is defined as

invertible transformations of the inputs, outputs, and states $\Theta_{u,y,x} \in \mathbb{R}^{n_{u,y,x} \times n_{u,y,x}}$ that preserve the dynamics

$$\begin{bmatrix} \Theta_x & 0 \\ 0 & \Theta_y \end{bmatrix}^{-1} \begin{bmatrix} A & B \\ C & D \end{bmatrix} \begin{bmatrix} \Theta_x & 0 \\ 0 & \Theta_u \end{bmatrix} = \begin{bmatrix} A & B \\ C & D \end{bmatrix} \tag{3a}$$

and the cost function

$$\begin{bmatrix} \Theta_x & 0 \\ 0 & \Theta_u \end{bmatrix}^{-1} \begin{bmatrix} Q & S \\ S^\mathsf{T} & R \end{bmatrix} \begin{bmatrix} \Theta_x & 0 \\ 0 & \Theta_u \end{bmatrix} = \begin{bmatrix} Q & S \\ S^\mathsf{T} & R \end{bmatrix}. \tag{3b}$$

For instance, a CFTOC (1) is rotationally symmetric if definition (3) holds for some collection of rotation matrices.

Remark 1. The addition of a terminal cost and constraint to the CFTOC (1) will not necessarily break symmetry. A symmetric stabilizing terminal cost $x^\mathsf{T}Px$ can be designed using a variety of methods. For instance, Danielson (2014) showed that if the dynamics and cost are symmetric (3) then the solution P of the algebraic Riccati equation is symmetric $\Theta_x^{-1} P \Theta_x = P$. A similar result for terminal cost design using linear matrix inequalities was shown in Cogill et al. (2008); Danielson and Di Cairano (2015). Likewise, a symmetric terminal constraint set that provides persistent feasibility can also be found using various methods. Danielson (2014) showed that the maximal control invariant set for symmetric systems is symmetric, furthermore it provided a procedure for constructing a sub-maximal symmetric control invariant set from a sub-maximal non-symmetric control invariant set. Thus, the symmetry requirement (3) will not limit the closed-loop properties of the MPC. ◇

In this paper, we are interested in CFTOCs (1) whose symmetries (3) have the particular form (or can be transformed into the form)

$$\Theta_{y,u,x} = \begin{bmatrix} \Pi \otimes I_{n^1_{u,y,x}} & 0 \\ 0 & I_{n^{m+1}_{u,y,x}} \end{bmatrix} \tag{4}$$

for some m-dimensional permutation matrix $\Pi \in \mathbb{R}^{m \times m}$. The set $\mathfrak{S}_m$ of all permutation matrices $\Pi \in \mathbb{R}^{m \times m}$ is called the symmetric group. Thus, we say the CFTOC (1) is invariant under the symmetric group $\mathfrak{S}_m$ if the definition (3) holds for all permutation matrices of the form (4). The Kronecker product $\Pi \otimes I_{n^1_{u,y,x}}$ means that the symmetries can permute the state-space and cost-function matrices block-wise rather than the more restrictive element-wise permutations.

The symmetries (4) have a fixed subspace for all $\Pi \in \mathfrak{S}_m$ due to the identity matrix $I_{n^{m+1}_{u,y,x}}$. The dimension of this fixed-space $n^{m+1}_{u,y,x}$ relative to the size m of the symmetric group $\mathfrak{S}_m$ provides a measure of the asymmetry of the CFTOC (1) where $mn^1_{u,y,x} + n^{m+1}_{u,y,x} = n_{u,y,x}$. This asymmetry will strongly influence the computational benefits of our symmetric ADMM algorithm.

3.2 Symmetric Decomposition

In this section we describe how symmetry (3) can be used to decompose the dynamics and cost function of the CFTOC (1).

The symmetric decomposition is essentially a common SVD shared by each of the state-space and cost-function matrices. The symmetric decomposition finds orthogonal transformations of the inputs, outputs, and states $\Phi_{u,y,x} \in \mathbb{R}^{n_{u,y,x} \times n_{u,y,x}}$ that decompose the dynamics

$$\begin{bmatrix} \Phi^i_x & 0 \\ 0 & \Phi^i_y \end{bmatrix}^\mathsf{T} \begin{bmatrix} A & B \\ C & D \end{bmatrix} \begin{bmatrix} \Phi^j_x & 0 \\ 0 & \Phi^j_u \end{bmatrix} = \begin{cases} \begin{bmatrix} \hat{A}^{ii} & \hat{B}^{ii} \\ \hat{C}^{ii} & \hat{D}^{ii} \end{bmatrix} & \text{if } i = j \\ 0 & \text{if } i \neq j \end{cases} \tag{5a}$$

