Optimal Control Applications and Methods

WILEY

Allergy EUROPEAN JOURNAL OF ALLERGY AND CLINICAL IMMUNOLOGY EAACI

ORIGINAL ARTICLE OPEN ACCESS

An adaptive multiple shooting strategy for optimal control

Robert Lampel[1] | Sebastian Sager[1,2]

[1]Otto von Guericke University Magdeburg, Germany | [2]Max Planck Institute for the Dynamics of Complex Technical Systems Magdeburg, Germany |

Correspondence: Robert Lampel (robert.lampel@ovgu.de)

Received: **Revised:** **Accepted:**

Academic Editor: | **Guest Editor:**

Funding: Deutsche Forschungsgemeinschaft (DFG, German Research Foundation) via grants 314838170, GRK 2297 MathCoRe and SA 2016/3-1, SPP 2331 and European Regional Development Fund via grants timingMatters and IntelAlgen, under the European Union's Horizon Europe Research and Innovation Program.

Keywords: Newton-type Methods | Direct Optimal Control | Lifting | Multiple Shooting | Condensing

ABSTRACT

The use of multiple shooting has become the standard for the numerical solution of optimal control problems. We investigate how multiple shooting affects the convergence properties of Newton-type methods. For the first time, we conduct a systematic comparison of several multiple shooting strategies on a set of 40 optimal control problems. In addition, we consider differences between interior-point and sequential quadratic programming methods, accounting for both Quasi-Newton approximations and exact Hessians. Based on these observations, we propose an adaptive multiple shooting algorithm that reduces the number of iterations by about 27% on average across all problems and by more than 50% for selected problems compared with naïve multiple shooting approaches.

1 | Introduction

The numerical solution of Optimal Control Problems (OCPs) is a cornerstone of modern engineering, enabling the optimization of complex dynamic systems ranging from aerospace trajectories to chemical reactors. Among the various numerical strategies, direct methods have gained prominence by transforming the continuous infinite-dimensional problem into a finite-dimensional Nonlinear Programming (NLP) problem [1]. Specifically, the Direct Multiple Shooting method, pioneered by Bock and Plitt [2], has become a standard approach due to its superior stability in handling unstable systems and its ability to utilize state-of-the-art ODE and DAE solvers [3].

While traditional multiple shooting relies on a fixed uniform time grid, many real-world applications involve "stiff" dynamics or control trajectories with localized high-frequency behavior. In such cases, a uniform grid is often computationally inefficient, either being too coarse to capture critical transients or too fine, leading to an unnecessarily large NLP. This limitation has motivated the development of adaptive multiple shooting techniques.

By incorporating iterative grid refinement strategies, the shooting nodes can be dynamically redistributed to areas of high discretization error or nonsmoothness. More advanced schemes utilize multiscale analysis, such as wavelets, to automatically detect where additional resolution is required [4]. These adaptive frameworks ensure that the solution achieves a prescribed accuracy while maintaining the computational efficiency required for demanding applications, such as real-time nonlinear model predictive control (NMPC) [5].

Multiple shooting is a special case of lifting, i.e., the process of transforming a problem into a higher dimension. Throughout this work we will use the terms *lifting* and *multiple shooting* interchangeably. Multiple shooting is also motivated by a range of additional objectives. Mattheij and Staarink [6] suggest three main criteria for an optimal lifting. First, a reduced amount of work for the automatic integrator. Second, numerical stability, by bounding the growth of rounding errors within a shooting interval. Third, the amount of memory required. Their work was later extended by Geiger [7].
In this work, we focus on another property of multiple shooting: the convergence of Newton-type methods. This effect was described in [8], where first steps towards the explanation of this phenomenon were taken. For boundary-value problems, constructive algorithms that aim to minimize local residual contraction in each Newton step were derived in [9]. Here, we address the open problem of deriving algorithms that adapt the number of multiple shooting intervals to accelerate the convergence of NLP solvers, such as `IPOPT` [10] and `blockSQP2` [11, 12], for optimal control problems.

1.1 | Outline and contributions

We start with a short overview of direct multiple shooting for optimal control problems and related theoretical results in Section 2. Section 3 contains a new collection of 40 benchmark problems for which we investigate and analyze the effect of lifting on the number of iterations for `IPOPT` and `blockSQP2`. Inspired by the previous section's findings, Section 4 proposes algorithms for better initialization of intermediate lifting points and for switching from multiple to single shooting. Finally, we present an extensive numerical study to showcase the effectiveness of our algorithms in Section 5. We conclude with a discussion in Section 6.

2 | Direct multiple shooting

In this section, we first formulate a template continuous optimal control problem in 2.1 before moving to its discretized formulation in 2.2. In 2.3 we briefly address sequential quadratic programming and condensing. Afterwards, 2.4 introduces common globalization techniques. In 2.5 we introduce the concept of the (Quasi-) Newton path before concluding the section by addressing aspects of real-time performance in 2.6.

2.1 | Lifted optimal control problems

We are concerned with problems of the following type

$$\begin{aligned}
\min_{u} \quad & \int_{t=0}^{T} \mathcal{L}(x(t), u(t))\,\mathrm{d}t + \Phi(x(T)) \\
\text{s.t.} \quad & x(0) = x_0 \in X_0 && && \text{(initial values)} \\
& \dot{x}(t) = f(x(t), u(t)), && t \in [0, T] && \text{(ODE)} \\
& \mathrm{lb}_g \le g(x(t), u(t)) \le \mathrm{ub}_g, && t \in [0, T] && \text{(bounds)}
\end{aligned}$$

Here the objective consists of the Mayer term Φ and the Lagrange term $\int \mathcal{L}(\cdots)\mathrm{d}t$, $x_0 \in X_0 \subset \mathbb{R}^{n_y}$ is the initial value, $u : [0, T] \to \mathbb{R}^{n_u}$ the control function, f defines the dynamics, and g the mixed constraints. We introduce intermediate variables $\mathbf{s}_1, \dots, \mathbf{s}_n$ at n distinct ascending time points $\{t_i\}_{i=1}^{n}$, together with $\mathbf{s}_0 = x_0$. Let y_i be the solution of the initial value problem (IVP)

$$x(t_i) = \mathbf{s}_i, \quad \dot{x}(t) = f(x(t), u(t)) \quad \forall t \in [t_i, t_{i+1}]. \tag{1}$$

To ease the notation we omit the explicit dependency of y_i on the initial value $\mathbf{s}_i$ and the control u on the interval $[t_i, t_{i+1}]$, as they are encoded by the index i. This leads to the following lifted problem formulation:

$$\begin{aligned}
\min_{\mathbf{s}_0, \dots, \mathbf{s}_n, u} \quad & \sum_{i=1}^{n} \left(\int_{t=t_{i-1}}^{t_i} \mathcal{L}(y_{i-1}(t), u(t))\,\mathrm{d}t \right) + \Phi(y_n(T)) && && (2) \\
\text{s.t.} \quad & \mathbf{s}_0 = x_0 \in X_0 && && \text{(initial values)} \\
& \mathbf{s}_i = y_{i-1}(t_i), && \forall i \in [n] && \text{(matching condition)} \\
& \mathrm{lb}_g \le g(y_{i-1}(t), u(t)) \le \mathrm{ub}_g, \quad t \in [t_{i-1}, t_i], && \forall i \in [n]. && \text{(bounds)}
\end{aligned}$$

Optimal Experimental Design (OED) problems [13, 14, 15, 16] are a special case of optimal control problems [17]. If the variational differential equations and description of the Fisher Information Matrix are included in an augmented right

hand side, OED problems can also be written in the lifted form (2). They often have specific properties, such as optimal *bang-bang* structures [17, 18]. For all OED problems considered in the following we make use of the so-called A-criterion, one of multiple design criteria which turns the matrix-valued covariance of the parameter estimates into a scalar objective that we want to minimize. More precisely, it is defined as the trace of the inverse Fisher information matrix F, i.e, $\mathrm{tr}(F^{-1})$. Further widespread choices are the D-criterion, defined as $\det(F^{-1})$, or the E-criterion, considering the largest eigenvalue of F^{-1}. Here we will limit ourselves to the A-criterion and refer to [15] for a comparative discussion.

2.2 | Discretization

To solve this problem, we are going to use the *first discretize, then optimize* approach. There are three parts for which we have to choose a discretization: the controls u, the constraints g, and the shooting points t_i. We choose those discretizations as follows:

- **Controls**: The continuous control function u is discretized as a piecewise constant function on a control grid.
- **States**: On every shooting interval the state trajectory is obtained as the numerical solution of the IVP (1), using the piecewise-constant controls on that interval and starting from the free shooting variable.
- **Constraints**: The mixed constraints g are only evaluated at the time points of a chosen constraint grid, on the numerically computed state and the piecewise-constant control at that time.

In general, these three discretizations are completely independent from one another. However, there are practical aspects that heavily influence the choice. It is advantageous to choose the shooting and constraint grids as subsets of the control grid. The reason for this lies in the structure of the resulting Hessian of the Lagrangian. In particular, the different shooting intervals are only coupled linearly through matching conditions, leading to an advantageous exploitable block structure. The only exception is the end of the time interval, where we do not add an additional control (as it would have no effect). Throughout our investigations we will compare the effect of the number of shooting points on the convergence speed. Thus, we want all other discretizations to be unaffected by the distribution of shooting points. Therefore, we choose the constraint grid to be equal to the control grid. To summarize, we choose one discretization with time points $0 = t_0, t_1, \dots, t_n = T$. Referring to the indices of these time points, we define three index sets:

$$\text{a control grid } I_{\text{control}}, \quad \text{a constraint grid } I_{\text{constr}}, \quad \text{and} \quad \text{a shooting grid } I_{\text{shoot}}.$$

These sets satisfy

$$\{0\} \subseteq I_{\text{shoot}} \subseteq I_{\text{constr}} = I_{\text{control}} \cup \{n\} = \{0, \dots, n\}.$$

The index 0 is always contained in I_{shoot}, as it represents the start of the first shooting interval where $\mathbf{s}_0$ is introduced. However, $\mathbf{s}_0$ is fixed to the initial value x_0 and there is no matching condition that can be violated introduced at $t_0 = 0$. For this reason we do not count $t_0 = 0$ towards the number m of shooting points (e.g., in the notation MS-m), but keep the index 0 in I_{shoot} to simplify the notation later on.
Equidistant shooting point discretizations with m shooting points are abbreviated by MS-m (and always include the final time point $t_n = T$ for $m > 0$). We denote the discretized controls by $\mathbf{q}_i$ and the discretized constraint functions by $\mathbf{g}_i$, i.e., $\mathbf{g}_i := g(y(t_i), \mathbf{q}_i)$ evaluated at the i-th constraint grid point. The potential constraints for the initial value $\mathbf{s}_0$ are included in $\mathbf{g}_0$. To group these discretizations according to their corresponding shooting interval, we write

$$I_{\text{shoot}} = \{0 = j_0, j_1, \dots, j_m\}$$

and summarize them as

$$\mathbf{g}_{[j_k]} := \begin{cases} (\mathbf{g}_i)_{i=j_k}^{j_{k+1}-1} & \text{if } k < m \\ (\mathbf{g}_i)_{i=j_k}^{n} & \text{if } k = m \end{cases}, \qquad \mathbf{q}_{[j_k]} := \begin{cases} (\mathbf{q}_i)_{i=j_k}^{j_{k+1}-1} & \text{if } k < m \\ (\mathbf{q}_i)_{i=j_k}^{n-1} & \text{if } k = m \end{cases}.$$

