Exploiting Sparse Structures in Nonlinear Model Predictive Control with Hypergraphs

Christoph Rösmann, Maximilian Krämer, Artemi Makarow, Frank Hoffmann and Torsten Bertram

IEEE/ASME Conference on Advanced Intelligent Mechatronics (AIM), Auckland, New Zealand, 2018

Exploiting Sparse Structures in Nonlinear Model Predictive Control with Hypergraphs

Christoph Rösmann, Maximilian Krämer, Artemi Makarow, Frank Hoffmann and Torsten Bertram

Abstract— **This paper proposes a hypergraph formulation for solving MPC problems. The hypergraph approach exploits the sparse structure in the calculation of derivatives. It is therefore computationally more efficient in case of multiple-shooting, collocation and full-discretization methods compared to a dense formulation. Recent advances in realtime optimization rely on automatic differentiation (AD) to compute derivatives. An extensive analysis compares MPC variants with both hypergraph and AD on two benchmark control problems. Even though AD requires a computational overhead to set up the problem structure, solving the nonlinear program at each iteration is fast. The overhead in the hypergraph approach is negligible, and computational effort in the solving phase is inferior but comparable to AD. This observation favors the hypergraph representation for MPC problems with non-static problem structure.**

I. INTRODUCTION

Model predictive control (MPC) constitutes a promising approach to optimal regulation of nonlinear dynamic systems w.r.t. general performance criteria under explicit compliance with state and control input constraints [1]. The practical application of MPC faces two major challenges: First, it requires an accurate model of the system dynamics and second, it demands substantial computational resources in comparison to conventional control concepts such as PID or linear quadratic regulators. During the last two decades, the interest in numerically efficient and computationally fast implementations of nonlinear MPC has grown significantly in both theoretical and application-oriented research.

In the context of continuous-time dynamic models, established methods solve the underlying OCP either indirectly or directly. Indirect methods utilize the calculus of variations, whereas direct methods transform the continuous-time problem into a nonlinear program with a finite number of optimization variables. Direct methods discretize the underlying boundary value problem, i.e., adherence to the continuous-time dynamics, and are furthermore divided into either a sequential (single-shooting) or simultaneous strategy (multiple-shooting, collocation). Sequential approaches merely optimize the sequence of controls. Simultaneous strategies also consider the state trajectories which leads to sparse albeit larger problem structures and generally improves convergence. In multiple-shooting, the state trajectory is partitioned into multiple discrete intervals for which isolated initial value problems are solved [2]. Equality constraints enforcing continuity among shooting intervals are incorporated in the nonlinear program. Collocation methods interpolate the state and control trajectories (dynamics equation) by finite differences or quadrature rules (i.e., implicit numerical integration resp. spline interpolation). Similiar to multiple-shooting, the collocation equations are incorporated as additional equality constraints subject to optimization.

To reduce the computational effort in each sampling instance, Diehl et al. propose the real-time iteration (RTI) scheme which merely applies a single warm-started sequential-quadratic-programming step at each sampling interval [3]. A further application and analysis of RTI for embedded nonlinear MPC is presented in [4]. Zanelli et al. modify the RTI scheme to significantly reduce the number of variables subject to optimization by applying a backward Riccati sweep to a subset of the horizon [5]. In [6], a curvature-based measure of nonlinearity is exploited to reduce the number of sensitivity computations. Graichen et al. utilize projected gradients for efficient real-time capable MPC [7]. [8] proposes an efficient gradient-based method in which the OCP is transformed to an unconstrained auxiliary problem with interior penalties. For time-optimal control tasks, [9] presents a dynamic shooting-grid in which the number of control inputs becomes as low as the required number of control interventions.

Exploiting the sparse structure of the underlying OCPs significantly reduces the computational burden in case of larger horizon lengths, e.g., Wang et al. suggest to use simultaneous methods and to apply warm-starting [10]. Hereby, the nonlinear program is solved using sparse and early-terminating interior-point methods. Condensing techniques also exploit the structure to transform the nonlinear program into a smaller and dense auxiliary program [11]. Nielsen et al. focus on the parallelization of the Newton step arising in both active-set and interior-point solvers by exploiting the sparse structure of the nonlinear program as well [12].

Most of the above methods have in common that they compute at least first order and often second order derivatives of the objective function as well as constraints. If closed form analytic derivatives are not available, they are computed automatically in either numeric or symbolic form. The numeric computation relies on finite differences, usually central differences to achieve the desired precision. On the other hand, automatic differentiation (AD) recently emerged as a popular and ubiquitous approach for computing derivatives symbolically as in the ACADO toolkit [13]. AD inherently retains the sparse structure of the OCP and hence avoids the numerical evaluation of structured zero elements in first and second order derivatives. CasADi [14] constitutes a mature

This work is funded by the German Research Foundation (DFG, BE 1569/13-1). The authors are with the Institute of Control Theory and Systems Engineering, Technische Universität Dortmund, D-44227, Germany, christoph.roesmann@tu-dortmund.de

and efficient open-source AD framework frequently reported in the MPC literature as the preferred tool in the realm of optimal control. The symbolic framework of AD enables an elegant and simple way to formulate the OCP by merely formulating the mathematical expressions of the nonlinear program while preserving its sparse structure.

Robotics and computer vision, simultaneous localizing and mapping as well as bundle adjustment require the solution of large, structured, albeit unconstrained optimization problems [15]. These optimization problems are defined in terms of a graph to restrict computations to only structured non-zeros. E.g., Kümmerle et al. present an efficient general graph optimization framework for unconstrained least-squares problems [15]. Problems are formulated in terms of a hypergraph in which edges correspond to cost function terms. The edges reflect the connectivity of optimization parameters represented by vertices and hence the hypergraph preserves the structure of the optimization problem. In [16], the hypergraph is adopted to an unconstrained trajectory optimization problem for mobile robots.

Considering the recent advances in numerical optimization in MPC, it is worthwhile to investigate and analyze the hypergraph-based optimization for MPC type constrained optimal problems, and to contrast its performance and computational efficiency with established AD frameworks. This contribution presents the utilization and extension of hypergraph-based optimization to constrained OCPs in MPC. A major focus lies in the extensive comparative analysis with CasADi and conventional dense central differences solutions.

