Implementation Overview

In the following we give an overview over the implementation of the outer approximation algorithm, devised in [2], in C++, using the DUNE-library [4], for optimal control problems with combinatorial switching constraints of the form

$$\begin{cases} \min \quad J(y,u) = \frac{1}{2}\,\|y - y_{\mathrm{d}}\|^2_{L^2(Q)} + \frac{\alpha}{2}\,\|u - u_{\mathrm{d}}\|^2_{L^2(0,T;\mathbb{R}^n)} \\ \text{s.t.} \quad \partial_t y(t,x) - \Delta y(t,x) = \sum_{j=1}^{n} u_j(t)\,\psi_j(x) + f(t,x) \quad \text{in } Q := \Omega \times (0,T), \\ \qquad\qquad y(t,x) = g \qquad\qquad \text{on } \Gamma_D \times (0,T), \\ \qquad\qquad \partial_\nu y = j \qquad\qquad \text{on } \Gamma_N \times (0,T), \\ \qquad\qquad y(0,x) = y_0(x) \qquad\qquad \text{in } \Omega, \\ \text{and} \quad u \in D, \end{cases} \tag{P}$$

where D is the set of feasible switching patterns. The header files can be found in **dune/MIOCP** and the source codes can be found in **src**.

outerapprox.hh implements the outer approximation algorithm and the constructor requires

(1) a structure containing all needed parameter settings for the outer approximation algorithm as well as the spatial grid function space (see **Parameter.hh**),

(2) the solution Σf of

$$\begin{aligned} \partial_t \zeta - \Delta \zeta &= f && \text{in } Q := \Omega \times (0,T), \\ \zeta &= g && \text{on } \Gamma_D \times (0,T), \\ \partial_\nu \zeta &= j && \text{on } \Gamma_N \times (0,T), \\ \zeta(0) &= y_0 && \text{in } \Omega, \end{aligned}$$

(3) the feasible switching pattern D, and

(4) a structure to output the results of the iterations (see **Parameter.hh**).

In the source code **master-oa.cc**, the environment for the experiments in Secion 5 in [2], you can find an example how to set up all the required data. Hereby, the ini file **master.ini** is used to specify

- the domain $\Omega = [0,1]^2$,
- the number $N_{x_i} = 30$, $i = 1, 2$, of nodes for the spatial grid,
- the end time $T = 2$,
- the number $N_t = 100$ of equidistant time intervals,
- the Tikhonov parameter $\alpha = 0.01$, and
- the folder name where to output the results.

The ini file is read by a Dune::ParameterTreeParser. To run the outer approximation algorithm call the method *apply* providing

- the function $\Psi^*\Sigma^*(Sf - y_{\mathrm{d}}) \in L^2(0,T;\mathbb{R}^n)$, where Ψ^* is the adjoint of the operator

$$\Psi \colon L^2(0,T;\mathbb{R}^n) \to L^2(0,T;H^{-1}(\Omega)), \quad (\Psi u)(t) = \sum_{j=1}^{n} u_j(t)\psi_j$$

given by

$$\Psi^* : L^2(0,T;H_0^1(\Omega)) \to L^2(0,T;\mathbb{R}^n),$$
$$(\Psi^* w)(t) = \Big(\langle \psi_j, w(t)\rangle_{H^{-1}(\Omega),H_0^1(\Omega)}\Big)_{j=1}^n \quad \text{f.a.a. } t \in (0,T),$$

- a start control u, and
- a folder name where to output the solutions u^i of each iteration.

The semi-smooth Newton method is implemented in **activeset.hh** and is called by the outer approximation algorithm in each iteration. **linsolver.hh** contains the linear solver for the semi-smooth Newton system, which specifies the linear operator occurring in the semi-smooth Newton system and the inverse operator of the preconditioner P for the MINRESSolver. Additionally, it extends the CGSolver and MINRESSolver, implemented in DUNE, by the additional stopping criteria that the residual is less than a certain tolerance (in experiments 10^{-8}).