and the cost function

$$\begin{bmatrix} \Phi^i_x & 0 \\ 0 & \Phi^i_u \end{bmatrix}^\mathsf{T} \begin{bmatrix} Q & S \\ S^\mathsf{T} & R \end{bmatrix} \begin{bmatrix} \Phi^j_x & 0 \\ 0 & \Phi^j_u \end{bmatrix} = \begin{cases} \begin{bmatrix} \hat{Q}^{ii} & \hat{S}^{ii} \\ \hat{S}^{ii\mathsf{T}} & \hat{R}^{ii} \end{bmatrix} & \text{if } i = j \\ 0 & \text{if } i \neq j \end{cases} \tag{5b}$$

where $\hat{A}^{ii} = \Phi^{i\mathsf{T}}_x A \Phi^i_x \in \mathbb{R}^{n^i_x \times n^i_x}$ and likewise for each of the other dynamics and cost matrices. The tall-matrix $\Phi^i_{u,y,x} \in \mathbb{R}^{n_{u,y,x} \times n^i_{u,y,x}}$ contains a sub-set of the columns of the transformations $\Phi_{u,y,x} \in \mathbb{R}^{n_{u,y,x} \times n_{u,y,x}}$ that defines the i-th channel. Unlike a traditional SVD, the channels of the symmetric decomposition are defined by multiple column-vectors and the singular values (5) are blocks rather than scalars. Throughout this paper, we will use the hat-notation $\hat{\cdot}$ to denote variables and problems represented in the symmetry adapted basis $\Phi_{u,y,x}$. A procedure for computing the decomposition (5) for a generic symmetry group (3) was presented in Murota et al. (2010); de Klerk et al. (2011); Danielson and Bauer (2015).

In this paper we are concerned with CFTOCs (1) with a specific symmetry group, the symmetric group $\mathfrak{S}_m$. For the symmetric group $\mathfrak{S}_m$ with symmetries of the form (4), the symmetric transformations have the form

$$\Phi_{u,y,x} = \begin{bmatrix} I_{n^1_{u,y,x}} \otimes \Phi & 0 \\ 0 & I_{n^{m+1}_{u,y,x}} \end{bmatrix} \tag{6}$$

where the dimensions $n^1_{u,y,x}$ and $n^{m+1}_{u,y,x}$ match those in (4) and

$$\Phi = \begin{bmatrix} 1 & 1 & 1 & \dots & 1 \\ -1 & 1 & 1 & \dots & 1 \\ & -2 & 1 & \dots & 1 \\ & & \ddots & \ddots & \vdots \\ & & & -m{+}1 & 1 \end{bmatrix} \Lambda \in \mathbb{R}^{m \times m}, \tag{7}$$

where $\Lambda \in \mathbb{R}^{m \times m}$ is a diagonal matrix, with elements $\lambda_{ii} = 1/\sqrt{i^2 + i}$ for $i = 1, \dots, m-1$ and $\lambda_{mm} = 1/\sqrt{m}$, that ensures that the column-vectors of Φ are unit-vectors. The structure of the transformations (7) will play an important role in the computational complexity of our symmetric ADMM algorithm.

For the symmetric group $\mathfrak{S}_m$, the generic decomposition (5) of the dynamics and cost matrices has additional structure, namely that first $m-1$ sub-systems and sub-costs are identical

$$\begin{bmatrix} \hat{A}^{ii} & \hat{B}^{ii} \\ \hat{C}^{ii} & \hat{D}^{ii} \end{bmatrix} = \begin{bmatrix} \hat{A}^{11} & \hat{B}^{11} \\ \hat{C}^{11} & \hat{D}^{11} \end{bmatrix} \tag{8a}$$

$$\begin{bmatrix} \hat{Q}^{ii} & \hat{S}^{ii} \\ \hat{S}^{ii\mathsf{T}} & \hat{R}^{ii} \end{bmatrix} = \begin{bmatrix} \hat{Q}^{11} & \hat{S}^{11} \\ \hat{S}^{11\mathsf{T}} & \hat{R}^{11} \end{bmatrix} \tag{8b}$$

for $i = 1, \dots, m-1$. The fixed dynamics $i = m$ are different and typically have a different input, output, and state dimensions $n^1_{u,y,x} \neq n^1_{u,y,x} + n^{m+1}_{u,y,x}$. The explicit repetition (8) of problem-data will be used to reduce the memory required for the symmetric ADMM algorithm.

4. SYMMETRIC ADMM ALGORITHM

In this section we present a symmetry exploiting variant of Algorithm 1. We compare the computational and memory costs of our symmetric ADMM algorithm with the baseline design.