The next section summarizes the nonlinear MPC formulations. Section III introduces a hypergraph formulation for the specific MPC formulations which is then evaluated in terms of a systematic comparative analysis in section IV. Finally, section V discusses and summarizes the results.

II. Nonlinear Model Predictive Control

This section introduces the fundamentals of direct methods in MPC that provide the foundation for the hypergraph formulation in section III.

A. Single-Shooting Approach

Continuous-time dynamics with control input $\mathbf{u}(t) \in \mathbb{R}^q$ and state $\mathbf{x}(t) \in \mathbb{R}^p$ are defined as $\dot{\mathbf{x}}(t) = \mathbf{f}\big(\mathbf{x}(t), \mathbf{u}(t)\big)$. The single-shooting approach discretizes the control input trajectory along a fixed grid with N partitions: $t_0 < t_1 < \ldots < t_k < \ldots < t_N = t_f$. Hereby, $\mathbf{u}(t) := \mathbf{u}_k$ for $t \in [t_k, t_{k+1})$ is defined as a piecewise constant trajectory, also denoted as control input sequence. The state trajectory emerges as the solution of the initial value problem (IVP) $\mathbf{x}(t) = \int_{t_0}^{t_f} \mathbf{f}\big(\mathbf{x}(t), \mathbf{u}(t)\big)\,\mathrm{d}t$ with initial state $\mathbf{x}_s$. The single-shooting OCP is defined as follows:

$$\min_{\mathbf{u}_0, \mathbf{u}_1, \ldots, \mathbf{u}_{N-1}} \Big[V_f\big(\mathbf{x}(t_f)\big) + \int_{t_0}^{t_f} \ell\big(\mathbf{x}(t), \mathbf{u}(t)\big)\,\mathrm{d}t\Big] \qquad (1)$$

subject to

$$\mathbf{x}(t_0) = \mathbf{x}_s, \quad \mathbf{x}(t_k) \in \mathbb{X}, \quad \mathbf{u}_k \in \mathbb{U}, \quad \mathbf{x}(t_f) \in \mathbb{X}_f.$$

Hereby, $\mathbf{x}(t_k)$ denotes the state at grid time instance t_k obtained from the IVP. The sets $\mathbb{U}$ and $\mathbb{X}$ define the feasible region of controls and states. In order to enforce stability, the problem formulation includes a terminal state cost $V_f : \mathbb{R}^p \to \mathbb{R}$ and a terminal set $\mathbb{X}_f$ [17]. For sake of readability, the integral terms of the cost functional and IVP employ the notation $\mathbf{u}(t)$. Obviously, in any implementation, $\mathbf{u}(t)$ is replaced by the sequence of $\mathbf{u}_k$ for which the set of relevant subscripts k immediately follows from the integral limits and the temporal grid.

B. Multiple-Shooting Approach

Direct optimal control with single shooting results in a compact optimization problem with a discrete set of control inputs as the only parameters. However, the structure of NLP (1) is dense since the running cost terms explicitly depend on the entire sequence $\mathbf{u}_k$ for $k = 0, 1, \ldots, N-1$. The key idea of multiple-shooting is to partition the IVP on the interval $[t_0, t_f]$ into multiple IVPs to be solved in isolation. Connectivity and compliance among the IVPs is enforced by additional equality constraints. Similar to single-shooting, the control input trajectory is discretized along the fixed grid with N partitions in which t_k for $k = 0, 1, \ldots, N$ defines the grid points. In addition, the state trajectory is discretized either on the same or on a sparser grid with subscripts i: $t_0 < t_1 < \ldots < t_i < \ldots < t_M = t_f$ with $M \le N$ and $t_i \in \{t_k \mid k = 0, 1, \ldots, N\}$. The latter condition enforces that state grid points coincide with control grid points. States on the state grid are denoted as so-called shooting nodes $\mathbf{s}_i := \mathbf{x}(t_i)$ as they constitute the initial states of the IVP on the interval $[t_i, t_{i+1}]$. The solution of the i-th IVP is denoted by $\varphi\big(\mathbf{u}(t), \mathbf{s}_i\big) = \int_{t_i}^{t_{i+1}} \mathbf{f}\big(\mathbf{x}(t), \mathbf{u}(t)\big)$ with $\mathbf{x}(t_i) = \mathbf{s}_i$ associated with the NLP defined by:

$$\min_{\substack{\mathbf{u}_0, \mathbf{u}_1, \ldots, \mathbf{u}_{N-1} \\ \mathbf{s}_0, \mathbf{s}_1, \ldots, \mathbf{s}_M}} \Big[V_f(\mathbf{s}_N) + \sum_{i=0}^{M-1} \underbrace{\int_{t_i}^{t_{i+1}} \ell\big(\mathbf{x}(t), \mathbf{u}(t)\big)\,\mathrm{d}t}_{V_i\left(\mathbf{x}(t), \mathbf{u}(t)\right)}\Big] \qquad (2)$$

subject to

$$\mathbf{s}_0 = \mathbf{x}_s, \quad \mathbf{s}_i \in \mathbb{X}, \quad \mathbf{u}_k \in \mathbb{U}, \quad \mathbf{s}_M \in \mathbb{X}_f,$$
$$\mathbf{s}_{i+1} = \varphi\big(\mathbf{u}(t), \mathbf{s}_i\big).$$

Note, cost term $V_i(\cdot)$ only depends on $\mathbf{s}_i$ and the subset of $\mathbf{u}_k$ that fall belong to the shooting interval $[t_i, t_{i+1}]$. The additional equality constraints enforce connectivity among two consecutive shooting intervals and depend on the above subset of parameters as well as $\mathbf{s}_{i+1}$. For the sake of simplicity, state constraints are only evaluated at shooting nodes $\mathbf{s}_i$. It is straightforward to enforce them for $\mathbf{x}(t_k)$, $k = 0, 1, \ldots, N$ as well, by considering intermediate solutions of the underlying IVP.