(1,6) **Parameter.hh** contains the structure *Param* providing all parameter settings and the grid function space for the outer approximation algorithm, which has the template variables

- GridView - view on the spatial grid,
- Constraint - type of the constraints, e.g. conforming Dirichlet constraints,
- Problem - class providing the control u and the form functions Ψ_j, $1 \le j \le n$ occurring in the heat equation

$$\begin{aligned} \partial_t y - \Delta y &= \sum_{j=1}^n u_j(t)\,\psi_j(x) && \text{in } Q := \Omega \times (0,T), \\ y &= 0 && \text{on } \Gamma_D \times (0,T), \\ \partial_\nu y &= 0 && \text{on } \Gamma_N \times (0,T), \\ y(0) &= 0 && \text{in } \Omega, \end{aligned} \tag{1}$$

  with homogeneous initial condition and boundary constraints,
- k - degree of spatial ansatz and test functions, e.g., $k = 1$ for piecewise linear functionals, and
- n - number of heat sources.

Note that the temporal discretization for the controls u and the state y is fixed. For y, continuous and piecewise linear functionals in time are chosen, while for u constant functionals are used. The constructor of *Param* requires

- a spatial grid,
- a class derived from *HeatProblemInterface<GridView,n>* in **probleminterface.hh** providing box constraints $u_a \le u \le u_b$ on the control, the form functions Ψ_j, $1 \le j \le n$ and the functional u_d,
- a class derived from *AdjointProblemInterface<GridView>* in **probleminterface.hh** providing the desired temperature y_d,
- the Tikhonov parameter α,
- the parameter ρ (in experiments 10^{-5}) to determine active cutting planes,
- (a) the end time T and the number N_t of time intervals for the temporal discretization, or
  (b) the vector of time interval lengths

- the maximum number of iterations of the semi-smooth Newton method,
- the maximum number of iterations of the linear solver of the semi-smooth Newton system,
- the absolute tolerance (in experiments 10^{-8}) for the linear solver of the semi-smooth Newton system,
- the reduction factor for the linear solver of the semi-smooth Newton system,
- the maximum number of iterations of the outer approximation algorithm,
- a time limit for the outer approximation algorithm, and
- a boolean to indicate, whether reoptimization is desired (by default true).

An example how to set up the spatial grid and to specify the problem classes can be found in **master-oa.cc**. Note that in **master-oa.cc** a uniform spatial triangulation of the domain Ω is implemented by using *UGGrid* as grid type and *createSimplexGrid* to set up the grid. **Parameter.hh** also contains the structure *OutputData* to store results of the outer approximation algorithm.

probleminterface.hh contains the abstract class *HeatProblemInterface<GridView,n>*, which provides the right hand side f, the form functions Ψ_j, $1 \leq j \leq n$, the control u_d, the boundary conditions and the initial function y_0 of (P). It also provides the box constraints $u_a \leq u \leq u_b$ on the control explicitly given by the set D of feasible switching patterns. *CGProblemInterface<GridView,n>* is an abstract class providing a control u and form functions Ψ_j, $1 \leq j \leq n$, occurring in the heat equation (3) with homogeneous initial condition and boundary constraints. An interface for backward-in-time problems of the form (4) is given by *AdjointProblemInterface<GridView>*. Given a solution y of a heat equation, *Adjoint<GFS>* is an interface for the backward-in-time problem

$$\begin{aligned} -\partial_t p - \Delta p &= y && \text{in } Q := \Omega \times (0,T), \\ p &= 0 && \text{on } \Gamma_D \times (0,T), \\ \partial_\nu p &= 0 && \text{on } \Gamma_N \times (0,T), \\ p(T) &= 0 && \text{in } \Omega. \end{aligned} \tag{2}$$

Hereby, the template variable *GFS* is the type of the grid function space on which y was calculated.

(2) **heatdriver.hh** provides a scheme to solve equations of the form

$$\begin{aligned} \partial_t y - \Delta y &= q && \text{in } Q := \Omega \times (0,T), \\ y &= g && \text{on } \Gamma_D \times (0,T), \\ \partial_\nu y &= j && \text{on } \Gamma_N \times (0,T), \\ y(0) &= y_0 && \text{in } \Omega. \end{aligned} \tag{3}$$

and needs

- the spatial grid function space,
- a class derived from *HeatProblemInterface<GridView,n>* or *CGProblemInterface<GridView,n>* in **probleminterface.hh** providing the boundary data, the right hand side and the initial data,
- the vector of time interval lengths, and
- a vector to store the solution y.