4.1 Symmetric ADMM

In this section we present our symmetric variant of Algorithm 1 and show that these algorithms are equivalent.

Algorithm 2 Symmetric ADMM

Input: State $x(t)$
Output: Input $u(t)$
1: Set $\hat{x}^i_0 = \Phi^{i\mathsf{T}}_x x(t)$ for $i = 1, \dots, m$
2: Set $\{\hat{y}^i_k\}_{k=0}^{N-1} = 0$ and $\{\hat{\gamma}^i_k\}_{k=0}^{N-1} = 0$ for $i = 1, \dots, m$
3: **repeat**
4: Update $\{\hat{x}^i_k, \hat{u}^i_k\}_{k=0}^{N-1}$ using (ŝP1) for $i = 1, \dots, m$
5: Update $\{\hat{y}^i_k\}_{k=0}^{N-1}$ using (ŝP2) for $i = 1, \dots, m$
6: Update $\{\hat{\gamma}^i_k\}_{k=0}^{N-1}$ using (ŝP3) for $i = 1, \dots, m$
7: **until** $\|{\gamma_k}^+ - \gamma_k\|_\infty < \epsilon, \|{y_k}^+ - y_k\|_\infty < \epsilon$
8: **return** $u(t) = \Phi_u \hat{u}_0$

The symmetry exploiting variant of Algorithm 1 is described by Algorithm 2. The main difference between Algorithms 1 and 2 is

that the optimal control sub-problem (SP1) is solved in the symmetric domain. Using the symmetric decomposition (5), the unconstrained optimal control problem (SP1) can be decomposed into m decoupled sub-problems

$$\min_{\hat{x}_k^i, \hat{u}_k^i} \sum_{k=0}^{N-1} \begin{bmatrix} \hat{x}_k^i \\ \hat{u}_k^i \end{bmatrix}^\mathsf{T} \begin{bmatrix} \hat{Q}^{ii} & \hat{S}^{ii} \\ \hat{S}^{ii\mathsf{T}} & \hat{R}^{ii} \end{bmatrix} \begin{bmatrix} \hat{x}_k^i \\ \hat{u}_k^i \end{bmatrix} + \rho \|\hat{C}^{ii}\hat{x}_k^i + \hat{D}^{ii}\hat{u}_k^i - \hat{r}_k^i\|^2$$
$$\text{s.t. } \hat{x}_{k+1}^i = \hat{A}^{ii}\hat{x}_k^i + \hat{B}^{ii}\hat{u}_k^i, \hat{x}_0^i = \hat{x}^i(t) \qquad (\hat{\text{SP1}})$$

for $i = 1, \dots, m$ where $\hat{x}^i(t) = \Phi_x^{i\mathsf{T}} x(t)$ and $\hat{r}_k^i = \Phi_y^{i\mathsf{T}} r_k$ are the i-th components of the initial state $x(t)$ and reference r_k in the symmetric domain. In the next section we will show that the m sub-problems ($\hat{\text{SP1}}$) can be solved either sequentially or in parallel for a reduction in computational cost. However, first we show that sub-problems ($\hat{\text{SP1}}$) for $i = 1, \dots, m$ are equivalent to the original sub-problem (SP1) in the following lemma.

Lemma 1. The state $\{\hat{x}_k^i\}_{k=0}^{N-1} = \{\Phi_x^{i\mathsf{T}} x_k\}_{k=0}^{N-1}$ and input $\{\hat{u}_k^i\}_{k=0}^{N-1} = \{\Phi_u^{i\mathsf{T}} u_k\}_{k=0}^{N-1}$ trajectories produced by solving sub-problems ($\hat{\text{SP1}}$) are the transformations of the state $\{x_k\}_{k=0}^{N-1}$ and input $\{u_k\}_{k=0}^{N-1}$ trajectories produced by solving sub-problem (SP1).

Algorithm 2 solves the constraint-projection problems (SP2) in the original domain, since transforming simple constraint (1d) into the symmetric domain can complicate the constraints. The saturation (SP2) and transformations Φ, Φ^T can be combined into a single operation

$$\hat{y}_k^{i+} = \Phi_y^{i\mathsf{T}} \operatorname{sat}\left(\Phi_y(\hat{C}\hat{x}_k^+ + \hat{D}\hat{u}_k^+ - \hat{\gamma}_k)\right) \qquad (\hat{\text{SP2}})$$

for $k = 0, \dots, N-1$ and $i = 1, \dots, m$ where $\hat{\gamma}_k = \Phi_y^\mathsf{T}\gamma_k$ are the symmetric transformations of the dual-variables γ_k. The dual-variable updates (SP3) can also be decomposed

$$\hat{\gamma}_k^{i+} = \hat{\gamma}_k^i + \hat{C}^{ii}\hat{x}_k^{i+} + \hat{D}^{ii}\hat{u}_k^{i+} - \hat{y}_k^{i+} \qquad (\hat{\text{SP3}})$$

for $k = 0, \dots, N-1$ and $i = 1, \dots, m$. The decoupled dual-variable updates ($\hat{\text{SP3}}$) can be performed in parallel, although with minuscule practical computational benefit.