The NLP (2) has more parameters but usually converges faster to the optimal solution [2]. Furthermore, a proper initial guess of the state trajectory in terms of $\mathbf{s}_i$ significantly reduces the computation time. This property is exploited in common warm-start techniques as they utilize the solution from the previous time step to initialize the parameters of the current problem [10].

C. Direct Collocation Approach

Direct collocation also discretizes both the state and the control input trajectory. Instead of solving IVPs over (commonly) a piecewise constant control sequence, collocation methods interpolate the dynamics equation and cost functional between two consecutive states and controls with predefined basis functions, e.g., polynomials, usually obtained from numerical quadrature. The grid $t_0 < t_1 < \ldots < t_k < \ldots < t_N = t_f$ is defined for both states and control inputs such that $\mathbf{x}_k$ and $\mathbf{u}_k$ are the states and controls at grid points t_k respectively. The further analysis relies on Hermite-Simpson collocation commonly used in practice. Hereby, system dynamics $\mathbf{f}\big(\mathbf{x}(t), \mathbf{u}(t)\big)$ are interpolated on the interval $[t_k, t_{k+1}]$ by Simpson quadrature [18], in particular

$$\begin{aligned}\boldsymbol{\phi}(\mathbf{x}_k, \mathbf{x}_{k+1}, \mathbf{u}_k, \mathbf{u}_{k+\frac{1}{2}}, \mathbf{u}_{k+1}) = \frac{1}{6}\Delta t_k \cdot \ldots \\ \cdot\big(\mathbf{f}(\mathbf{x}_k, \mathbf{u}_k) + 4\mathbf{f}(\mathbf{x}_{k+\frac{1}{2}}, \mathbf{u}_{k+\frac{1}{2}}) + \mathbf{f}(\mathbf{x}_{k+1}, \mathbf{u}_{k+1})\big)\end{aligned} \tag{3}$$

with $\Delta t_k = t_{k+1} - t_k$. The midpoint state is defined as $\mathbf{x}_{k+\frac{1}{2}} = 0.5(\mathbf{x}_k + \mathbf{x}_{k+1}) + \Delta t_k\big(\mathbf{f}(\mathbf{x}_k, \mathbf{u}_k) - \mathbf{f}(\mathbf{x}_{k+1}, \mathbf{u}_{k+1})\big)/8$. The midpoint control $\mathbf{u}_{k+\frac{1}{2}}$ is subject to optimization. Simpson quadrature interpolates the system dynamics $\dot{\mathbf{x}}(t) = \mathbf{f}(\cdot)$ and control trajectory $\mathbf{u}(t)$ via quadratic splines. Consequently, the resulting state trajectory $\mathbf{x}(t)$ is a hermite-cubic spline that complies with the actual dynamics at grid points t_k. Accordingly, the running cost $\ell\big(\mathbf{x}(t), \mathbf{u}(t)\big)$ is also approximated by Simpson quadrature: $V_k\big(\mathbf{x}_k, \mathbf{x}_{k+1}, \mathbf{u}_k, \mathbf{u}_{k+\frac{1}{2}}, \mathbf{u}_{k+1}\big) = \Delta t_k\big(\ell(\mathbf{x}_k, \mathbf{u}_k) + 4\ell(\mathbf{x}_{k+\frac{1}{2}}, \mathbf{u}_{k+\frac{1}{2}}) + \ell(\mathbf{x}_{k+1}, \mathbf{u}_{k+1})\big)/6$. The resulting NLP is defined by:

$$\min_{\substack{\mathbf{u}_0, \mathbf{u}_{\frac{1}{2}}, \mathbf{u}_1, \ldots, \mathbf{u}_N \\ \mathbf{x}_0, \mathbf{x}_1, \ldots, \mathbf{x}_N}} \Big[V_f\big(\mathbf{x}(t_f)\big) + \sum_{k=0}^{N-1} V_k(\cdot)\Big] \tag{4}$$

subject to

$$\begin{aligned}&\mathbf{x}_0 = \mathbf{x}_s, \quad \mathbf{x}_k \in \mathbb{X}, \quad \mathbf{u}_k \in \mathbb{U}, \quad \mathbf{u}_{k+\frac{1}{2}} \in \mathbb{U} \quad \mathbf{x}_N \in \mathbb{X}_f, \\ &\mathbf{x}_{k+1} - \mathbf{x}_k = \boldsymbol{\phi}(\mathbf{x}_k, \mathbf{x}_{k+1}, \mathbf{u}_k, \mathbf{u}_{k+\frac{1}{2}}, \mathbf{u}_{k+1}).\end{aligned}$$

Similar to multiple shooting, state constraints are evaluated at midpoints $\mathbf{x}_{k+\frac{1}{2}}$ if necessary.

D. Full Discretization

Direct optimal control via full discretization is a particular variant of multiple-shooting respectively collocation. It assumes, that each shooting interval coincides with a single control action, i.e. $M = N$ which implies $i = k$, and Δt_k small, for quadrature a simple one-step scheme such as forward Euler (as long as the dynamics are not stiff) is often sufficient. The integral of the running costs is approximated by $V_k\big(\mathbf{x}(t), \mathbf{u}(t)\big) \approx \ell(\mathbf{x}_k, \mathbf{u}_k)\Delta t_k$. Similarly, a full discretization is achieved via direct collocation, if system dynamics are approximated by finite differences, e.g. forward differences, and controls are assumed to be piecewise constant (midpoints are omitted).

III. Hyper-Graph Formulation

Solving the NLPs in (1), (2) and (4) requires solvers that cope with nonlinear cost functions and hard-constraints.

