Implementation Overview

In the following we give an overview over the implementation of branch-and-bound algorithm in C++, using the DUNE-library [1], 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** and the source codes can be found in **src**. Hereby, we divide into classes for the branch-and-bound algorithm (**../bnb**) and for the solution of the relaxations of each subproblem (**../OCP**).

Classes for the branch-and-bound algorithm

The basic framework of a branch-and-bound algorithm is implemented in the header files **subproblem.h, master.h, branching.h** and **enumeration.h**. In addition, **breadthfirst.h, bestfirst.h** and **depthfirst.h** specifies different enumeration strategies.

The abstract class **OCPMaster.hh** for the master problem of (P) in particular contains the outer approximation algorithm to solve each subproblem, the DWR error estimator to correct the time-mesh dependent lower bounds of the subproblems, if necessary, and the adaptive refinement strategy. Some helper functions to calculate the DWR error estimator are in **errorfunctions.hh**. However, the switching constraints specific member functions *int separate(·), void heuristics(·)* and *int updateFixControls(·)* of the master problem for the separation of an infeasible control, the calculation of a heuristic solution, and for the update of the implicitly fixations of the control by the fixings in the subproblem, need to be specified. For the case of an upper bound on the total number of switchings of a single switch, i.e.

$$D = \{u \in BV(0,T) : u(t) \in \{0,1\} \text{ a.e. in } (0,T),\ |u|_{BV(0,T)} \leq \sigma_{\max}\}\ ,$$

their specification can be found in **OCPDmax.hh** and **OCPDmax.cc**, respectively. In order to start the branch-and-bound algorithm for (P), one needs to generate a master problem, for instance of type *OCPDmax*, whose template variables 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

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

with homogeneous initial condition and boundary constraints,

- k - degree of spatial ansatz and test functions, e.g., $k = 1$ for piecewise linear functionals, and
- number - number of switches.

Note that the temporal discretization for the controls u and the state y is fixed to piecewise constant functions. The constructor requires

- a spatial grid,
- a 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,
- the temporal grid of y_d (which is assumed to be piecewise constant in time),
- the feasible switching pattern D,
- the parameters α, ρ (penalty term of cutting planes), β (penalty term of box constraints),
- the temporal grid for the root node problem,
- the maximum iterations of the conjugate gradient solver in the ADMM algorithm,
- the reductions factor for the conjugate gradient solver in the ADMM algorihm,
- the tolerances ϵ^{abs}, ϵ^{rel}, $\varepsilon^{\mathrm{abs}}$ for the ADMM algorithm,
- the percentage $\gamma > 0$ of cells to be refined (by default 50%),
- the minimum relative change of the objective in the outer approximation algorithm (by default 1%),
- the relative exactness of the branch-and-bound algorithm (by default 1%),
- the maximum number of outer approximation iterations (by default 100), the maximum time for the outer approximation (by default 1800 sec.), and
- the number how often the subproblems are output to the console (by default 1).

By default, the depthfirst enumeration strategy is used and can be changed with the help of *void setEnumerationStrategy(·)*. The default branching strategy can be found in **OCPbranch.hh** and can be changed by *void setBranchingStrategy(·)*. In the source code **BnBTest.cc** you can find an example to set up all the required data for an object of the master class *OCPDmax*, especially the spatial grid for the case $\Omega \subseteq \mathbb{R}$ by using *OneDGrid* as grid type. Moreover, the default enumeration strategy is changed to the breadthfirst strategy. After generating the master problem, one cal call the function *void writeParameterSetting(·)* to save the parameter setting into a file and then call *void optimize(·)* to start the branch-and-bound algorithm. In the *CMakeLists* file in the source code folder **src**, one may specify the *OUTPUTDIR* flag, which states the folder name to output the parameter settings of the master problem, and add the flag *BNBTREE*, if the whole branch-and-bound tree should be output to file.

The class **OCPSub.hh** for the subproblems of (P) contains all relevant information such as the fixings and the added cutting planes. When a new subproblem is generated, the data of the parent node is copied in order to warm start the outer approximation of the subproblem.

Classes to solve the OCP relaxations

The ADMM algorithm to solve the tailored convexification of the subproblems is implemented in **ADMMsolver.hh** and is called by the master problem within the outer approximation. The linear operator occuring in the ADMM algorithm to determine the next control is specified in **ADMMsystem.hh**, as well as the inverse operator of the preconditioner $P = (\alpha + \beta)I + \rho G^\star G$.

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 with homogeneous initial condition and boundary constraints. An interface for backward-in-time problems of the form () is given by *AdjointProblemInterface<GridView>*. Given a solution y of a heat equation, *Adjoint<GFS>* is an interface for the backward-in-time problem

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

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

The abstract class *SwitchPoly* in **switchconstr.hh** is a general interface for the set D of feasible switching patterns. For an upper bound σ_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.

heatdriver.hh provides a scheme to solve equations of the form

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

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

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

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

The temporal local operator of the adjoint equation are specified in **adjointheatfem.hh**.

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 contains helper functions to extend controls or states to the joint temporal grid of a certain control and the desired state y_d.

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