The following theorem shows that Algorithms 1 and 2 are equivalent.

Theorem 1. Algorithms 1 and 2 terminate after the same number of iterations and produce the equivalent optimal solutions $u_0 = \Phi_u \hat{u}_0$.

Theorem 1 means that Algorithms 1 and 2 require the same number of iterations to reach the same optimal solution. The benefit of Algorithm 2 is that the computational cost of each iteration is cheaper than Algorithm 1. This is shown in the following section.

4.2 Computational Complexity

In this section we compare the computational complexities of Algorithms 1 and 2. First, we show that the decoupled sub-problems ($\hat{\text{SP1}}$) are significantly cheaper to solve than the original optimal control problem (SP1). Obviously, the computational complexity depends on the method used to solve the problems (SP1) and ($\hat{\text{SP1}}$). Therefore, we will assume that these sub-problems have a fixed step-size ρ so that the matrix-factorizations can be performed offline and only matrix-vector products need to be performed online.

Lemma 2. Sub-problem (SP1) has computational complexity $O(Nn^2)$ where $n = \max\{n_{x,u,y}\}$. Sub-problems ($\hat{\text{SP1}}$) for $i = 1, \dots, m$ have computational complexity $O\left(N(mn_1^2 + n_{m+1}^2)\right)$ when solved sequentially on a single-processor and $O(N(n_1 + n_{m+1})^2)$ when solve in parallel on m processors where $n_1 = \max\{n_{x,u,y}^1\}$ and $n_{m+1} = \max\{n_{x,u,y}^{m+1}\}$.

If the dimension of the fixed-space of the symmetries (4) is small $n_{m+1} \ll n_1$ then $n^2 = \left(\sum_{i=1}^m n_i\right)^2 \approx n_1^2 m^2$ and $n_1 + n_{m+1} \approx n_1$. Thus, solving sub-problems ($\hat{\text{SP1}}$) sequentially is approximately m times faster than solving sub-problem (SP1) and solving sub-problems ($\hat{\text{SP1}}$) in parallel is approximately m^2 times faster where m is the size of the symmetry group $\mathfrak{S}_m$. Thus, for CFTOCs with a high-degree of symmetry $m \gg 1$ the decomposition ($\hat{\text{SP1}}$) provides a massive reduction in computational complexity. Indeed, the computational cost of solving sub-problems ($\hat{\text{SP1}}$) in parallel remains constant regardless of the problem size. This is a reasonable since symmetry allows us to efficiently use parallel computationally resource which we assume are growing with problem size.

However, Algorithm 2 is only useful if the computational savings shown in Lemma 2 outweigh the additional computational burden of transforming between domains in sub-problem ($\hat{\text{SP2}}$). Naively, this is not the case since the applying the transformations Φ_y for $k = 0, \dots, N$ using generic matrix-vector multiplication has complexity $O(Nn_y^2)$. Since typically the number of constrained-outputs $n_y \geq n_x + n_u$ is larger than the number of states n_x and inputs n_u, this would increase the computational cost of Algorithm 2. However, we will show that by exploiting the special structure (7) of the transformation Φ we can transform between domains with linear-complexity.

Algorithm 3 Symmetric transformation $\hat{z} = \Phi^\mathsf{T} z$

Input: Original vector z
Output: Transformed vector $\hat{z}^i$ for $i = 1, \dots, m$
1: $s^0 = 0$
2: **for** $i = 1, \dots, m-1$ **do**
3: $\quad s^i = s^{i-1} + z^i$
4: $\quad \hat{z}^i = \lambda_{ii}(s^i - iz^{i+1})$
5: **end for**
6: $\hat{z}^m = \lambda_{mm} s^m$