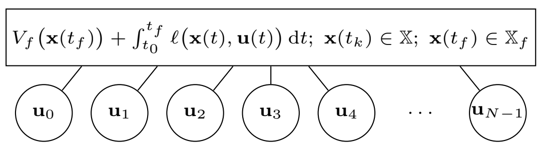

$V_f\big(\mathbf{x}(t_f)\big) + \int_{t_0}^{t_f} \ell\big(\mathbf{x}(t), \mathbf{u}(t)\big)\,\mathrm{d}t;\ \mathbf{x}(t_k) \in \mathbb{X};\ \mathbf{x}(t_f) \in \mathbb{X}_f$
$\mathbf{u}_0$
$\mathbf{u}_1$
$\mathbf{u}_2$
$\mathbf{u}_3$
$\mathbf{u}_4$
$\cdots$
$\mathbf{u}_{N-1}$

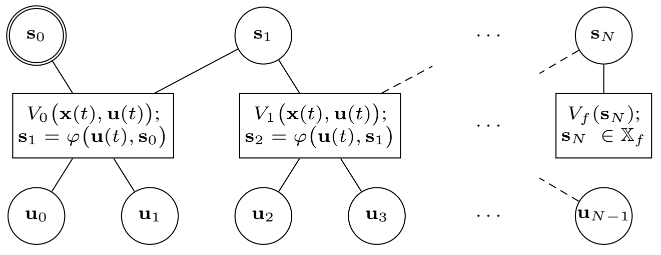
Fig. 1: Single-shooting hypergraph example

Newton-type methods are well established, in particular interior-point solvers, sequential quadratic programming approaches with underlying active-set solvers, or gradient based methods such as projected gradients [19]. Numerical solvers at least calculate first order derivatives. In many MPC applications with Newton-type methods, Broyden-Fletcher–Goldfarb-Shanno (BFGS) methods approximate second order derivatives rather than calculating the Hessian explicitly. In numerical optimization, derivatives are usually calculated from finite differences. In particular central differences are well established as they balance the trade-off between accuracy and computational effort. The derivative of a function $\psi(x)$ is approximated by $\mathrm{d}\,\boldsymbol{\psi}(x)/\,\mathrm{d}\,x \approx 0.5h^{-1}\big(\boldsymbol{\psi}(x+h) - \boldsymbol{\psi}(x-h)\big)$, e.g., with $h = 10^{-9}$. Without prior knowledge of the structure of the NLP, each constraint and cost term is evaluated twice per parameter to obtain its first order derivative. Sparsity in the context of MLP goes both ways, namely that the stage cost terms V_k, V_i and constraints in the OCP (2) and (4) only depend on a few parameters x_k, u_k and vice versa that parameters x_k, u_k only contribute to few cost terms. The former property implies sparse constraint Jacobians and Hessians. The latter sparsity allows an efficient computation of the cost gradient.

In the following, the OCPs are formulated in terms of hypergraphs (HG) with the objective to exploit the inherent sparse problem structure algorithmically. A HG is a graph $\mathcal{G}(\mathcal{V}, \mathcal{E})$ composed of a set of vertices $\mathcal{V}$ and a set of edges $\mathcal{E}$. Edges in a HG connect several vertices rather than only pairs of vertices as conventional graphs. Vertices and edges are defined for OCPs (1), (2) and (4) as follows:

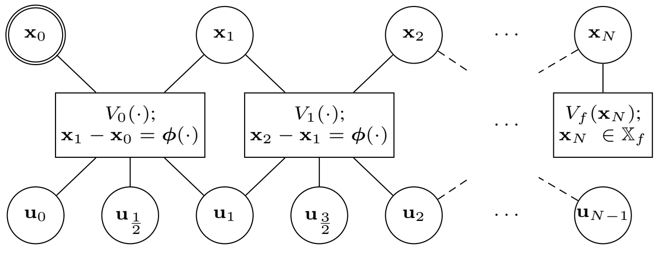
Vertices: A vertex refers to a vector of optimization parameters $\mathbf{v}_j \in \{\mathbb{R}^p, \mathbb{R}^q\}$, i.e. either a state or control input vector. Box constraints on $\mathbb{X}$ and $\mathbb{U}$, in particular lower and upper limits $\mathbf{v}_{\min} \leq \mathbf{v}_j \leq \mathbf{v}_{\max}$, are directly cached in the vertex. The parameters of a so called fixed vertex, although not subject to optimization, nevertheless appear in cost and constraint terms. Fixed states replace trivial equality constraints such as $\mathbf{x}_0 = \mathbf{x}_s$ by substitution of $\mathbf{x}_0$ by $\mathbf{x}_s$.

Edges: An edge refers to a scalar cost term as well as equality and inequality constraints, figuratively the 3-tupel $(V_l, \mathbf{g}_l, \mathbf{h}_l)$. The idea behind considering a tupel of costs and constraints is to share common resources which is advantageous for higher order numerical integrators or quadrature. Note, the evaluation of cost terms might incorporate intermediate solutions of the dynamics IVP or collocation equation. Shooting and collocation methods impose constraints on intermediate states respectively controls. $V_l \colon \mathcal{D}_l \to \mathbb{R}$, $\mathbf{g}_l \colon \mathcal{D}_l \to \mathbb{R}^n$ and $\mathbf{h}_l \colon \mathcal{D}_l \to \mathbb{R}^m$ define the mapping of the directly dependent parameters $\mathcal{D}_l = \{\mathbf{v}_j \mid \mathbf{v}_j \subseteq \mathcal{V}\}$ onto the cost or constraint terms.



$\mathbf{s}_0$
$\mathbf{s}_1$
$\mathbf{s}_N$
$V_0(\mathbf{x}(t),\mathbf{u}(t))$; $\mathbf{s}_1 = \varphi(\mathbf{u}(t),\mathbf{s}_0)$
$V_1(\mathbf{x}(t),\mathbf{u}(t))$; $\mathbf{s}_2 = \varphi(\mathbf{u}(t),\mathbf{s}_1)$
$V_f(\mathbf{s}_N)$; $\mathbf{s}_N \in \mathbb{X}_f$
$\mathbf{u}_0$
$\mathbf{u}_1$
$\mathbf{u}_2$
$\mathbf{u}_3$
$\mathbf{u}_{N-1}$

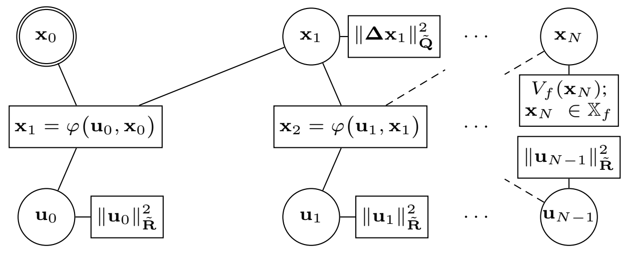
Fig. 2: Multiple-shooting hypergraph with two controls per shooting interval. A double circle indicates a fixed vertex.