The same notation is used for the corresponding lower and upper bounds $\mathrm{lb}_{\mathbf{g}_{[j_k]}}$ and $\mathrm{ub}_{\mathbf{g}_{[j_k]}}$. Sometimes we want to refer to all discretizations. For this purpose we summarize $\mathbf{g}_{[j_0]}, \dots, \mathbf{g}_{[j_m]}$ as $\mathbf{g}$, $\mathbf{q}_{[j_0]}, \dots, \mathbf{q}_{[j_m]}$ as $\mathbf{q}$, and $\mathbf{s}_{j_0}, \dots, \mathbf{s}_{j_m}$ as $\mathbf{s}$. Usually, the matching conditions are separated from the other bounds, but for measuring the infeasibility of the whole problem it may be useful to summarize all constraints. For this purpose we define $\hat{\mathbf{g}}$ as the function including $\mathbf{g}$ and all matching conditions.
Analogously to (1), we define $y_{[j_k]}$ as the solution of the discretized version of the IVP, with the difference that the control $u(t)$ is replaced by the matching part of $\mathbf{q}_{[j_k]}$. With this notation, the *local contribution to the objective function* on shooting

interval k is

$$h_{[j_k]}(\mathbf{s}_{j_k}, \mathbf{q}_{[j_k]}) := \int_{t_{j_k}}^{t_{j_{k+1}}} \mathcal{L}(y_{[j_k]}(t), \mathbf{q}_{[j_k]}(t))\,\mathrm{d}t, \quad k < m,$$

i.e., the discretized analogue of the Lagrange term in (2) restricted to interval k (numerically approximated by the same scheme used to integrate (1)), with the Mayer term $\Phi(y_{[j_m]}(T))$ added for $k = m$. The summed up total objective is then written as $h = \sum_{k=0}^{m} h_{[j_k]}$.
Putting everything together, the discretized counterpart of the lifted problem (2) reads

$$\begin{aligned}
&\min_{\mathbf{s},\mathbf{q}} \sum_{k=0}^{m} h_{[j_k]}(\mathbf{s}_{j_k}, \mathbf{q}_{[j_k]}) && && (3)\\
&\text{s.t.} \quad \mathbf{s}_0 = x_0 && && \text{(initial values)}\\
&\qquad \mathbf{s}_{j_{k+1}} = y_{[j_k]}(t_{j_{k+1}}), && \forall k = 0, \dots, m-1 && \text{(matching conditions)}\\
&\qquad \mathrm{lb}_{\mathbf{g}_{[j_k]}} \le \mathbf{g}_{[j_k]}(\mathbf{s}_{j_k}, \mathbf{q}_{[j_k]}) \le \mathrm{ub}_{\mathbf{g}_{[j_k]}}, && \forall k = 0, \dots, m && \text{(bounds)}
\end{aligned}$$

where the optimization variables are the finitely many shooting variables $\mathbf{s} = (\mathbf{s}_{j_0}, \dots, \mathbf{s}_{j_m})$ and discretized controls $\mathbf{q} = (\mathbf{q}_{[j_0]}, \dots, \mathbf{q}_{[j_m]})$.
To measure the violation of the bounds on a shooting interval we define

$$v_{[j_k]}(\mathbf{s}_{j_k}, \mathbf{q}_{j_k}) = \left\| (\mathbf{g}_{[j_k]}(\mathbf{s}_{j_k}) - \mathrm{ub}_{\mathbf{g}_{[j_k]}})^+ + (\mathrm{lb}_{\mathbf{g}_{[j_k]}} - \mathbf{g}_{[j_k]}(\mathbf{s}_{j_k}))^+ \right\|_p$$

where $(\cdot)^+$ denotes the positive part and p represents the chosen norm. Unless stated otherwise, $\|\cdot\|$ always stands for the Euclidean norm.

2.2.1 | FSInit

The introduction of shooting variables comes with the additional freedom of choosing their initialization. One straightforward possibility of initializing the $\mathbf{s}_{j_k}$ is to evaluate the ODE (1) (its discretized version) over the entire time interval starting at $\mathbf{s}_0$ and to set $\mathbf{s}_{j_{k+1}} = y_{[j_k]}(t_{j_{k+1}})$. Put simply, we compute one forward simulation and use it to initialize all variables along one trajectory, without any violations of the matching conditions. We refer to this operation as *Forward Sweep Initialization (FSInit)* and will use it repeatedly throughout Section 4 below.

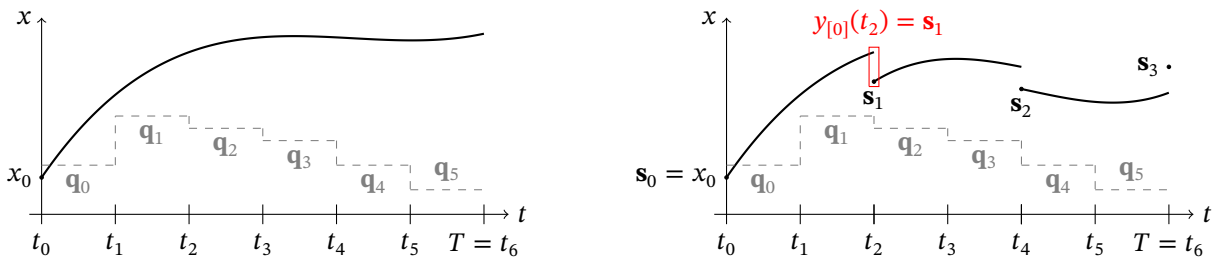

FIGURE 1 | Illustration of direct multiple shooting with $I_{\text{control}} = \{0, \dots, 5\}$. The left plot shows the single shooting version with $I_{\text{shoot}} = \{0\}$, whereas the right one shows a lifted version with $I_{\text{shoot}} = \{0, 2, 4, 6\}$. For the bottom discretization we would get $\mathbf{q}_{[0]} = (\mathbf{q}_0, \mathbf{q}_1)$, $\mathbf{q}_{[2]} = (\mathbf{q}_2, \mathbf{q}_3)$, $\mathbf{q}_{[4]} = (\mathbf{q}_4, \mathbf{q}_5)$, and $\mathbf{q}_{[6]} = ()$. The same pattern applies to the constraints, with the difference being $\mathbf{g}_{[6]} = \mathbf{g}_6$.

2.3 | Solution via Newton-type methods and condensing

We want to solve the discretized version of (2) using Quasi-Newton methods applied to the (perturbed) KKT conditions, compare [19, Chapter 18 and 19]. Let B be the exact or an approximation of the Hessian of h with respect to $(\mathbf{s}, \mathbf{q})$. This leads to a subproblem in $\mathbf{s}$ and $\mathbf{q}$. As the solution, we obtain step directions $d_{\mathbf{s}}$ and $d_{\mathbf{q}}$. We update the variables using these step directions and repeat this process until the convergence criteria are satisfied. Apart from the total constraint violation, this includes the optimality error (see [10] for `IPOPT` and [11, 12] for `blockSQP2`), denoted by opt. Because of the linear coupling in the matching conditions of the shooting points the Hessian exhibits a special block structure. This

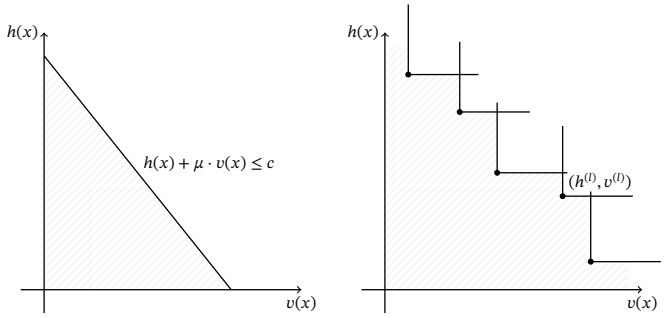

FIGURE 2 | Illustration of penalty methods (left) and filter methods (right). The hatched region shows which points have a better value with respect to the penalty function or are accepted by the filter.

allows for a technique called condensing (cf. [20]). Instead of computing a step for the original problem, we *condense* it to compute a step only in the original variable $\mathbf{s}_0$ and the controls $\mathbf{q}$. The remaining $d_{\mathbf{s}_1}, \dots, d_{\mathbf{s}_n}$ and the Lagrange multipliers corresponding to the matching conditions can then be reconstructed from the QP solution. For the details we refer to the literature [2, 12, 20].

2.4 | Globalization

For the rest of this section, let us consider the general optimization problem

$$\min h(x) \quad \text{s.t. } g(x) \leq 0 \tag{4}$$

We define the violation of the constraints as $v(x) := \|g(x)^+\|$. During the solution process we have to consider both the objective as well as the constraints. The globalization is performed via line search. Given the step direction Δx obtained from the (condensed) Newton-type subproblem, we choose a step length $\alpha \in (0, 1]$ so that the resulting iterate $x + \alpha\Delta x$ achieves an improvement in a measure that combines objective and feasibility. We present two popular choices for this measure.

Merit functions combine both objectives into a single function. Typical merit functions are either augmented Lagrangians or penalty functions, the latter of which we want to succinctly introduce (cf. [19, Chapter 17]).

A *penalty* function adds a scaled norm of the constraint violation to the objective:

$$P(x, \mu) := h(x) + \mu \cdot v(x), \quad \mu > 0.$$

Common choices for the norm of the violation are the squared euclidean norm (quadratic penalty) or the ℓ_1 norm (non-smooth penalty). The drawback of this approach is the choice of the penalty parameter μ. It can be shown that the minimizer of the penalty function is equal to the feasible optimum if the penalty parameter μ is chosen large enough. In practice, however, high penalty parameters may lead to numerical problems.

The filter method was suggested by Fletcher and Leyffer [21]. It considers the objective and the constraint violation $v(x)$ as two separate aims. We briefly summarize the main idea. Let $(h^{(k)}, v^{(k)})$ denote the values of $h(\cdot)$ and $v(\cdot)$ evaluated at the point $x^{(k)}$.