Algorithm 3 provides a computationally efficient method for transforming $\hat{z} = \Phi^\mathsf{T} z$ a vector z into the symmetric domain. In each iteration, Algorithm 3 computes a running-sum $s_i = \sum_{j=1}^i z^i$ of the elements z^i of the vector $z \in \mathbb{R}^m$. The i-th component $\hat{z}^i$ of the transformed vector $\hat{z} = \Phi^\mathsf{T} z$ is the scaled difference $\hat{z}^i = \lambda_{ii}(s^i - iz^{i+1})$ between the running-sum s^i and the next element z^{i+1} of the original vector z where $\lambda_{ii} = 1/\sqrt{i^2 + i}$. The final element $\hat{z}^m = \lambda_{mm} s^m$ of the transformed vector $\hat{z} = \Phi^\mathsf{T} z$ is given by the total running-sum $s_m = \sum_{j=1}^m z^i$ normalized by $\lambda_{mm} = 1/\sqrt{m}$. A similar algorithm for the inverse-transformation is described in the extended version of this paper Danielson (2018).

Algorithm 3 and its inverse are used in each iteration of the symmetric ADMM Algorithm 2 for evaluating sub-problems ($\hat{\text{SP2}}$). Furthermore, Algorithm 3 is used to pre-process $\hat{x}_0^i = \Phi_x^{i\mathsf{T}} x(t)$ the current state $x(t)$ of the system. And the inverse of Algorithm 3 is used to post-process $u_0 = \sum_{i=1}^m \Phi_u^i \hat{u}_0^i$ the implemented control-input $u(t) = u_0$.

The following lemma shows that Algorithm 3 correctly performs the symmetric transformation with linear complexity.

Lemma 3. Algorithm 3 computes $\hat{z} = \Phi^\mathsf{T} z$ with $O(m)$ complexity.

Lemma 3 shows that the computational cost of transforming between domains is insignificant in comparison to the computational benefits shown in Lemma 2. This is the pivotal property of the symmetric transformation that makes Algorithm 2 viable. In contrast, a generic SVD does not have this property and therefore would not provide the computational benefits of the symmetric decomposition.

The following theorem combines Lemmas 2 and 3 to show that Algorithm 2 has cheaper iteration than Algorithm 1.

Theorem 2. The main iteration of Algorithm 1 has computational complexity $O(Nn^2)$ where $n = \max\{n_{x,u,y}\}$. The main iteration of Algorithm 2 has computational complexity $O\left(N(mn_1^2 + n_{m+1}^2)\right)$ when implemented sequentially and $O(Nn)$ when implemented in parallel on m processors.

According to Theorem 1, Algorithms 1 and 2 required the same number of iterations to converge to the optimal solution. Therefore, Theorem 2 means that Algorithm 2 is more computationally efficient

than Algorithm 1 since its iteration are cheaper. Indeed, Algorithm 2 is asymptotically $O(m)$ times faster than Algorithms 1 when implemented either sequentially on a single processor or in parallel on m processors, although typically the constant $O(m)$ reduction factor is higher for the parallel implementation than the sequential implementation.

Remark 2. In our experience, the most common symmetry groups encountered in real-world applications are the cyclic $\mathfrak{C}_m$, dihedral $\mathfrak{D}_m$, symmetric $\mathfrak{S}_m$ and hyperoctohedral $\mathfrak{H}_m$ groups (see Danielson (2014)). In fact, we are not aware of any real-world applications with a different symmetry group. The dihedral group $\mathfrak{D}_m$ is a signed version of the cyclic group $\mathfrak{C}_m$ and the hyperoctohedral group $\mathfrak{H}_m$ is a signed version of the symmetric group $\mathfrak{S}_m$ where the sign does not affect the symmetric decomposition (5). For cyclic/dihedral groups, the symmetric transformation Φ is the Fourier transform matrix. It is well known that the Fourier transform can be applied in $O(m \log m)$ time, which is slower than Algorithm 3. This is most likely due to the fact that the cyclic group $|\mathfrak{C}_m| = m$ is smaller than the symmetric group $|\mathfrak{S}_m| = m!$. Thus, the ADMM Algorithm 2 can also be applied to problems (1) with cyclic or dihedral symmetry where the optimal control problem (SP1) is solved in the spatial Fourier domain. It is unclear whether fast algorithms exist for performing the symmetric decomposition (5) for generic symmetry groups (3). But certainly, Algorithm 2 provides computational benefits for the four most common symmetry groups encountered in practice. □

4.3 Memory Complexity

The symmetric ADMM Algorithm 2 is intended for large-scale $n_{u,y,x} \gg 1$ problems (1) with a lot of symmetry $m \gg 1$. For large-scale problems, storing the problem-data in memory on an embedded platform can be nonviable. In addition, the computational benefits provided by Algorithm 2 are often so drastic that the computational bottleneck becomes accessing memory rather than the linear algebra computations. Fortunately, the symmetry of the problem (1) can also be used to reduce the memory requirements of the algorithm.