$\mathbf{x}_0$
$\mathbf{x}_1$
$\mathbf{x}_2$
$\mathbf{x}_N$
$V_0(\cdot)$; $\mathbf{x}_1 - \mathbf{x}_0 = \boldsymbol{\phi}(\cdot)$
$V_1(\cdot)$; $\mathbf{x}_2 - \mathbf{x}_1 = \boldsymbol{\phi}(\cdot)$
$V_f(\mathbf{x}_N)$; $\mathbf{x}_N \in \mathbb{X}_f$
$\mathbf{u}_0$
$\mathbf{u}_{\frac{1}{2}}$
$\mathbf{u}_1$
$\mathbf{u}_{\frac{3}{2}}$
$\mathbf{u}_2$
$\mathbf{u}_{N-1}$

Fig. 3: Hermite-Simpson collocation hypergraph example

By definition, scalar costs V_l with $l = 0, 1, \ldots, L$ edges are accumulated in ascending order to sum up the overall cost, and constraints $\mathbf{h}_l$ and $\mathbf{g}_l$ are concatenated to mimic the complete NLP. To exploit sparsity, the main objective of transforming OCPs to HGs is to identify the subset of edges with a minimum or at least small number of dependent vertices $\mathcal{D}_l$. Obviously, single-shooting (1) contains no isolated summands in its cost function. Hence, the HG is composed of a single global edge with $\mathcal{D}_0 = \{\mathbf{u}_0, \mathbf{u}_1, \ldots, \mathbf{u}_{N-1}\}$. Fig. 1 depicts an example HG. The edge is shown as a rectangular box with the relevant cost terms and constraints. For the sake of clarity, the notation complies to (1). A particular implementation describes the set $\mathbb{X}$ and $\mathbb{X}_f$ as equality and inequality constraints, $\mathbf{h}_0$ and $\mathbf{g}_0$ respectively. In this example, the vertices include the box constraints on the controls $\mathbf{u}_k$. Obviously, the HG reveals no sparse structure in case of single-shooting. In contrast, the multiple shooting OCP (2) is partitioned into a graph with several edges, each corresponding to a particular shooting interval. The last edge captures the terminal cost and constraint. Fig. 2 shows an example HG with two controls per shooting interval. The subset of vertices for the shooting interval edges is $\mathcal{D}_l = \{\mathbf{u}_k, \mathbf{u}_{k+1}, \mathbf{s}_l, \mathbf{s}_{l+1}\}$ with $k = 2l$. Consequently, the number of structural non-zeros depends on the number of controls in a shooting interval. In Hermite-Simpson collocation (refer to Fig. 3) each edge depends on two consecutive states and three controls.



$\mathbf{x}_0$
$\mathbf{x}_1$
$\|\boldsymbol{\Delta}\mathbf{x}_1\|^2_{\tilde{\mathbf{Q}}}$
$\mathbf{x}_N$
$V_f(\mathbf{x}_N)$; $\mathbf{x}_N \in \mathbb{X}_f$
$\mathbf{x}_1 = \varphi(\mathbf{u}_0, \mathbf{x}_0)$
$\mathbf{x}_2 = \varphi(\mathbf{u}_1, \mathbf{x}_1)$
$\|\mathbf{u}_{N-1}\|^2_{\tilde{\mathbf{R}}}$
$\mathbf{u}_0$
$\|\mathbf{u}_0\|^2_{\tilde{\mathbf{R}}}$
$\mathbf{u}_1$
$\|\mathbf{u}_1\|^2_{\tilde{\mathbf{R}}}$
$\mathbf{u}_{N-1}$

Fig. 4: Full discretization hypergraph with quadratic cost

Algorithm 1: Compute derivatives from edge iterations

```
1: procedure COMPUTEDERIVATIVESEB($\mathcal{V}, \mathcal{E}, \boldsymbol{\nabla} V, \mathbf{J}_g, \mathbf{J}_h$)
2:   for all edges $l = 0, 1, \ldots, L$ do
3:     for each connected unfixed vertex $v$ do
4:       Compute gradient of $V_l$ and dense block Jacobians of $\mathbf{g}_l$ and $\mathbf{h}_l$ w.r.t. $v$
5:       Write values to proper positions in $\boldsymbol{\nabla} V, \mathbf{J}_g, \mathbf{J}_h$
6:   return $\boldsymbol{\nabla} V, \mathbf{J}_g, \mathbf{J}_h$
```

Algorithm 2: Compute derivatives from vertex iterations

```
1: procedure COMPUTEDERIVATIVESVB($\mathcal{V}, \mathcal{E}, \boldsymbol{\nabla} V, \mathbf{J}_g, \mathbf{J}_h$)
2:   for all unfixed vertices $j = 0, 1, \ldots, V$ do
3:     for each element $e$ in vertices[j] do
4:       $e \leftarrow e + h$          ▷ Central differences forward step
5:       Evaluate and cache $V_l, \mathbf{g}_l$ and $\mathbf{h}_l$ of connected edges
6:       $e \leftarrow e - 2h$         ▷ Central differences backward step
7:       Evaluate $V_l, \mathbf{g}_l$ and $\mathbf{h}_l$ of connected edges
8:       Apply central differences formula and write values to proper positions in $\boldsymbol{\nabla} V, \mathbf{J}_g, \mathbf{J}_h$
9:       $e \leftarrow e + h$          ▷ Revert variable to original value
10:  return $\boldsymbol{\nabla} V, \mathbf{J}_g, \mathbf{J}_h$
```

The most sparse structure is achieved by full discretization. Each edge is associated with cost terms that are independent of intermediate states of the dynamics IVP (see section II-D). Figure 4 depicts an example with a quadratic cost $V_k(\cdot) = \boldsymbol{\Delta}\mathbf{x}_k^\intercal \tilde{\mathbf{Q}} \boldsymbol{\Delta}\mathbf{x}_k + \mathbf{u}_k^\intercal \tilde{\mathbf{R}} \mathbf{u}_k$ with $\boldsymbol{\Delta}\mathbf{x}_k = \mathbf{x}_k - \mathbf{x}_{\text{ref}}$. The two cost terms for controls and state error are convex with positive definite weight matrices $\tilde{\mathbf{Q}} = \Delta t_k \mathbf{Q}$ and $\tilde{\mathbf{R}} = \Delta t_k \mathbf{R}$.