Definition 1. A pair $(h^{(k)}, v^{(k)})$ is said to dominate another pair $(h^{(l)}, v^{(l)})$ if both $h^{(k)} \leq h^{(l)}$ and $v^{(k)} \leq v^{(l)}$.

This means that $x^{(k)}$ is better than or equally as good as $x^{(l)}$ with respect to both the objective and the feasibility. Using this concept we can now define a filter.

Definition 2. A filter is a list of pairs $(h^{(l)}, v^{(l)})$ such that no pair dominates any other. A point $(h^{(k)}, v^{(k)})$ is said to be acceptable for inclusion in the filter if it is not dominated by any point in the filter.

In practice, a small *envelope* is added around the filter to prevent cycling. A newly computed iterate $x^{(k+1)}$ is then only accepted as an *improvement* in case it is acceptable for inclusion in the filter.

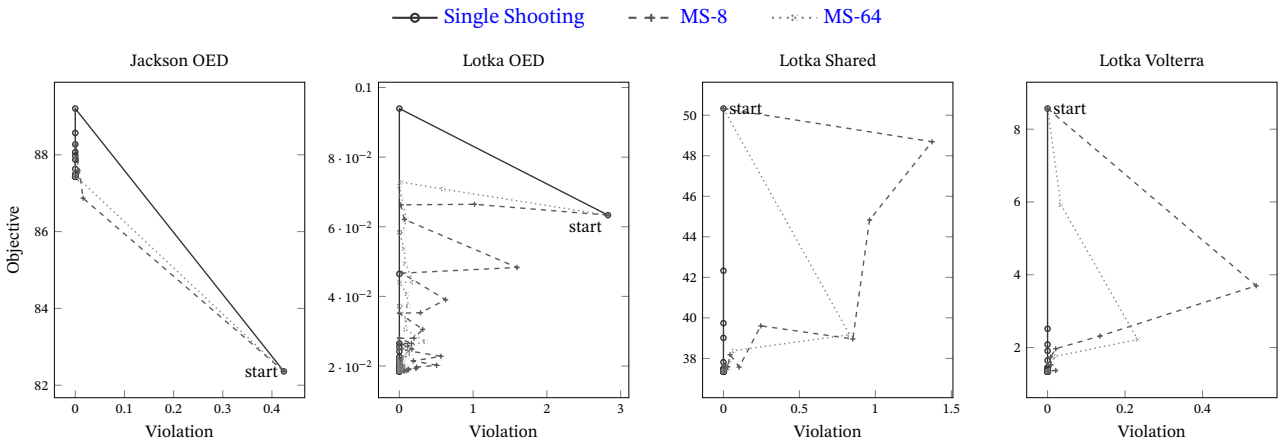

FIGURE 3 | Visualization of the paths taken by `blockSQP2` using SR1/BFGS Hessians. The control and constraint discretizations are identical for all shooting grids. Generally, we observe that few lifting points lead to larger violations of the matching conditions, since the linearizations of these conditions become less accurate for larger intervals. We notice that close to the solution the trajectory oscillates between improving the objective and reducing the violation of the matching conditions. Especially the multiple shooting formulations of the two OED problems exhibit very slow convergence close to the solution.

2.5 | The (Quasi-) Newton path

We refer to the sequence of iterates produced by the solver as *(Quasi-) Newton path*, depending on whether we use exact or approximate Hessians. A rigorous introduction to the concept of the Newton path can be found in [22]. The theoretical investigation of this path allowed us to gain further insights for BVP in [9]. For BVP, it turned out that the *theoretical Newton path* (obtained by using infinitely small step size control) of the original variables is not affected by lifting. Furthermore, we saw that lifting at infinitely many points may lead to the same Newton path as not lifting at all.

Due to the fundamental structural differences between optimal control and boundary value problems, these results do not carry over. More precisely, the matching constraints for optimal control problems also introduce new Lagrange multipliers, upon which the (Quasi-) Newton path depends. For BVP, on the other hand, lifting only increased the dimension of the root finding problem.

Since the controls and states make the problem very high-dimensional we cannot visualize the path directly. Instead, we opt to show the sequence of objective values and norms of constraint violations for the iterates to assert whether two liftings take a similar path. We compare these paths for some select problems in Figure 3.

Beyond the simple observation that paths can be similar or quite different, two structural patterns are worth pointing out. During the first couple of iterations the violations of the matching conditions are quite large, but expectedly decrease quickly over the number of iterations. Therefore, once we are close to the solution, all matching conditions are almost satisfied. However, we observe that the solution trajectory oscillates between improving the objective and reducing the violation of the matching conditions. For small problems such as Lotka Shared or Lotka Volterra this effect is quite minor. However, for the two OED problems, which are known to suffer from poor local conditioning of the A-criterion [23], this drastically slows down the convergence speed.

2.6 | Aspects of real-time performance

The real-word computational time required to compute a single Newton-type step for optimal control problems depends on two main factors. First, there is the time to solve the resulting system of linear equations. Naturally, lifting introduces more variables and therefore increases the size of the linear system. However, this does not directly imply that the lifted linear system of equations is harder to solve. The resulting system is highly structured and techniques like condensing can reduce the system to the size of the one for single shooting. Second, there is the time required to compute the gradient of the objective and constraint function and, if necessary, the exact Hessian. Since the first factor heavily depends on the chosen linear solver, we focus on the last factor and how we can influence it by means of lifting.

In our numerical experiments we use the automatic differentiation framework `CasADi` [24]. We observed that multiple shooting leads to massive speedup in terms of the cost per Newton-type iteration, as exemplified in Figure 4. As described in [25], there are two main reasons for this effect.

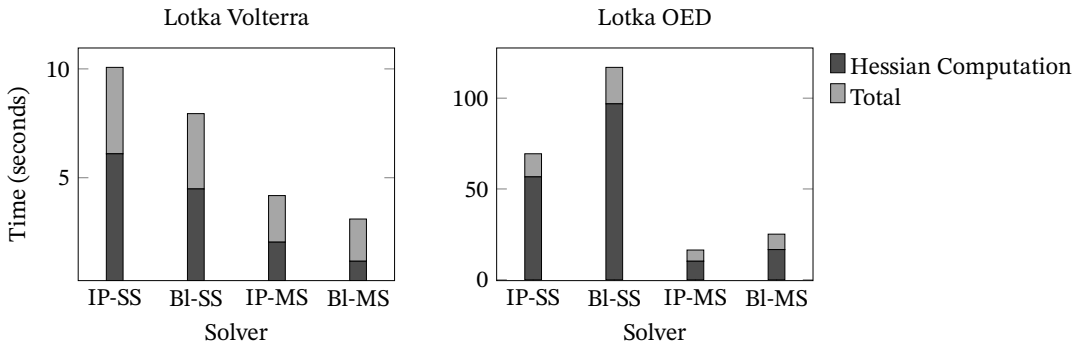

FIGURE 4 | Visualization of the real-time required to solve the Lotka Volterra problem (11 iterations for `IPOPT`, 12 for `blockSQP2`) and to perform 10 iterations for the Lotka OED problem. We compare `IPOPT` (IP) and `blockSQP2` (Bl) for both single shooting (SS) and multiple shooting (MS), using MS-64 with the same control discretizations. The darker regions illustrate the fraction of the time used for computing the exact Hessian. Multiple shooting drastically decreases the time required to compute the Hessian. This discrepancy increases for problems with many differential states and controls (such as Lotka OED).

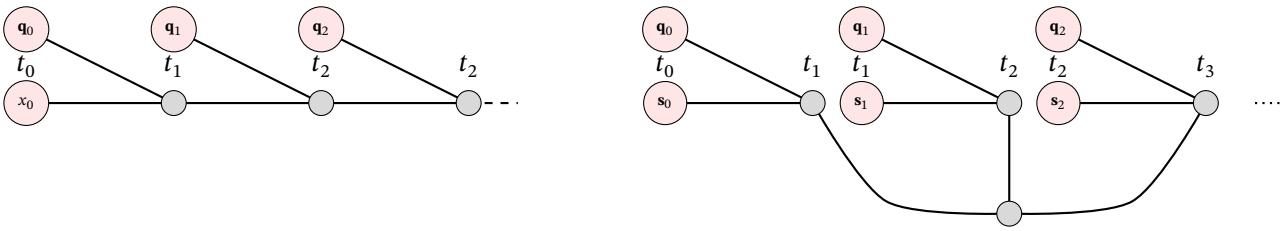

FIGURE 5 | Visualization of the graphs used by the algorithmic differentiation framework for single shooting (left) and multiple shooting (right). The shooting intervals in the bottom graph all contribute to the summed up objective term (2).

First, the total number of non-zero derivatives may even decrease due to lifting. The reason is that the state now only depends on the controls in the current shooting interval, not on all previous controls. On the other hand, the single shooting derivatives have to be calculated across the entire time interval, possibly making it a lot more expensive, see Figure 5.
Second, the linear coupling allows for parallelization across the shooting intervals. Hence, the derivative computation and the evaluation of both objective and constraints can be parallelized across the shooting intervals, saving further time in the process. This parallelization was also used in Figure 4.

3 | A first comparison

A first small-scale quantification of the impact of lifting on the number of SQP iterations was undertaken in [26]. To get a more general overview, we perform a numerical study for a greatly expanded set of optimal control problems taken from the open benchmark library for optimal control problems mintOC [27]. It currently encompasses more than 80 problems, ranging from one-dimensional to more than twenty-dimensional examples and from quadratic tracking problems to OED problems. Every Lagrange term objective can be rewritten as a Mayer objective. For this reason we also compare different formulations of some problems, wherever practical.
In our numerical study we examine both the interior-point solver `IPOPT` and the structure-exploiting SQP solver `blockSQP2`. For consistency, we use a fixed-step-size Runge–Kutta 4 method and keep the control and constraint discretizations with 64 equidistant intervals constant across all choices of lifting points. We always use an equidistant distribution of shooting nodes. If a problem requires more than 200 iterations, or if the solver itself fails, we consider the solution to have failed.
All computations were performed on an AMD Ryzen 7 4800H with 16GB of RAM running Ubuntu 24. Regarding software, we used Python 3.13.5, CasADi 3.7.0 [24], NumPy 2.3.1 [28], and SciPy 1.16.0 [29]. The code is open source and published on GitHub and Zenodo [30].
We show the number of iterations across all considered problems and shooting grids in Figure 1 and the corresponding computational times in Figure 2, both placed in the Appendix for space reasons. We summarize the results in Figure 6 and perform a more detailed analysis in the following subsections.