The following theorem shows the memory benefits of the symmetric algorithm.

Theorem 3. Algorithms 1 and 2 have memory complexity $O(Nn^2)$ and $O(N(n_1^2 + (n_1 + n_{m+1})^2))$ where $n = \max\{n_{x,u,y}\}$, $n_1 = \max\{n^1_{x,u,y}\}$, and $n_{m+1} = \max\{n^{m+1}_{x,u,y}\}$

Theorem 3 says that the amount of read-only memory required for the symmetric ADMM Algorithm 2 does not grow with the problem size $n = mn_1 + n_{m+1}$ where n_1 and n_{m+1} are fixed. This reflects our intuition that the memory required to describe the CFTOC (1) should not increase due to repetition in the system. Note that the memory needed to store the system signals (inputs u_k, outputs y_k, states x_k, and dual-variables γ_k) stills grows with problem size $n_{u,y,x} = mn^1_{u,y,x} + n^{m+1}_{u,y,x}$.

5. CASE STUDY: BATTERY BALANCING

In this section we apply the ADMM algorithms to a battery balancing control problem. Details about this problem can be found in Danielson et al. (2012, 2013); Preindl et al. (2013).

The objective of the battery balancing problem is to redistribute charge in a battery of m cells so that every cell has the same state-of-charge. Model Predictive Control (MPC) is used to achieve fast and energy-efficient balancing. This is an ideal application for Algorithm 2 since the problem is computationally challenging for large-scale $m \gg 1$ battery packs, but also very symmetric.

The battery cell dynamics (1b) are modeled as integrators where $A = I \in \mathbb{R}^{m\times m}$ and $B \in \mathbb{R}^{m\times m}$ describes the balancing circuit topology. The state $x(t) \in \mathbb{R}^m$ is the state-of-charge of each cell and $u(t)$ is the balancing current. There are constraints on the minimum $\underline{x}$ and maximum $\overline{x}$ state-of-charge for each cell. In addition, the balancing circuit can only draw charge from a single cell at a time. This integral constraint can be relax into a 1-norm ball $\|u(t)\|_1 \leq 1$ constraint as shown in Preindl et al. (2013). These state and input constraints can be expressed as $4m+1$ constraints of the form (1c) and (1d) using the standard trick of splitting the input $u(t) = u^+(t) - u^-(t)$ into positive and negative $u^+(t), u^-(t) \geq 0$ components with $\sum_{i=1}^m u_i^+(t) + u_i^-(t) \leq 1$. The cost-function (1a) penalizes the cell imbalance and usage of balancing current where $Q = I - \frac{1}{m}11^\mathsf{T} \in \mathbb{R}^{m\times m}$, $R = rI \in \mathbb{R}^{2m\times 2m}$, and $S = 0 \in \mathbb{R}^{m\times 2m}$. The CFTOC (1) is solved every 1 minute with a prediction horizon of 10 minutes.

Intuitively, this problem is symmetric since the battery cells are nearly identical and they are each connected to the balancing circuit in the same manner. Numerically, it can be easily verified that the resulting CFTOC (1) is invariant (3) under the symmetric group (4) where $(n^1_x, n^{m+1}_x) = (1, 0)$, $(n^1_u, n^{m+1}_u) = (2, 0)$, and $(n^1_y, n^{m+1}_y) = (4, 1)$. Note that only the constraints have a non-trivial fixed-space $n^{m+1}_y = 1 > 0$. This fixed-space is a result of the constraint $\sum_{i=1}^m u_i^+(t) + u_i^-(t) \leq 1$ on the total balancing current i.e. permuting the balancing currents does not change their sum.

The battery balancing problem was used to evaluate the performance of the ADMM Algorithms 1 and 2. The number of cells m in the battery pack was varied from $m = 10$ cells, which would be found in an appliance, to $m = 100$ cells, which would be found in an electric vehicle. For each battery size m, the CFTOC (1) was solved for 10 randomly generated initial cell imbalances $x(t)$. The CFTOCs (1) were solved in MATLAB using a single-core, with code acceleration disable, and using brute-force matrix-vector multiplication in order to emulate the performance of an embedded platform. The termination toleration used for all algorithms was $\epsilon = 10^{-8}$.