We introduce two different strategies for computing the cost gradient and constraint Jacobian (and optionally Hessian) matrices. The discussion focuses on first order derivatives with central differences as they are most relevant in the context of MPC with BFGS Hessian approximation. However, the HG formulation extends to finite difference approximations of higher order. Both strategies include a preparation phase that sets up the HG structure, establishes edge connectivity and collects strategy-related information. Vertices are associated with a unique range of indices that map to the cost gradient and constraint Jacobian columns. The Jacobian matrices are represented in sparse matrix format, e.g., compressed column/row format, or as triplet list as required for IPOPT (see section IV). Accordingly, edges are tagged with their equality and inequality Jacobian row indices. The second phase is the actual solution phase in which new derivatives are computed at each iteration of the solution parameters. Algorithm 1 presents the edge-based variant which facilitates the incorporation of user-defined block Jacobians directly added to the overall matrix rather than triggering central differences for computing edge block Jacobians. On the other hand, the vertex-based strategy in Algorithm 2 requires the vertices to maintain a list of connected edges. This list is generated within the preparation phase. The vertex-based strategy facilitates the parallel processing of connected edges.

IV. Comparative Analysis and Evaluation

The evaluation and comparative performance analysis, w.r.t. CasADi (v. 3.3.0), is carried out on two common

MPC benchmark problems. Rather than comparing individual gradient and Jacobian CPU times, computation times are compared for overall optimization run (open-loop) for otherwise identical solver parameters. The analysis takes into account that CasADi provides exact derivatives, whereas the proposed HG approaches operate wit approximate derivatives. The OCPs are solved with the established C++ interior-point solver IPOPT [20] and HSL-MA27 as internal linear solver [21] (Ubuntu, 3.4 GHz Intel i7). Optimizations terminate upon convergence with a relative tolerance of 10^{-4}. The correct convergence is confirmed by testing the solutions for equality within the tolerance. CasADi provides two modes, SX and MX respectively. SX is intended for fast computation whereas MX focuses on memory efficiency.

Control of the Van-der-Pol oscillator is an established benchmark problem:

$$\dot{\mathbf{x}} = \mathbf{f}(\mathbf{x}, u) = [\dot{x}, \quad -(x^2-1)\dot{x} - x + u]^\intercal.$$

from $\mathbf{x}_s = [0,0]^\intercal$ to $\mathbf{x}_{\text{ref}} = [1,0]^\intercal$ while adhering to $|u| \leq 1$ and minimizing a quadratic objective with $\mathbf{Q} = \text{diag}([1.5, 0.5])$ and $R = 0.1$. The temporal grid is defined uniformly with $\Delta t = 0.1$. The second benchmark problem is to balance the cart-pole system:

$$\ddot{x} = \frac{l m_p \sin(\phi)\dot{\phi}^2 + u + m_p g \cos(\phi)\sin(\phi)}{m_c + m_p\big(1-\cos^2(\phi)\big)}, \tag{5}$$

$$\ddot{\phi} = -\frac{l m_p \cos(\phi)\sin(\phi)\dot{\phi}^2 + u\cos(\phi) + (m_c+m_p) g \sin(\phi)}{l m_c + l m_p\big(1-\cos^2(\phi)\big)}$$

with a fourth order state space model with states $\mathbf{x} = [x, \phi, \dot{x}, \dot{\phi}]^\intercal$. The task is to upswing the pendulum from the lower stable equilibrium $\mathbf{x}_s = [0,0,0,0]^\intercal$ to the upper equilibrium $\mathbf{x}_{\text{ref}} = [-0.5, \pi, 0, 0]^\intercal$ and to stabilize it there. It is $\mathbf{Q} = \text{diag}([2, 10, 0.25, 0.5])$, $R = 0.1$ and $\Delta t = 0.01$. Dynamic parameters are taken from [18]. For both benchmarks, the terminal state cost is set to $V_f(\mathbf{x}_N) = \|\mathbf{\Delta x}\|^2_{\mathbf{Q}}$ and the terminal constraint is omitted.

The CPU time is partitioned into the preparation time T_p and the solving time T_s according to section III. For CasADi, the preparation time includes the complete AD-graph generation, the solving time involves the numerical evaluation, in particular the invocation of IPOPT. Table I shows the mean computation times over 100 repetitions for both benchmarks with increasing horizon lengths N. The results indicate that methods which exploit sparsity such as HG and AD clearly outperform the dense computation in case of multiple-shooting and collocation. Notice, since IPOPT is a sparse solver, dense matrix operations automatically imply an additional overhead. For single-shooting, the HG offers no advantages, as the cost function is monolithic w.r.t. the parameters (Fig. 1). However, AD already exploits the structure on the function graph level even for single shooting. AD is in general slightly faster in terms of the actual solving time, but requires a substantial and non-negligible preparation time. This observation favors AD in case of MPC problems with static structure. The overhead of setting up the structure is more than compensated by the efficiency of solving the problem of identical structure repeatedly in MPC closed-loop control.