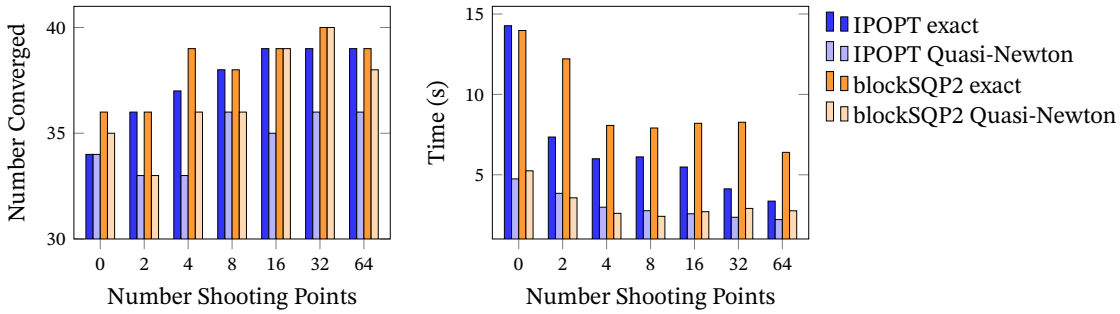

FIGURE 6 | On the left we show the number of problems that converged for a given solver configuration and shooting grid. The right image displays the average time required to solve a problem. In the latter plot, the average is taken over the problems that converged across all shooting grids for the given solver configuration. Generally, the finest the shooting grid exhibits the most robust convergence and the shortest required real-time. However, for MS-64 and blockSQP2 the *"Egerstedt"* problems no longer converge within 200 iterations.

3.1 | Exact Hessian methods

First, let us address the performance of `IPOPT`. Generally, we observe that increasing the number of lifting points has little effect on the number of required iterations. There is no benchmark problem for which the number of iterations monotonically increases with the number of lifting points. In contrast, there are a couple of problems for which lifting clearly reduces the number of iterations or enables convergence in the first place. Among these are problems such as *"Cart Pendulum"*, *"Ducted Fan"*, *"Moon Landing"*, and *"Van der Pol Mayer"*. This can be attributed to the decreased nonlinearity or the possibility to escape infeasible regions more easily (e.g. for *"Van der Pol"*). When considering the real-time performance, lifting at all control discretizations is indisputably the best choice across all problems.
Next, we want to discuss the performance of `blockSQP2` for exact Hessians. As with `IPOPT`, many problems do not show a clear trend when it comes to the number of iterations. As shown in Figure 6, the finer shooting grids MS-16, MS-32, and MS-64 exhibit more robust convergence than single shooting or the coarser grids. For simple problems, such as *"Batch Reactor"* or *"Bioreactor"*, lifting slightly increases the number of iterations. The reasons are extra steps which are needed to satisfy the matching conditions to sufficient accuracy. Considering the computational times, the finest shooting discretization constitutes the fastest choice for the overwhelming majority of problems.
Nevertheless, there are problems for which more lifting points increase the number of iterations, even offsetting the real-time benefits of faster multiple shooting iterations. Among these are *"Egerstedt"*, *"Egerstedt Mayer"*, and select OED problems like *"Van der Pol OED"*.

3.2 | Quasi-Newton methods

The Quasi-Newton version of `IPOPT` does not make use of block-wise Hessian approximations. Therefore, we cannot expect the quality of the Hessian approximations to improve by introducing more lifting points. Regarding the influence of lifting on the number of required iterations and the real-time performance, it mostly behaves similarly to its exact Hessian counterpart. One notable exception is the *"Electric Car"* and *"Electric Car Mayer"* problem, where it performs a lot worse for few shooting intervals.
Unlike the exact Hessian methods, for most problems the Quasi-Newton version of `blockSQP2` exhibits a clear trend of improved performance with more lifting points. These performance gains can be observed for problems which showed no clear improvements for the exact Hessian version of `blockSQP2`, like *"Lotka Competitive / Shared / Volterra"*. This suggests that the improvements stem from better Hessian approximations caused by the higher-dimensional block-wise updates.
Quasi-Newton methods update Hessian approximation based on local secant information along the iterates. For single shooting the accumulated sensitivities over the entire time interval can lead to curvature information that cannot be captured accurately. When using multiple shooting, the curvature can be captured more accurately on a per-interval basis. To validate this hypothesis, we have to examine how well the Quasi-Newton Hessian approximates the real one on the null space of the constraints. Therefore, we define the following matrix, which projects onto the null space of the current Jacobian matrix of the constraints $A^{(k)}$:

$$P^{(k)} := I - (A^{(k)})^T \left(A^{(k)}(A^{(k)})^T\right)^{-1} A^{(k)}.$$

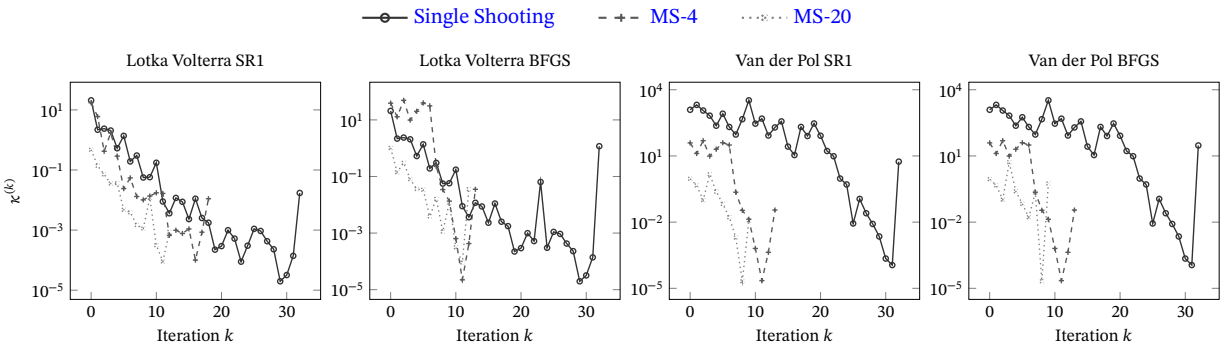

FIGURE 7 | Estimated values for $\kappa^{(k)}$ from (5). The control and constraint discretizations are the same as MS-20. We observe that both the SR1 and the BFGS approximation become more accurate the finer we choose the shooting grid. (The increased $\kappa^{(k)}$ value in the final iteration can be attributed to numerical errors due to the small step size.)

The better the approximation of the real Hessian the better the local convergence behavior in the vicinity of the solution. Following Nocedal [19, Theorem 18.5], let $B^{(k)}$ denote the approximate and $\nabla_x^2 \mathcal{L}^{(k)}$ the exact Hessian in iteration k. We compute the approximation error in iteration k as

$$\kappa^{(k)} := \frac{\|P^{(k)}(B^{(k)} - \nabla_x^2 \mathcal{L}^{(k)})(x^{(k+1)} - x^{(k)})\|}{\|x^{(k+1)} - x^{(k)}\|} \tag{5}$$

and display the computed values for $\kappa^{(k)}$ in Figure 7.
The real-time advantages of multiple shooting carry over and add to the better Hessian approximations. Again, this makes the finest shooting grid the best choice for most problems, with the same exception of *"Egerstedt"*, *"Egerstedt Mayer"*, and select OED problems like *"Lotka OED"* as for the exact Hessian variant.
The slow convergence and instability of Quasi-Newton SQP methods for OED were already observed in [23]. There, the authors suggest replacing the A-criterion by applying the transformation $x \longmapsto -x^2$ and demonstrate improved convergence.

4 | Algorithms

The results from Section 3 heavily favor multiple shooting with a shooting grid that is equal to the control grid. However, there are problems exhibiting worse performance for these fine shooting grids. These problems motivate us to devise algorithms which address these issues.
In [9] we considered lifting algorithms for boundary value problems. The core of these algorithms was the observation that lifting, starting with FSInit (recall Subsection 2.2), does not affect the theoretical Newton path. This enabled us to determine the lifting points with the best residual contraction in every iteration by means of dynamic programming. For optimization problems, which contain the continuity conditions for the lifted variables as additional constraints, there are several technical and practical difficulties:

- Optimal control problems are typically highly nonlinear. Far away from the optimal solution the local contraction varies strongly, as shown in Figure 8. This makes approaches based on local contraction estimates, as derived in [9], impractical. Moreover, optimal control problems often require step size control for the Newton-type iteration itself (i.e., a line search reducing the step length α below 1, as introduced in Subsection 2.4, not the time step of the ODE integrator), which further impedes local convergence analysis.
- Multiple shooting speeds up the computation (see Section 2.6) and for Quasi-Newton methods it yields better approximations (compare Figure 7). These aspects have to be balanced with the effect that lifting has on the local *curvature* of the function and thus the number of required iterations. In addition, aspects like the barrier parameter in `IPOPT` or heuristics like inertia serve to aggravate the theoretical analysis.

The main approaches for adaptive lifting of BVP are based on

- local residual contraction [9],

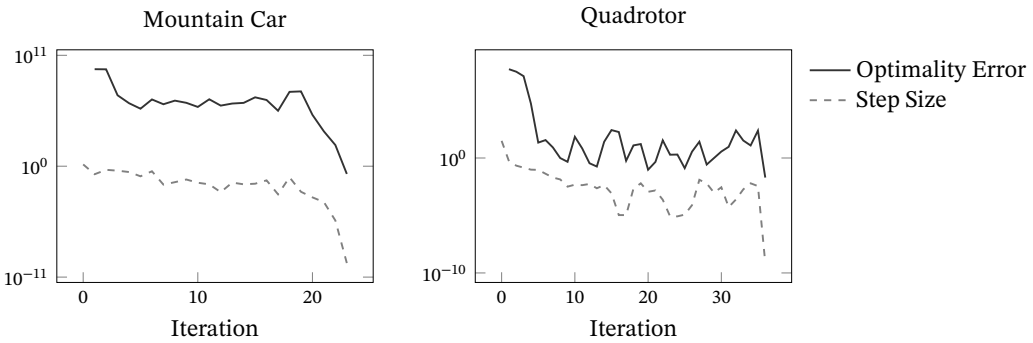

FIGURE 8 | Step sizes and optimality for the "Mountain Car" and "Quadrotor" problems solved using `blockSQP2` (Quasi-Newton, MS-64). We cannot expect monotonic convergence in either the optimality measure or the step sizes.

- bounding sensitivities [6, 7] or
- removing lifting points based on matching violation [7].

Currently, `IPOPT` does not provide the callback functionality to change the state values during the optimization process. Hence, the following algorithms focus on `blockSQP2`, unless stated otherwise.
In 4.1 we will first adapt the approach based on local residual contraction to make it suitable for our scenario. Building on the insights from 2.5 and 4.1, we devise an algorithm for automatic FSInit in Subsection 4.2. In 4.3 we give attention to the sensitivity-bounding approach. Penultimately, we will describe a modification of the residual-based thinning-out strategy for both `blockSQP2` and `IPOPT` in 4.4 and 4.5. Finally, we describe a combination of the previously introduced algorithms in 4.6.
To ease the notation, we will sometimes summarize the primal states, i.e., $\mathbf{s}$ and $\mathbf{q}$, as x.