The spatial and temporal local operator of the equation are specified in **linearheatfem.hh**.

(3) **adjointdriver.hh** provides a scheme to solve backward-in-time problems of the form

$$\begin{aligned} -\partial_t p - \Delta p &= q && \text{in } Q := \Omega \times (0,T), \\ p &= g && \text{on } \Gamma_D \times (0,T), \\ \partial_\nu p &= j && \text{on } \Gamma_N \times (0,T), \\ p(T) &= p_T && \text{in } \Omega. \end{aligned} \tag{4}$$

and needs

- the spatial grid function space,
- a class derived from *AdjointProblemInterface<GridView>* in **probleminterface.hh** providing the boundary data, the right hand side and the end data, and
- a vector to store the solution p.

The temporal local operator of the equation is specified in **adjointheatfem.hh** and **master-oa.cc** calls the function for the calculation of $\Sigma^* y_\mathrm{d}$.

(3,4) **hfunctions.hh** specifies the adjoint operator $\Psi^* : L^2(0,T;H_0^1(\Omega)) \to L^2(0,T;\mathbb{R}^n)$,

$$(\Psi^* w)(t) = \Big(\langle \psi_j, w(t) \rangle_{H^{-1}(\Omega),H_0^1(\Omega)} \Big)_{j=1}^n \quad \text{f.a.a. } t \in (0,T),$$

and $\chi_{\mathcal{I}} \Psi^*$, where $\chi_{\mathcal{I}}$ is the characteristic function mapping from $L^2(0,T;\mathbb{R}^n)$ to $L^2(I;\mathbb{R}^n)$. The adjoint operator is used in **master-oa.cc** for the calculation of $\Psi^* \Sigma^* y_\mathrm{d}$. Moreover, it contains a function to calculate the objective term $\frac{1}{2}\|y - y_d\|^2_{L^2(Q)}$.

(5) The abstract class *SwitchPoly* in **switchconstr.hh** is a general interface for the set D of feasible switching patterns. For an upper bound $\sigma_{\max}$ on the number of switchings of a single switch, the calculation of a most cutting plane and a linear optimization algorithm are implemented in **Dmax.cc**. Hereby, a control is considered as feasible as soon as the violation of the most violated cutting plane falls below 1% of the right hand side.

For the instances in Section 5 in [2], one can recalculate the objective value of a control with a finer temporal discretization using the perl script **robj.pl**. For that, one needs to write the jumps points $0 < t_1 < t_2 < \cdots < t_{11} < T$ of the instance (see, the files *NX30x30_step100.txt* in results/outerapprox) and the finer temporal discretization N_t ($N_t = 400$ used for results) in the ini file and then to pass a file *cut_*.txt*, in which the control is written, to the perl script.

The environment for the experiments in Section 4 in [1] is given in **master-gurobi.cc**. Again an ini file **master-gurobi.ini** is used to specify

- the domain $\Omega = [0,1]^2$,
- the number N_{x_i}, $i = 1, 2$, of nodes for the spatial grid,
- the end time $T = 2$,
- the number N_t of equidistant time intervals, and
- the folder name where to output the results.

The implementation based on the Gurobi solver [3] can be found in **gurobi.hh**. The template variables of *Gurobi* are:

- GridView - view on the spatial grid,
- Constraint - type of the constraints, e.g. conforming Dirichlet constraints,
- Problem - class providing the control u and the form functions Ψ_j, $1 \leq j \leq n$ occurring in the heat equation (1) with homogeneous initial condition and boundary constraints, and
- k - degree of spatial ansatz and test functions, e.g., $k = 1$ for piecewise linear functionals.