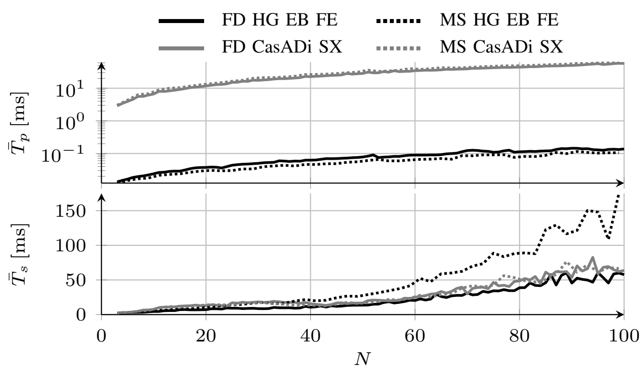


Fig. 5: Cart-pole benchmark results of selected approaches

In numerous MPC applications, the problem structure is varying, as the number of parameters, the number of cost terms or their connectivity is adopted during closed-loop control. MPC with non-static structures accounts for variable horizon length and dynamic, possibly non-uniform shooting grid partitions such as time-optimal MPC approaches [9], [22]. Robot motion planning in dynamic environments is an example in which cost and constraint terms emerge and disappear during runtime, as dynamic obstacles enter the workspace. In these cases, HG is more efficient as the setup time becomes as or even more relevant than the computation time itself. Table I indicates, that the relative advantage of from AD w.r.t. T_s diminishes in case of the sparse structure of full discretization) compared to medium sparsity of multiple-shooting. For short horizon lengths N, HG sometimes even outperforms AD on the computation time. Fig. 5 compares the setup and computation times w.r.t. problem size for HG and AD in case of full discretization (FD) and multiple-shooting (HD). Whereas computation times are similar for low and mid horizon lengths, the setup time of HG is two orders of magnitude lower.

V. Conclusions

The benchmarking of computational effort confirms that sparsity exploitation (HG or AD) in MPC is substantially more efficient than a dense approach. The comparative analysis reveals that in general the time to solve the NLP in the HG formulation is inferior but comparable to AD. However, the preparation phase of the HG is about two order of magnitude faster than AD. Consequently, the HG formulation is therefore preferred for MPC problems which structure and number of parameters and cost terms varies at runtime. Both AD and HG facilitate the implementation of MPC solvers as the programmer merely specifies the cost and constraint functions[1]. AD automatically determines the exact derivatives whereas HG operates with central differences without noticeable degradation in convergence. Future work investigates symbolic computations in the HG framework to support automatic code-generation similar to AD.

[1]We plan to share the generic HG C++ framework in Q4 2018.

TABLE I: Benchmark results for single-shooting (SS), multiple-shooting with 2 controls per interval (MS), full-discretization (FD) and Hermite-Simpson collocation (HS); shooting with forward Euler (FE) and 5th-order Runge-Kutta (RK5) integration.

		Van-der-Pol Oscillator						**Cart-Pole System**					
		$N = 5$		$N = 45$		$N = 85$		$N = 5$		$N = 45$		$N = 85$	
		$\bar{T}_p$ [ms]	$\bar{T}_s$ [ms]	$\bar{T}_p$ [ms]	$\bar{T}_s$ [ms]	$\bar{T}_p$ [ms]	$\bar{T}_s$ [ms]	$\bar{T}_p$ [ms]	$\bar{T}_s$ [ms]	$\bar{T}_p$ [ms]	$\bar{T}_s$ [ms]	$\bar{T}_p$ [ms]	$\bar{T}_s$ [ms]
SS FE	Dense	–	2.11	–	42.78	–	283.08	–	1.33	–	27.48	–	397.49
	HG EB	**0.01**	2.08	**0.02**	37.47	**0.03**	250.37	**0.01**	1.29	**0.02**	25.37	**0.03**	362.54
	HG VB	0.01	**2.06**	0.03	38.33	0.04	258.06	0.01	**1.28**	0.02	26.48	0.04	378.83
	CasADi SX	2.47	2.94	10.70	**13.04**	19.36	**34.21**	2.98	3.03	19.76	**6.40**	37.36	**40.66**
	CasADi MX	4.10	3.25	26.94	22.88	50.71	77.02	6.90	3.66	63.90	15.03	121.90	116.41
SS RK5	Dense	–	3.49	–	220.58	–	1561.79	–	2.18	–	140.26	–	2037.26
	HG EB	**0.01**	3.40	**0.02**	221.03	**0.03**	1561.86	**0.01**	2.15	**0.02**	138.96	**0.03**	2014.99
	HG VB	0.01	3.37	0.02	223.11	0.04	1574.47	0.01	**2.13**	0.02	140.31	0.03	2036.13
	CasADi SX	2.24	**2.98**	10.03	**13.00**	17.61	**34.19**	2.96	2.97	19.25	**6.29**	35.87	**39.62**
	CasADi MX	4.33	3.22	26.92	23.10	50.45	77.23	7.16	3.70	64.97	15.03	122.33	116.45
MS FE	Dense	–	2.90	–	127.60	–	615.32	–	2.88	–	278.85	–	2681.32
	HG EB	**0.02**	**2.69**	**0.06**	13.55	**0.10**	29.20	**0.01**	3.04	**0.05**	25.22	**0.08**	123.53
	HG VB	0.02	2.76	0.08	14.07	0.14	29.62	0.02	**2.58**	0.06	25.49	0.10	125.59
	CasADi SX	3.05	3.29	14.34	**11.74**	24.77	**14.22**	4.36	3.06	29.33	**15.14**	51.93	**50.52**
	CasADi MX	8.27	4.20	71.27	30.84	145.24	50.25	18.49	4.86	192.44	60.26	407.96	284.00
MS RK5	Dense	–	5.47	–	650.09	–	2506.54	–	7.54	–	1326.15	–	13 810.81
	HG EB	**0.01**	4.21	**0.05**	53.35	**0.08**	102.27	**0.01**	5.19	**0.05**	106.79	**0.08**	600.42
	HG VB	0.02	4.30	0.06	53.98	0.10	103.34	0.02	5.19	0.06	107.44	0.10	592.04
	CasADi SX	2.78	**3.20**	12.80	**11.24**	22.76	**14.27**	4.25	**2.97**	27.42	**15.03**	50.64	**49.67**
	CasADi MX	8.42	4.22	71.43	31.07	145.43	50.53	18.77	4.88	193.54	60.29	408.30	284.29
FD	Dense	–	2.77	–	106.73	–	489.39	–	3.47	–	402.58	–	5136.10
	HG EB	**0.02**	2.52	**0.09**	**8.58**	**0.14**	**11.63**	**0.02**	2.75	**0.07**	**12.30**	**0.11**	**46.84**
	HG VB	0.02	**2.51**	0.12	8.61	0.17	11.77	0.02	**2.74**	0.09	12.83	0.16	48.35
	CasADi SX	2.91	3.36	12.46	9.98	22.45	15.36	4.06	3.13	25.06	17.05	47.80	60.50
	CasADi MX	6.09	3.83	51.53	18.16	112.31	35.47	11.94	3.94	144.46	40.66	345.94	197.04
HS	Dense	–	5.39	–	482.11	–	2294.94	–	7.75	–	1258.13	–	17 234.41
	HG EB	**0.02**	3.78	**0.10**	25.71	**0.16**	52.87	**0.02**	5.06	**0.08**	49.04	**0.14**	243.79
	HG VB	0.02	**3.71**	0.12	25.41	0.19	53.31	0.02	**4.95**	0.10	48.82	0.19	251.76
	CasADi SX	4.87	4.74	34.90	**17.83**	62.74	**33.51**	9.54	5.88	83.76	**29.10**	157.89	**138.68**
	CasADi MX	16.71	7.10	186.23	55.72	419.43	130.68	44.28	11.53	626.33	136.18	1526.34	797.26