4.1 | Custom initialization with merit function

As examined in [9], custom initialization has great potential to speed up the optimization process. Taking a step back, the residual-based lifting algorithms did nothing else than to choose better initializations (which resulted from the Newton step) for the lifted variables. To this end we have to derive a theoretical foundation regarding which initialization is better. The initial guess for a start point usually lies far away from the optimal solution. Multiple shooting introduces additional degrees of freedom by allowing us to choose custom initial values of the states at the start of the shooting intervals. Naturally, we want to make use of this freedom to find a better initialization. Consequently, we look for a criterion to measure the quality of a faraway start point. The optimality error and the augmented Lagrangian heavily depend on the initial choice of the Lagrange multipliers. At the same time, numerical experiments suggest that the initial choice of these multipliers typically has little to no influence on the overall convergence behavior.
Therefore, let us consider the globalization criteria from Section 2.4. For filter methods, it is not straightforward to derive a single objective from its two components. This leaves the merit or ℓ_1 penalty function, which only requires an upper bound on the maximum norm of the Lagrange multipliers. Moreover, the objective value function and the constraint functions on the different multiple shooting intervals are decoupled. Therefore, we can apply a similar approach to [9], using dynamic programming. The *weight* for choosing a certain initial value for a shooting interval $[t_{[j_k]}, t_{[j_{k+1}]}]$ is given by $h_{[j_k]} + \mu \cdot v_{[j_k]}$, i.e., the penalty for the contribution to the objective and constraint violations of the current interval, plus μ times the resulting violation of the matching conditions. For this algorithm we only have to provide sets $\mathcal{S}_k = \mathcal{S}_k^{(\text{old})} \cup \mathcal{S}_k^{(\text{new})}$, where $\mathcal{S}_k^{(\text{new})}$ are the newly introduced candidates at time t_{j_k} and $\mathcal{S}_k^{(\text{old})}$ are the states computed from $\mathcal{S}_{k-1}$ via FSInit. We describe the resulting algorithm as *merit-based initialization via dynamic programming* and illustrate it in Figure 9.
Using the merit function introduces the scaling factor μ, raising the question of how to choose said parameter. Theory suggests that $\mu > \|\lambda\|_\infty$ for the Lagrange multipliers λ, compare [19]. Nevertheless, many variations are possible. Most notably, choosing $\mu = 0$ just corresponds to finding a initialization with better objective. This can be especially helpful for parameter estimation problems, where initialization with given measurements allows us to speed up convergence or even to avoid local minima.
On the other hand, ignoring the objective and only minimizing constraint violation (which is equivalent to choosing μ arbitrarily large) leads to an initial guess that is closer to the set of feasible solutions. This latter ansatz is particularly useful for interior point methods, which have to find a feasible interior start point. It may allow us to skip a potential

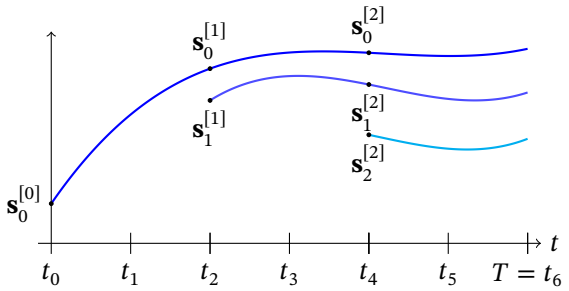

FIGURE 9 | Illustration of the merit-based initialization via dynamic programming. Here we have $\mathcal{S}_i^{(\text{new})} = \{\mathbf{s}_i^{[i]}\}$ and $\mathcal{S}_i^{(\text{old})} = \{\mathbf{s}_k^{[i]} \mid k = 0, \dots, i-1\}$, for $i = 0, 1, 2$.

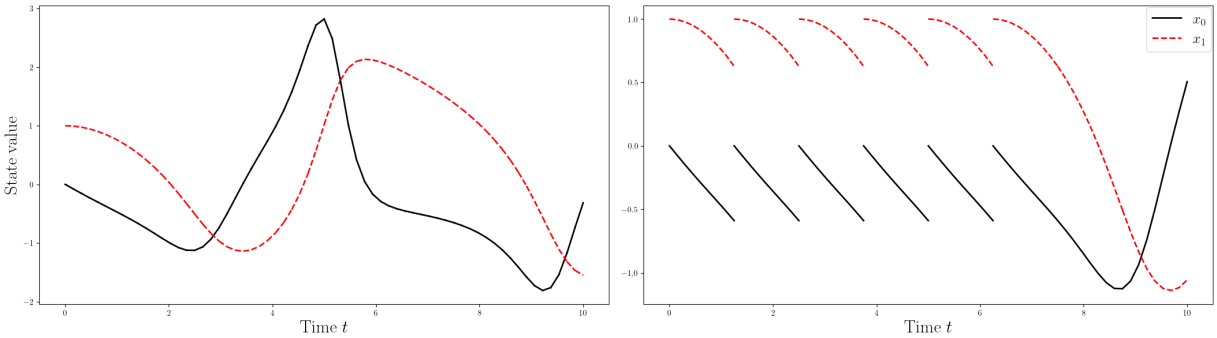

FIGURE 10 | Comparison of FSInit (left) and the custom initialization using merit-based initialization via dynamic programming (right) applied to the "Van der Pol" problem. We chose initial values (0, 1) as candidates for eight equidistant shooting intervals, $\mu = 1$, and a constant control of 0.5. Using `blockSQP2` with control and constraint discretizations as for MS-64, starting at the merit-based initialization converges after 10 iterations, compared to 19 iterations for FSInit.

restoration phase if the states determined by FSInit are infeasible. If the ODE cannot be evaluated on a certain shooting interval this can be incorporated by setting the corresponding weight to $+\infty$. One such example is described in [16].

Conceptually, the algorithm proceeds in two passes over the shooting intervals, much like a standard shortest-path dynamic program. In a *backward* pass, starting from the final interval m and working towards the first, it computes, for every candidate state $s^{[k]} \in \mathcal{S}_k$, the best achievable cost-to-go $J(s^{[k]})$. The local penalty $P_{[j_k]}(s^{[k]}) = h_{[j_k]}(s^{[k]}) + \mu \cdot v_{[j_k]}(s^{[k]})$ incurred on interval k, plus the cheapest way to continue from there, either by following FSInit to $s^{[k+1]} = y_{[j_k]}(t_{j_{k+1}}, s^{[k]})$ at no extra matching cost, or by jumping to one of the newly offered candidates $s' \in \mathcal{S}_{k+1}^{(\text{new})}$ at the price of an extra $\mu \|s^{[k+1]} - s'\|_1$ matching penalty. For each $s^{[k]}$ the cheaper of these two options is kept, together with a pointer $\pi(s^{[k]})$ to the chosen successor. Once this backward sweep reaches interval 0, a *forward* pass simply selects the cheapest starting candidate $\mathbf{s}_0$ and follows the stored pointers π forward to recover the full sequence $(\mathbf{s}_0, \dots, \mathbf{s}_{j_m})$. The full pseudocode is given as Algorithm 4 in the Appendix.

Commonly, the initial controls are chosen to be constant over the time interval. Additionally, without loss of generality, we assume the underlying ODE to be autonomous, i.e., have no explicit time dependence. If we introduce one possible custom initialization $\mathbf{s}_c$ at all lifting points, then we only have to compute the resulting trajectory once, starting at time 0. All other trajectories are then obtained by simply shifting this trajectory in time. Consequently, the cost of determining a custom initialization with respect to the merit function is greatly reduced. Figure 10 showcases the initialization computed via merit-based dynamic programming.

4.2 | Automatic FSInit

Algorithm 4 allows us to determine where the added violation of a matching condition is beneficial and where it is not. Instead of invoking this algorithm only once to determine better start points, it seems promising to perform it after every (Quasi-) Newton iteration. This, however, proved to be prohibitively expensive. Moreover, we observed that the algorithm usually just replaces all intermediate states using FSInit once the violations are small.

Based on this observation we examine a new algorithm that only checks whether replacing all states by FSInit improves the current iterate. Close to the solution we have to account for the change in the optimality error caused by the changed

states and make sure that the determined new points do not lie inside the filter. For very large violations, on the other hand, the trajectory resulting from FSInit may exhibit very strong oscillations or be very far away from the previous iterate. For both of these scenarios FSInit often leads to worse performance.
When using Quasi-Newton updates, large changes through FSInit can lead to further problems. To explain this, let $x^{(k)}$ be the previous iterate, $x^{(k+1)}$ be the new iterate, and $x_{\text{fs}}^{(k+1)}$ be the new iterate adapted by FSInit. Normally, we would update the old Hessian approximation $B^{(k)}$ along the computed step from $x^{(k)}$ to $x^{(k+1)}$. This may be a bad Hessian approximation at $x_{\text{fs}}^{(k+1)}$.
Consequently, we apply the automatic FSInit when the violations of the matching conditions are not too large, but only while we are not too close to the solution. Depending on how close the previous iterate was to the optimal solution, we replace the current shooting variables by FSInit if it improves the merit or the optimality error. To put it simply, after every (Quasi-) Newton step from $x^{(k)}$ to $x^{(k+1)}$ we check whether it is beneficial to apply FSInit to the newly computed states. If we choose to do so, we obtain $x_{\text{fs}}^{(k+1)}$. For Quasi-Newton methods we then update the Hessian along the adapted step from $x^{(k)}$ to $x_{\text{fs}}^{(k+1)}$. The general idea is described in Algorithm 1.
To perform as few additional evaluations as possible we choose Line 1 in Algorithm 1 as

$$\|\hat{\mathbf{g}}(x^{(k+1)})\|_\infty \leq 0.1 \quad \text{and} \quad \text{opt}(x^{(k)}) > 0.005.$$

Both these values are fixed absolute thresholds used across the entire benchmark set, rather than being tuned per problem. The value 0.1 bounds the infeasibility of the matching and bound constraints at which FSInit is still considered numerically trustworthy, whereas 0.005 is a generic, deliberately loose cutoff for "not yet converged" in terms of the optimality error reported by the solver ([11, 12]), which is itself already normalized by the solver. We found this single pair of thresholds to work robustly across all 40 benchmark problems without further tuning, but we do not claim that either value is individually optimal. Still being far away from the optimum, i.e., Line 3 of Algorithm 1, is quantified by

$$\text{opt}(x^{(k)}, \lambda^{(k)}) > 0.1,$$

i.e., by a second, larger optimality threshold than the one in Line 1. Together, these two thresholds split the "not too close" regime selected by the outer condition into two cases. If we are far away (Line 3), we fall back to the ℓ_1-penalty merit comparison because the optimality error is not yet a reliable indicator of progress. Otherwise, if we are at a medium distance, we instead compare the optimality error directly. The final if clause in Line 10 is used if the problem is already close to the area of quadratic convergence with the goal that only few iterations have to be performed afterwards. For the high-dimensional OED problems, FSInit only rarely leads to an improvement in the optimality error, leading to many unnecessary evaluations of $\text{opt}(x_{\text{fs}}^{(k+1)}, \lambda^{(k+1)})$. For this reason we skip this part for OED problems.
Algorithm 1 may prevent the multiple shooting problem from taking a *bad* path by bringing it back to the Newton path of the single shooting problem if these paths are not too far from one another, see Section 2.5.