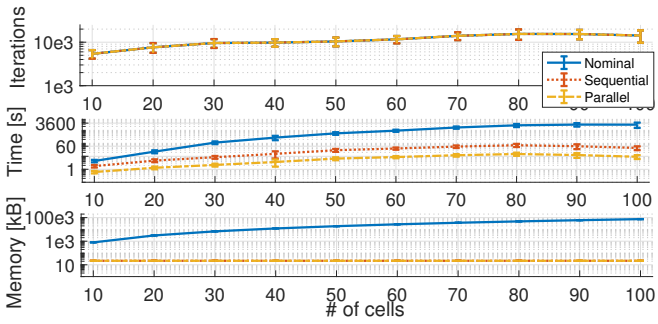

Fig. 1. Comparison of the ADMM algorithms (a) number of iterations and (b) solver-time and (c) solver-memory.

Fig. 1 shows the performance of Algorithm 1 and Algorithm 2 with both a sequential and parallel implementation. The first sub-figure shows the number of iterations required to solved the CFTOC (1). Note that the number of iterations is identical for each of the algorithms. This empirically verifies Theorem 1 which showed that these algorithms are equivalent, and therefore have the same convergence rate and terminate after the same number of iterations.

On the other hand, the second sub-figure shows that both the sequential and parallel implement of the symmetric ADMM Algorithm 2 are significantly faster than the baseline Algorithm 1. This empirically verifies Theorem 2 which showed the computational benefits of symmetry. The computational cost of the symmetric algorithms grow linearly with the pack size m instead of quadratically for the baseline algorithm. As a result, the symmetric algorithms outperform the baseline algorithm as the number of cells increases. For the largest battery pack $m = 100$, the baseline Algorithm 1 spent more than 1 hour solving the CFTOCs (1) whereas the sequential implementation of the symmetric Algorithm 2 solved the CFTOCs (1) in less than 1 minute, which is the typical sample period for this problem. The parallel implementation of the symmetric ADMM was even faster, spending approximately 10 seconds for the largest problem.

The memory benefits of the symmetric ADMM Algorithm 2 over the baseline Algorithm 1 are even more impressive. The third subfigure

in Fig. 1 shows the memory usages for each of the algorithms. The memory required for the symmetric algorithms is constant regardless of the pack size m whereas the memory required for the baseline Algorithm 1 grows quadratically with the problem size, which empirically verifies Theorem 3. For the largest battery pack $m = 100$, the baseline Algorithm 1 required over 100 megabytes of memory whereas the symmetric ADMM Algorithm 2 required approximately 10 *kilo*bytes.

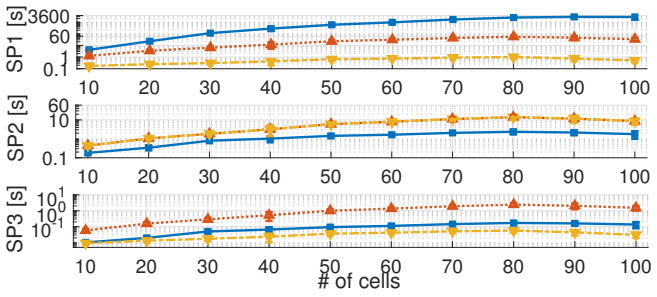

Fig. 2. Comparison of time spent solving each of the sub-problems by each of the ADMM algorithms.

The computational benefits of the symmetric algorithms are further demonstrated by Fig. 2 which shows the computation-time spent solving each of the sub-problems. The first subfigure shows the time spent solving the optimal control sub-problems (SP1) and ($\hat{\text{SP}}$1). This sub-problem dominated the computation-time for the baseline Algorithm 1 and the sequentially implemented symmetric Algorithm 2. For the parallel implementation of the symmetric Algorithm 2 the computation-time is constant since the computational resources grow with the number of cells m. This result empirically verifies Lemma 2.

The second subfigure of Fig. 2 shows the computation-time for solving the constraint-projection sub-problems (SP2) and ($\hat{\text{SP}}$2). For the symmetric Algorithm 2, the computation-time includes the time required to transform the constrained outputs y_k between the symmetric and original domains. As a result, computation-time of the constraint-projection sub-problem ($\hat{\text{SP}}$2) is increased compared to (SP2), although only by a constant factor that does not grow with problem-size m. This verifies our key-result, Lemma 3 which allows for the rapid transformation between domains. Note that since sub-problem ($\hat{\text{SP}}$2) cannot be (spatially) parallelized, the computation-time is identical for both the sequential and parallel variants of Algorithm 2.

The third subfigure of Fig. 2 shows the computation-time for updating the dual-variables (SP3) and ($\hat{\text{SP}}$3). This figure shows that updating the dual-variables in the symmetric domain ($\hat{\text{SP}}$3) is indeed faster than in the original domain (SP3). However, since the dual-variables only require a faction-of-a-second to update, this sub-problem is completely dominated by the other sub-problems. Thus, the computational benefits of decomposing the dual-variable updates are insignificant.