REFERENCES

[1] M. Morari and J. H. Lee, "Model predictive control: past, present and future," *Computers & Chemical Engineering*, vol. 23, no. 4-5, pp. 667–682, 1999.

[2] M. Diehl, H. G. Bock, J. P. Schlöder, R. Findeisen, Z. Nagy, and F. Allgöwer, "Real-time optimization and nonlinear model predictive control of processes governed by differential-algebraic equations," *Journal of Process Control*, vol. 12, no. 4, pp. 577–585, 2002.

[3] M. Diehl, H. G. Bock, and J. P. Schlöder, "A real-time iteration scheme for nonlinear optimization in optimal feedback control," *SIAM Journal on Control and Optimization*, vol. 43, no. 5, pp. 1714–1736, 2005.

[4] D. Kouzoupis, H. J. Ferreau, and M. Diehl, "First-order methods in embedded nonlinear model predictive control," in *European Control Conference (ECC)*, 2015, pp. 2617–2622.

[5] A. Zanelli, R. Quirynen, G. Frison, and M. Diehl, "A partially tightened real-time iteration scheme for nonlinear model predictive control," in *IEEE Conference on Decision and Control (CDC)*, 2017.

[6] Y. Chen, D. Cuccato, M. Bruschetta, and A. Beghi, "A fast nonlinear model predictive control strategy for real-time motion control of mechanical systems," in *IEEE International Conference on Advanced Intelligent Mechatronics (AIM)*, 2017, pp. 1780–1785.

[7] K. Graichen and B. Käpernick, "A real-time gradient method for nonlinear model predictive control," in *Frontiers of Model Predictive Control*, 2012, ch. 1.

[8] B. Käpernick and K. Graichen, "Nonlinear model predictive control based on constraint transformation," *Optimal Control Applications and Methods*, vol. 37, no. 4, pp. 807–828, 2016.

[9] C. Rösmann, A. Makarow, F. Hoffmann, and T. Bertram, "Sparse shooting at adaptive temporal resolution for time-optimal model predictive control," in *IEEE Conf. on Decision and Control (CDC)*, 2017.

[10] Y. Wang and S. P. Boyd, "Fast model predictive control using online optimization," in *IFAC World Congress*, vol. 17, 2008, pp. 6974–6979.

[11] G. Frison, D. Kouzoupis, J. B. Jørgensen, and M. Diehl, "An efficient implementation of partial condensing for nonlinear model predictive control," in *IEEE Conference on Decision and Control (CDC)*, 2016.

[12] I. Nielsen and D. Axehill, "A parallel structure exploiting factorization algorithm with applications to model predictive control," in *IEEE Conference on Decision and Control (CDC)*, 2015, pp. 3932–3938.

[13] B. Houska, H. Ferreau, and M. Diehl, "ACADO Toolkit – An Open Source Framework for Automatic Control and Dynamic Optimization," *Optimal Control Applications and Methods*, vol. 32, no. 3, 2011.

[14] J. Andersson, "A General-Purpose Software Framework for Dynamic Optimization," PhD thesis, KU Leuven, 2013.

[15] R. Kümmerle, G. Grisetti, H. Strasdat, K. Konolige, and W. Burgard, "G2o: A general framework for graph optimization," in *IEEE International Conference on Robotics and Automation*, 2011, pp. 3607–3613.

[16] C. Rösmann, W. Feiten, T. Wösch, F. Hoffmann, and T. Bertram, "Efficient trajectory optimization using a sparse model," in *European Conference on Mobile Robots*, 2013, pp. 138–143.

[17] D. Q. Mayne, J. B. Rawlings, C. V. Rao, and P. Scokaert, "Constrained model predictive control: Stability and optimality," *Automatica*, vol. 36, no. 6, pp. 789–814, 2000.

[18] M. Kelly, "An introduction to trajectory optimization: How to do your own direct collocation," *SIAM Rev.*, vol. 59, no. 4, pp. 849–904, 2017.

[19] J. Nocedal and S. J. Wright, *Numerical optimization*, ser. Springer series in operations research. New York: Springer, 1999.

[20] A. Wächter and L. T. Biegler, "On the implementation of a primal-dual interior point filter line search algorithm for large-scale nonlinear programming," *Mathematical Programming*, vol. 106, no. 1, 2006.

[21] "HSL. A collection of Fortran codes for large scale scientific computation." [Online]. Available: http://www.hsl.rl.ac.uk/

[22] C. Rösmann, F. Hoffmann, and T. Bertram, "Timed-elastic-bands for time-optimal point-to-point nonlinear model predictive control," in *European Control Conference (ECC)*, 2015, pp. 3357–3362.