4.3 | Sensitivity-bounding approach

We proceed in a similar way as Geiger [7] proposed for boundary value problems. In comparison to boundary value problems, we have to take two additional aspects into account, the current objective as well as the current constraint violations. Under the assumption that the constraint functions are linear, we can focus on the differential states and the objective function. For the sake of simplicity we further assume that the objective is formulated via a Mayer term that linearly depends on the states at the end of the time interval. In this simple regime the straightforward idea would be to bound the sensitivities $\nabla_{(\mathbf{s}_{[j_k]}, \mathbf{q}_{[j_k]})} y_{[j_k]}(\mathbf{s}_{[j_k]}, \mathbf{q}_{[j_k]})$ of the states $y_{[j_k]}$ from Subsection 2.2.
While this approach seems enticing, it comes with several drawbacks. First, computing the sensitivities comes at a huge computational cost. Second, while bounded sensitivities may improve numerical stability during the integration, on their own they lack the theoretical justification to improve upon the convergence speed. In view of the plethora of possibilities to choose both norms and bound values, there is no obvious choice. Further problems arise with the different lengths of the resulting shooting intervals, which might hinder the ability to parallelize (e.g., if there is one rather long interval). When computing exact Hessians, these possibly longer intervals also counteract the performance gains achieved thanks to the decoupling on the short shooting intervals described in Section 3.

Algorithm 1 Automatic FSInit

Input:
$\lambda^{(k)}, \lambda^{(k+1)}, x^{(k)}, x^{(k+1)}$ ▷ dual and primal iterates
h ▷ objective function
$\hat{\mathbf{g}}$ ▷ matching conditions and bounds

if violation small and $x^{(k+1)}$ not too close to optimum then
  $x_{\text{fs}}^{(k+1)} \leftarrow \text{FSInit}(x^{(k+1)})$
  if $x^{(k)}$ far from optimum then ▷ consider the ℓ_1 penalty
    $v_{\text{fs}}, v \leftarrow \|\hat{\mathbf{g}}(x_{\text{fs}}^{(k+1)})\|_1, \|\hat{\mathbf{g}}(x^{(k+1)})\|_1$
    $h_{\text{fs}}, h \leftarrow h(x_{\text{fs}}^{(k+1)}), h(x^{(k+1)})$
    $\mu \leftarrow \|\lambda^{(k)}\|_\infty$
    if $h_{\text{fs}} + \mu \cdot v_{\text{fs}} < h + \mu \cdot v$ then
      $x^{(k+1)} \leftarrow x_{\text{fs}}^{(k+1)}$ ▷ replace with FSInit
    end if
  elseif $\text{opt}(x_{\text{fs}}^{(k+1)}, \lambda^{(k+1)}) < \text{opt}(x^{(k+1)}, \lambda^{(k+1)})$ ▷ consider the optimality error
    $x^{(k+1)} \leftarrow x_{\text{fs}}^{(k+1)}$ ▷ replace with FSInit
  end if
end if
return $x^{(k+1)}$

4.4 | Automatic condensing

Multiple shooting has advantages for start points that are far away from the solution. Either because the single shooting function is highly nonlinear or cannot even be evaluated. Just as we cannot predict the optimal solution or the required number of iterations in advance, we cannot make any a priori deductions about the paths that the different liftings may take. Once we are close to the solution, however, the added complexity and matching constraints of multiple shooting yield no real advantage. In this case, the matching conditions are usually satisfied up to a small error and satisfying them to the desired accuracy might even slow down convergence. The reason is that the solver's stopping criterion requires both the primal feasibility (including the matching conditions) and the dual optimality measure opt (which involves the Lagrange multipliers for those matching conditions) to fall below their respective tolerances. Choosing a finer shooting discretization makes the linearizations of the matching conditions more accurate. At the same time, the sensitivity of the objective function to any individual shooting variable becomes correspondingly smaller. This contrast between the matching conditions and the flat objective function (A-criterion) seems to be the source of ill-conditioning close to the solution for benchmarks such as the OED problems considered in Section 2.5. There, satisfying these additional matching conditions to the desired accuracy without deteriorating the small progress in the objective slows down convergence considerably. This also explains the oscillatory behavior of the solution trajectory mentioned in Section 2.5, suggesting the following approach: we start with a multiple shooting discretization and switch to single shooting once we are close to the solution and the matching conditions are all almost satisfied.

First, let us consider the case of exact Hessian matrices. Switching from multiple to single shooting once the matching conditions are almost satisfied removes the now-superfluous matching constraints and their multipliers, which can save a few iterations close to the solution for the reason discussed above. At the same time, each remaining single shooting iteration becomes more expensive, since evaluating the dense single shooting Hessian, gradient, and Jacobian over the whole time horizon is considerably more expensive than the block-structured, parallelizable evaluation used for multiple shooting (Subsection 2.6). Thus, there is a delicate trade-off between the iterations saved by switching to single shooting and the added per-iteration cost of single shooting compared with multiple shooting. The precise balance between these two effects heavily depends on such factors as the hardware being used, the way derivatives are calculated, and the specific solvers being applied.

Approximate Hessians add another layer of complexity to this problem. The higher-dimensional block-wise Quasi-Newton updates lead to better approximations of the exact Hessian than the dense updates for single shooting as discussed in 3.2. Once we switch from multiple to single shooting, we need an initial Hessian approximation for the lower-dimensional problem. One possibility would be to start over with the identity matrix. However, we would lose all the previously accumulated information. Therefore, we make use of the condensing algorithm and use the condensed Hessian of the multiple shooting problem as approximate Hessian for the single shooting problem.

The solver `blockSQP2` always maintains both SR1 and BFGS approximations. When switching from multiple to single shooting, we condense the SR1 approximation. Based on practical observations, we reset the BFGS approximation to the identity matrix. The reason for this is that the SR1 approximation can be shown to converge against the true Hessian, see [19, Chapter 6.2]. Thus, its condensed variant can be considered a good approximation to the true single shooting Hessian. For BFGS matrices, for which positive definiteness is always enforced, this is not necessarily the case.
We outline the general idea in Algorithm 2. From a practical point of view, this raises the question of when exactly we want to switch from multiple to single shooting. In other words, how we quantify being *close* to the solution with bad local contraction.
First, we have to assure that the matching conditions are almost satisfied and yield no real advantage. Second, we check whether we are close to a solution. Obviously, we do not have a priori knowledge about the solution. Hence, we enforce the norms of the steps computed by the Newton-type method to be small. To account for problems across all orders of magnitude we consider the relative step sizes.
Next, let us discuss our precise choices for Algorithm 2, starting with Quasi-Newton methods. To make sure that the current iterate x is close to the optimal solution we require the optimality and the relative error of the matching conditions to be small enough:

$$\text{opt}(x) < 5 \cdot 10^{-3}, \quad \varepsilon_{\text{match}} = \frac{\sqrt{\sum_{k=0}^{m-1} \|y_{[j_k]}(t_{j_{k+1}}) - \mathbf{s}_{j_{k+1}}\|^2}}{\varepsilon_{\text{m}} + \sqrt{\sum_{k=1}^{m} \|\mathbf{s}_{j_k}\|^2}} < 5 \cdot 10^{-3}, \tag{6}$$

where $\varepsilon_{\text{m}} = 10^{-16}$ just prevents division by zero. Finding suitable values that work across such a multitude of problems is not easy to do. One crucial point is that switching to single shooting too early may worsen the performance, since we lose the good high-rank Quasi-Newton updates. To ensure that the Hessian approximation does not change too drastically anymore, we add further constraints on the step sizes and the total norm of the constraint violation. However, these additional constraints are not enforced for the OED problems, as some may never reach step sizes that are small enough. For all other problems we consider the constraint violations, the relative step size, and the current step size:

$$\|\hat{\mathbf{g}}(x)\| < 10^{-3}, \quad \|\Delta x\|^2 < 10^{-3}, \quad \text{and} \quad \frac{\|\Delta x\|}{\|x\|} < 5 \cdot 10^{-2}. \tag{7}$$

Finally, there is no need to switch to single shooting if multiple shooting converges just fine, so we only switch if the optimality of the current iterate x is worse than that of the previous one. For a couple of problems slightly adjusting these values would improve the speedup via Algorithm 2 even further, but we found these choices to be the best trade-off across all problems.
Let us address the exact Hessian version of `blockSQP2`. Here, the difference in the number of iterations is generally way less pronounced. In turn, we do not expect the adaptive Algorithm 2 to have a significant impact for many problems. The additional requirements from Equation (7) prove to be too restrictive, since the exact Hessian leads to good performance very close to the solution and the switching to single shooting almost never occurs. Therefore, we will no longer consider (7) when using the exact Hessian.

Algorithm 2 Adaptive condensing

```
Input:
    x, λ                                  ▷ primal and dual multiple shooting variables
    B                                     ▷ (approximate) multiple shooting Hessian

if (x, λ) is close to the optimal solution then
    if local convergence is poor then
        B_cond ← condense B
        x_cond, λ_cond ← condense x, λ
    end if
end if
return x_cond, λ_cond, B_cond             ▷ continue with single shooting
```

4.5 | Condensing for IPOPT

IPOPT does not provide functionality to simply *condense* the Hessian, which limits us to the exact Hessian version, where this does not pose a problem. As investigated in Section 2.6, the finest shooting grid is usually the best choice. There are a few problems for which the number of iterations for MS-64 is slightly higher than for single shooting, e.g., *"Van der Pol OED"*. Notwithstanding, considering the real-time performance, we could not identify a problem for which switching to single shooting promises better performance.

4.6 | Combining all approaches

The algorithms proposed so far are not mutually exclusive. We propose the combined Algorithm 3, which switches to single shooting in case of bad convergence close to the solution or replaces the states by FSInit during the optimization process using Algorithm 1.
It would be straightforward and beneficial to include the merit-based initialization via dynamic programming in the combined algorithm. However, most of our problems do not benefit from this and the specific choice of the merit parameter μ would just create further ambiguity.