To construct an object of *Gurobi* one needs to provide

- the spatial grid on which the problem is solved,
- the vector of time interval lengths for the temporal grid on which the problem is solved,
- class derived from *HeatProblemInterface<GridView,n>* in **probleminterface.hh** providing box constraints $u_a \leq u \leq u_b$ on the control, the form functions Ψ_j, $1 \leq j \leq n$ and the functional u_d,
- a class derived from *AdjointProblemInterface<GridView>* in **probleminterface.hh** providing the desired temperature y_d,
- a spatial grid on which the objective value is recalculated, and
- (a) a number N_t if time intervals or
  (b) a vector of time interval lengths
  for the temporal grid on which the objective value is recalculated.

The discretization of (P), as described in [1], can be written as

$$
\left\{
\begin{aligned}
\min \quad & \sum_{k=0}^{N_t-1} \tfrac{1}{2}\tau_k \Big[(y^k - y_\mathrm{d}^k)^\top M (y^k - y_\mathrm{d}^k) \\
& \qquad + \tfrac{1}{3}(y^{k+1} - y^k - y_\mathrm{d}^{k+1} + y_\mathrm{d}^k)^\top M (y^{k+1} - y^k - y_\mathrm{d}^{k+1} + y_\mathrm{d}^k) \\
& \qquad + (y^k - y_\mathrm{d}^k)^\top M (y^{k+1} - y^k - y_\mathrm{d}^{k+1} + y_\mathrm{d}^k) \Big] \\
\text{s.t.} \quad & M(y^{k+1} - y^k) + \tau_k A y^{k+1} = \tau_k u_k M\psi + \tau M f^{k+1} \quad \forall k = 0, \ldots, N_t - 1 \\
& y^0 = y_0, \\
& u_0 + \sum_{k=1}^{N_t-1} |u_k - u_{k-1}| \leq \sigma_\mathrm{max}, \\
& u_k \in \{0,1\} \quad \forall k = 0, \ldots, N_t - 1.
\end{aligned}
\right. \tag{D}
$$

where M denotes the mass matrix, A the stiffness matrix, τ_k the length of k-th time interval, ψ the coordinate vector of the function $\psi(x)$ with respect to the spatial discretization, y_0 the coordinate vector of $y_0(x)$ and f^k the coordinate vector of $f(t_k, x)$ for $k = 1, \ldots N_t - 1$. The class *PDEElements< GFS, k >* provides all the data of the discretized problem (D) and is called by the constructor of the class *Gurobi*.

The constraint $u_0 + \sum_{k=1}^{N_t-1} |u_{k+1} - u_k| \leq \sigma_\mathrm{max}$ in (D) is linearized by introducing $N_t - 1$ auxiliary variables z_k expressing the absolute values $|u_k - u_{k-1}|$. More precisely, one requires $z_k \geq u_k - u_{k-1}$ and $z_k \geq u_{k-1} - u_k$ and use the linear constraint $u_0 + \sum_{k=1}^{N_t-1} z_k \leq \sigma_\mathrm{max}$ instead.

The function *optimize* of the class *Gurobi* set ups the Gurobi model of (D), when using mode 0, and solves it by calling the MINLP solver Gurobi 9.1.2 [3]. When using mode 1, the naive convex relaxation, which replaces the binarity constraint $u_k \in \{0,1\}$ with $u_k \in [0,1]$ for $k = 0, \ldots, N_t - 1$, is solved. Hereby, the function *callback* in **callback.cc** is used to call the separation algorithm in **gurobi-Dmax.cc**, which calculates a most violated cutting plane for the convex hull of feasible switching patterns in (D).

Literatur

[1] C. Buchheim, A. Grütering, and C. Meyer, *Parabolic optimal control problems with combinatorial switching constraints – Part I: Convex relaxations*, arXiv preprint arXiv:2203.07121, (2022).

[2] C. Buchheim, A. Grütering, and C. Meyer, *Parabolic optimal control problems with combinatorial switching constraints – Part II: Outer approximation algorithm*, arXiv preprint arXiv:2204.07008, (2022).

[3] Gurobi Optimization, LLC, *Gurobi Optimizer Reference Manual*, 2021, `https://www.gurobi.com`.

[4] O. Sander, *DUNE—The Distributed and Unified Numerics Environment*, vol. 140, Springer Nature, 2021.