6. CONCLUSIONS AND FUTURE WORK

This paper presented an ADMM algorithm for solving large-scale MPC problems. Using the symmetric decomposition (8), the optimal control sub-problem (SP1) and dual-variable updates (SP3) of a baseline ADMM Algorithm 1 were decomposed ($\hat{\text{SP}}$1) and ($\hat{\text{SP}}$3) respectively. The iterations of the resulting symmetric ADMM Algorithm 2 has linear complexity instead of the typically quadratic. Furthermore, the symmetric ADMM Algorithm 2 has constant memory requirements, independent from the problem size. The symmetric ADMM Algorithm 2 was applied to a battery balancing problem where it reduced computation-times from hours to seconds and reduced memory requirements from megabytes to kilobytes.

In future work, we plan to apply our symmetric ADMM algorithm to other applications, in particular HVAC. In addition, we plan to use symmetry to improve other baseline optimization algorithms, for instance interior-point algorithms.

REFERENCES

Bodi, R., Herr, K., and Joswig, M. (2011). Algorithms for highly symmetric linear and integer programs. *Mathematical Programming*.

Boyd, S., Diaconis, P., Parrilo, P., and Xiao, L. (2009). Fastest mixing markov chain on graphs with symmetries. *Journal on Optimization*.

Burns, D., Danielson, C., Zhou, J., and Di Cairano, S. (2018). Reconfigurable model predictive control for multi-evaporator vapor compression systems. *Trans on Control Systems Technology*.

Chuang, F., Danielson, C., and Borrelli, F. (2015). Robust approximate symmetric model predictive control. In *Conference on Decision and Control*.

Cogill, R., Lall, S., and Parrilo, P. (2008). Structured semidefinite programs for the control of symmetric systems. *Automatica*.

Danielson, C. (2014). *Symmetric Constrained Optimal Control: Theory, Algorithms, and Applications*. Ph.D. thesis, University of California, Berkeley.

Danielson, C. (2017). Symmetric control design for multi-evaporator vapor compression systems. In *Dynamic Systems and Control Conference*.

Danielson, C. (2018). An alternating direction method of multipliers algorithm for symmetric mpc. *merl.com*.

Danielson, C. and Bauer, S. (2015). Numerical decomposition of symmetric linear systems. In *Conference on Decision and Control*.

Danielson, C. and Borrelli, F. (2012). Symmetric explicit model predictive control. In *IFAC Nonlinear MPC Conference*.

Danielson, C. and Borrelli, F. (2014). Identification of the symmetries of linear systems with polytopic constraints. In *American Control Conference*.

Danielson, C. and Borrelli, F. (2015a). Symmetric constrained optimal control. In *IFAC Nonlinear MPC Conference*.

Danielson, C. and Borrelli, F. (2015b). Symmetric linear model predictive control. *Trans on Automatic Control*.

Danielson, C., Borrelli, F., Oliver, D., Anderson, D., Kuang, M., and Phillips, T. (2012). Balancing of battery networks via constrained optimal control. In *American Control Conference*, 0743–1619.

Danielson, C., Borrelli, F., Oliver, D., Anderson, D., and Phillips, T. (2013). Constrained flow control in storage networks: Capacity maximization and balancing. *Automatica*.

Danielson, C. and Di Cairano, S. (2015). Reduced complexity control design for symmetric LPV systems. In *Conference on Decision and Control*.

de Klerk, E., Dobre, C., and Ṗasechnik, D. (2011). Numerical block diagonalization of matrix *-algebras with application to semidefinite programming. *Mathematical Programming*.

Lin, F., Fardad, M., and Jovanović, M. (2012). Sparse feedback synthesis via the alternating direction method of multipliers. In *American Control Conference*.

Margot, F. (2010). Symmetry in integer linear programming. In *50 years of linear programming 1958-2008*. Springer.

Murota, K., Kanno, Y., Kojima, M., and Kojima, S. (2010). A numerical algorithm for block-diagonal decomposition of matrix *-algebras with application to semidefinite programming. *Japan Journal of Industrial and Applied Mathematics*.

Preindl, M., Danielson, C., and Borrelli, F. (2013). Performance evaluation of battery balancing hardware. In *European Control Conference*.

Raghunathan, A. and Di Cairano, S. (2014a). Alternating direction method of multipliers for strictly convex quadratic programs: Optimal parameter selection. In *American Control Conference*.

Raghunathan, A. and Di Cairano, S. (2014b). Infeasibility detection in alternating direction method of multipliers for convex quadratic programs. In *Conference on Decision and Control*.

Serre, J. (1977). *Linear representations of finite groups*. Springer-Verlag.