Algorithm 3 Combined lifting heuristic

Input:
$\mathcal{S}$ with $\mathcal{S}_0 = \{s_0\}$ ▷ candidate states
$h, \hat{\mathbf{g}}$ ▷ objective and constraint function
$\lambda^{(0)}$ ▷ initial dual variables
$B^{(0)}$ ▷ (approximate) multiple shooting Hessian

for $k = 0, 1, \dots$ do
  $x^{(k+1)}, \lambda^{(k+1)} \leftarrow$ perform step with $x^{(k)}, \lambda^{(k)}, B^{(k)}$
  if $x^{(k+1)}, \lambda^{(k+1)}$ satisfy criteria then
    return $x^{(k+1)}, \lambda^{(k+1)}$
  end if
  $B^{(k+1)} \leftarrow$ compute new (approx.) Hessian
  $x^{(k+1)}, \lambda^{(k+1)}, B^{k+1} \leftarrow$ Algorithm 2$(x^{(k+1)}, \lambda^{(k+1)}, B^{k+1})$
  $x^{(k+1)} \leftarrow$ Algorithm 1$(x^{(k)}, \lambda^{(k)}, x^{(k+1)}, \lambda^{(k+1)}, h, \hat{\mathbf{g}})$
end for

5 | Numerical experiments

We apply Algorithms 1, 2, and 3 to both the exact and Quasi-Newton versions of blockSQP2 on our benchmark set. The software and hardware setup is the same as in Section 3. We show the collected results in Figure 3, placed in the Appendix for space reasons. In this section, we provide a quantitative summary, while the discussion is deferred to Section 6.
Let us now quantify the improvement that Algorithm 3 brings compared with multiple shooting on the finest shooting discretization. The average number of required iterations for every solver is shown in Figure 11. Comparing MS-64 and Algorithm 3 directly, Algorithm 3 requires on average 26.8% fewer iterations for the exact Hessian and 27.1% fewer iterations for the Quasi-Newton variant.
Algorithm 1 has a further advantage which can be demonstrated using the *"Three Tank OED"* problem. We did not include this problem in our benchmark set since it converges very slowly and can only be solved consistently when using Algorithm 1 and the exact Hessian. The reason for this behavior is that the underlying dynamics of the system contain square roots. For single shooting, the terms within these square roots are always positive, but the matching violations can cause them to become negative, making the NLP solver fail. Replacing the states with FSInit mitigates this problem and restores the physically correct positive terms.
We do not claim that our current implementation is fully optimized. Some duplicate evaluations could be avoided through tighter integration with the solver. Yet, we observe convincing real-time improvements. For instance, applying Algorithm 3 to the Quasi-Newton MS-64 version of *"Lotka OED"* took on average around 41 seconds, compared with around 50 seconds

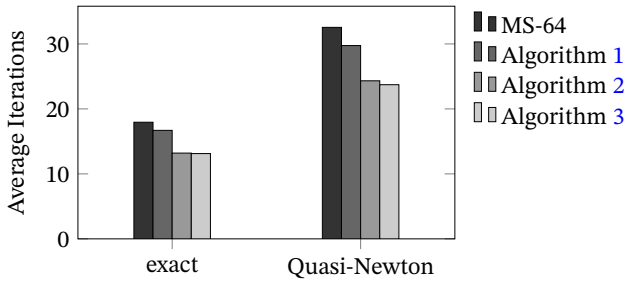

FIGURE 11 | Average number of iterations for MS-64 and the algorithms from Section 4 over all 40 benchmark, as in Figure 3.

without it. Similarly, the using Algorithm 3 for the exact Hessian version of *"Egerstedt"* took 39 seconds, compared with 81 seconds for the default MS-64.

6 | Discussion and outlook

We close by discussing the results from Section 5 in Subsection 6.1, followed by a conclusion and outlook in Subsection 6.2.

6.1 | Discussion

Consistent with Section 3, lifting at every control discretization proved to be the best choice. As expected, this leaves little room for improvement when applying our algorithms to these problems. In many cases, the algorithms either save only one or two iterations or, due to the unpredictability of the Newton path, add one or two iterations. Despite the generally small overhead of Algorithm 1 and the negligible overhead of Algorithm 2 (since the quantities in (6) and (7) are easy to obtain), the real-time performance of Algorithm 3 is then on par with, or slightly worse than, MS-64. We expect the performance of the combined Algorithm 3 to match the better of its two components. Apart from small deviations, this is what we observe, with performance being occasionally slightly better or slightly worse.
To address all algorithms and their use cases, let us differentiate between the following three scenarios:

Far away from the solution

In this case, the solution process is dominated by the nonlinearity of the problem and possibly numerical issues caused by bad initial guesses. This effect can be observed in the number of problems that failed when using single shooting (compare Figure 6). Therefore, multiple shooting is the best choice, with shooting variables that can be determined through merit-based initialization via dynamic programming.

Medium distance to the solution

At this point we assume that the convergence behavior is no longer governed by strong nonlinearities and numerical issues with the integrator. For Quasi-Newton methods the convergence now depends on the quality of the Hessian approximation. Moreover, both the Quasi-Newton and exact Hessian formulation benefit from the parallelization and sparse Hessian structure when it comes to real-time performance (see Figure 4). Here, Algorithm 1 may lead to a better path and avoid singularities (as for the *"Three Tank OED"* problem). To summarize, choosing the finest shooting grid and applying Algorithm 1 constitutes the best choice.

Close to the solution

Most lower-dimensional problems quickly enter the area of superlinear convergence once they are close to the solution and there is little room or demand for improvement. The interesting cases are those problems which exhibit very poor convergence close to the solution when using `blockSQP2` and mainly for the Quasi-Newton variant. This issue can be addressed by switching to single shooting via Algorithm 2. For Quasi-Newton methods this leads to strong real-time improvements for problems like *"Lotka OED"*. For the exact Hessian, however, the huge computational cost of computing the exact single shooting Hessian usually outweighs the time saved by performing fewer iterations.

The transformation $x \longmapsto -x^2$ described in [23] to improve the local convergence of OED problems does not contradict our approach. In fact, we observe even better performance when combining it with Algorithm 2. Moreover, this issue does not appear to affect IPOPT. We conjecture that poor local conditioning is mitigated by adjustments of the barrier parameter.

6.2 | Conclusion and outlook

We conducted a broad numerical study on the effect of lifting in optimal control problems.
For exact-Hessian methods, we observed that lifting at all control discretizations is usually the best choice. It has either a minor or a positive influence on the required number of iterations, while noticeably accelerating the evaluation of the Hessian, objective, and constraint functions.
Quasi-Newton methods with high-rank block-wise updates benefit even more from lifting because the Hessian approximations improve. However, several problems exhibit worse convergence as lifting is increased. In particular, OED problems often fall into this category.
The proposed algorithms produced promising results when applied to our benchmark set. In particular, we observe that poor convergence can be mitigated by switching to single shooting or correcting the shooting points via FSInit.
There are multiple enticing directions for future research. First, the current switching and initialization criteria could be made more adaptive, for instance by learning problem-dependent thresholds from online solver statistics instead of relying on manually chosen global values. Second, a custom condensing procedure which switches to a slightly coarser shooting grid might enable a better trade-off between the number of iterations and the cost per iteration. Third, a tighter integration into the solver could reduce duplicate evaluations and improve warm-start quality after switching. Finally, it would be valuable to augment our benchmark collection by closed-loop NMPC experiments and larger high-dimensional OED instances, in order to assess how reliably the proposed strategies improve not only iteration counts but also end-to-end runtime in more difficult settings.

ETHICS DECLARATION

During the preparation of this work the authors made use of ChatGPT 5 and Claude Sonnet 4.7 to refine the language, grammar, and readability.

References

1. John T. Betts 2001. *Practical Methods for Optimal Control Using Nonlinear Programming*, Philadelphia: SIAM.
2. Hans Georg Bock, and Karl-Josef Plitt. 1984. "A Multiple Shooting algorithm for direct solution of optimal control problems." In *Proceedings of the 9th IFAC World Congress*, 242–247. Budapest: Pergamon Press.
3. D.B. Leineweber 1999. *Efficient reduced SQP methods for the optimization of chemical processes described by large sparse DAE models*, Vol 613 of *Fortschritt-Berichte VDI Reihe 3, Verfahrenstechnik*. Düsseldorf: VDI Verlag.
4. Martin Schlegel 2005. *Adaptive discretization methods for the efficient solution of dynamic optimization problems*, Vol 829 of *Fortschritt-Berichte VDI Reihe 3, Verfahrenstechnik*. Düsseldorf: VDI Verlag.
5. Moritz Diehl, Hans Georg Bock, Johannes P. Schlöder, Rolf Findeisen, Zoltan Nagy, and Frank Allgöwer. "Real-time optimization and Nonlinear Model Predictive Control of Processes governed by differential-algebraic equations." *J. Proc. Contr.* 12, no. 4 (2002): 577–585.
6. R.M.M. Mattheij, and G.W.M. Staarink. "On optimal shooting intervals." *Mathematics of Computation* 42, no. 165 (1984): 25–40.
7. Michael Ernst Geiger 2015. "Adaptive Multiple Shooting for Boundary Value Problems and Constrained Parabolic Optimization Problems."
8. Jan Albersmeyer, and Moritz Diehl. "The Lifted Newton Method and Its Application in Optimization." *SIAM Journal on Optimization* 20, no. 3 (2010): 1655–1684.
9. Robert Lampel, and Sebastian Sager. 2025. "On liftings that improve convergence properties of Newton's Method for Boundary Value Optimization Problems."
10. Andreas Wächter, and Lorenz T. Biegler. "On the Implementation of an Interior-Point Filter Line-Search Algorithm for Large-Scale Nonlinear Programming." *Mathematical Programming* 106, no. 1 (2006): 25–57.
11. Dennis Janka, Christian Kirches, Sebastian Sager, and Andreas Wächter. "An SR1/BFGS SQP algorithm for nonconvex nonlinear programs with block-diagonal Hessian matrix." *Mathematical Programming Computation* 8, no. 4 (2016): 435–459.
12. Reinhold Wittmann, and Sebastian Sager. "blockSQP 2: exploiting structure to improve performance." Manuscript in preparation.

13. Jack Kiefer, and Jacob Wolfowitz. "Optimum Designs in Regression Problems." *The Annals of Mathematical Statistics* 30, no. 2 (1959): 271–294.
14. Irene Bauer, Hans Georg Bock, Stefan Körkel, and Johannes P. Schlöder. "Numerical methods for optimum experimental design in DAE systems." *J. Comput. Appl. Math.* 120, no. 1-2 (2000): 1–15.
15. Stefan Körkel, Ekaterina Kostina, Hans Georg Bock, and Johannes P. Schlöder. "Numerical Methods for Optimal Control Problems in Design of Robust Optimal Experiments for Nonlinear Dynamic Processes." *Optimization Methods and Software* 19 (2004): 327–338.
16. Stefan Körkel, Andreas Potschka, Hans Georg Bock, and Sebastian Sager. "A multiple shooting formulation for optimum experimental design." *Mathematical Programming* (2012): 18–79.
17. Sebastian Sager. "Sampling Decisions in Optimum Experimental Design in the Light of Pontryagin's Maximum Principle." *SIAM Journal on Control and Optimization* 51, no. 4 (2013): 3181–3207.
18. Revaz Gamkrelidze 1978. *Principles of Optimal Control Theory*, Plenum Press.
19. Jorge Nocedal, and Stephen J. Wright. 2006. *Numerical Optimization*, Springer.
20. Joel A.E. Andersson, Joris Gillis, Greg Horn, James B. Rawlings, and Moritz Diehl. "CasADi – A software framework for nonlinear optimization and optimal control." *Mathematical Programming Computation* 11, no. 1 (2019): 1–36.
21. Roger Fletcher, and Sven Leyffer. "Nonlinear programming without a penalty function." *Mathematical Programming* 91, no. 2 (2002): 239–269.
22. Peter Deuflhard 2006. *Newton Methods for Nonlinear Problems. Affine Invariance and Adaptive Algorithms*, Vol 35 of *Springer Series in Computational Mathematics*. Springer.
23. Mario S. Mommer, Andreas Sommer, Johannes P. Schlöder, and Hans G. Bock. "A nonlinear preconditioner for experimental design problems." *arXiv preprint arXiv:1108.1689*.
24. Joel Andersson, Johan Åkesson, and Moritz Diehl. 2012. "CasADi – A symbolic package for automatic differentiation and optimal control." In *Recent Advances in Algorithmic Differentiation*, edited by S. Forth, P. Hovland, E. Phipps, J. Utke, and A. Walther, Lecture Notes in Computational Science and Engineering, Berlin: Springer.
25. Dennis Janka, Stefan Körkel, and Hans Georg Bock. 2015. "Direct multiple shooting for nonlinear optimum experimental design." In *Multiple Shooting and Time Domain Decomposition Methods*, 115–141. Springer.
26. Dennis Janka 2015. "Sequential quadratic programming with indefinite Hessian approximations for nonlinear optimum experimental design for parameter estimation in differential-algebraic equations."
27. Sebastian Sager 2011. "A benchmark library of mixed-integer optimal control problems." In *Mixed Integer Nonlinear Programming*, 631–670. Springer.
28. Charles R. Harris, K. Jarrod Millman, Stéfan J. van der Walt, Ralf Gommers, Pauli Virtanen, David Cournapeau, Eric Wieser, Julian Taylor, Sebastian Berg, Nathaniel J. Smith, Robert Kern, Matti Picus, Stephan Hoyer, Marten H. van Kerkwijk, Matthew Brett, Allan Haldane, Jaime Fernández del Río, Mark Wiebe, Pearu Peterson, Pierre Gérard-Marchant, Kevin Sheppard, Tyler Reddy, Warren Weckesser, Hameer Abbasi, Christoph Gohlke, and Travis E. Oliphant. "Array programming with NumPy." *Nature* 585, no. 7825 (2020): 357–362.
29. Eric Jones, Travis Oliphant, Pearu Peterson, et al. 2001–2015. "SciPy: Open source scientific tools for Python."
30. Robert Lampel 2026. "PySHeRLOC." https://github.com/rlampel/PySHeRLOC, Archived at Zenodo: https://doi.org/10.5281/zenodo.21064485.

APPENDIX

Figures 1 and 2 show the relative number of iterations and the corresponding real-time performance as described in Section 3. Figure 3 shows the performance of our algorithms as written in Section 5.

FIGURE 1 | Comparative analysis of the relative number of required iterations across benchmark problems. Each bar shows the average of five runs with slightly perturbed control initializations. We do not perturb the controls for the *"Quadrotor"* problem, as it would converge to very different local minima. For every solver the number of iterations for the best lifting is set to one and all other iterations are scaled accordingly. Bars that touch the top of their figure represent problems that did not converge. The number of multiple shooting nodes is shown on the logarithmic x-axis. The titles refer to the description on mintoc.de [27].

FIGURE 2 | As in Figure 1, but comparing the average absolute computational times.

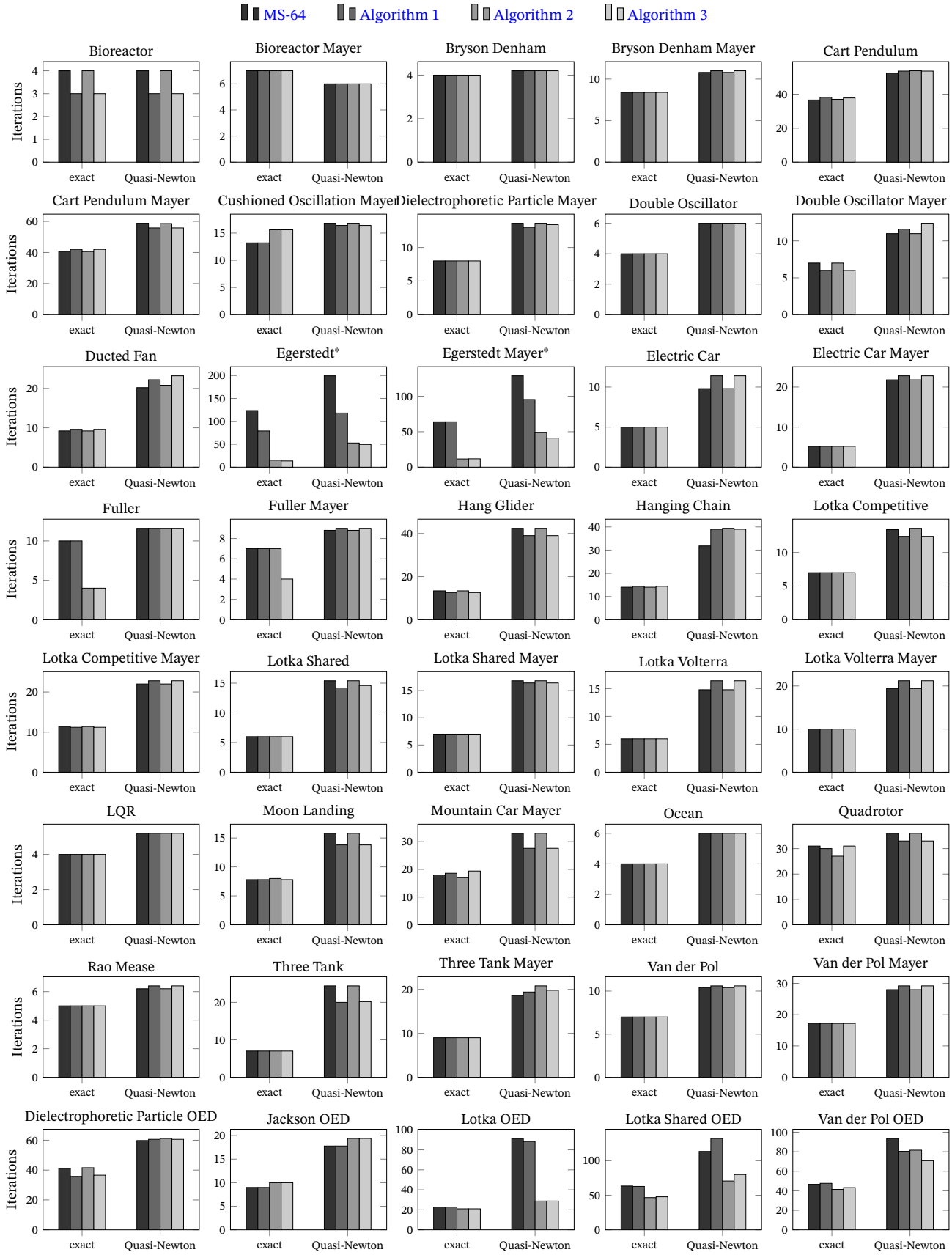

FIGURE 3 | As in Figure 1, but comparing the number of iterations for the algorithms from Section 4. We consider both the exact and the Quasi-Newton version of `blockSQP2`. (* For both *Egerstedt* problems we removed the cap of 200 iterations to calculate the number of iterations for MS-64.)

Algorithm 4 Merit-based initialization via dynamic programming

Input:
$\mathcal{S} = \{\mathcal{S}_k\}_{k=0}^{m}$ ▷ candidate states
$h_{[j_k]},\ v_{[j_k]},\ k = 0, \dots, m$
$\mu \in \mathbb{R}$ ▷ current penalty parameter

for $s \in \mathcal{S}_m$ do
  $J(s) \leftarrow h_{[j_m]}(s, \mathbf{q}_{[j_m]}) + \mu \cdot v_{[j_m]}(s, \mathbf{q}_{[\mathbf{j_m}]})$ ▷ final costs
end for
for $k = m-1, \dots, 0$ do ▷ main loop
  for all $s^{[k]} \in \mathcal{S}_k$ do
    $s^{[k+1]} \leftarrow y_{[j_k]}(t_k, s^{[k]}, \mathbf{q}_{[j_k]}) \in S_{k+1}^{(\text{old})}$ ▷ FSInit value
    $v_{[j_k]}(s^{[k]}) \leftarrow v_{[j_k]}(s^{[k]}, \mathbf{q}_{[j_k]})$ ▷ violation
    $h_{[j_k]}(s^{[k]}) \leftarrow h_{[j_k]}(s^{[k]}, \mathbf{q}_{[j_k]})$ ▷ objective
    $P_{[j_k]}(s^{[k]}) \leftarrow h_{[j_k]}(s^{[k]}) + \mu \cdot v_{[j_k]}(s^{[k]})$ ▷ local contribution to merit function
    $J(s^{[k]}) \leftarrow J(s^{[k+1]}) + P_{[j_k]}(s^{[k]})$
    $\pi(s^{[k]}) \leftarrow s^{[k+1]}$
    for all $s' \in \mathcal{S}_{k+1}^{(\text{new})}$ do
      $myval \leftarrow J(s') + \mu \|s^{[k+1]} - s'\|_1 + P_{[j_k]}(s^{[k]})$ ▷ cost with matching error
      if $myval < J(s^{[k]})$ then
        $\pi(s^{[k]}) \leftarrow s'$ ▷ store lifted transition
        $J(s^{[k]}) \leftarrow myval$ ▷ update cost-to-go
      end if
    end for
  end for
end for
choose $\mathbf{s}_0 \in \operatorname{argmin}_{s \in \mathcal{S}_0} J(s)$
for $i = 0, \dots, m-1$ do ▷ get final solution
  $\mathbf{s}_{[j_{i+1}]} \leftarrow \pi(\mathbf{s}_{[j_i]})$
end for
return $\mathbf{s} = (\mathbf{s}_0, \mathbf{s}_{j_1}, \dots, \mathbf{s}_{j_m})$