Combining ML and Mathematical Optimization to tackle automatic parameter tuning on HUC problems

Claudia D'Ambrosio, Antonio Frangioni, Gabriele Iommazzo, Leo Liberti

École Polytechnique, Università di Pisa

Data Science Summer School, École Polytechnique, 28 August – 1 September 2017

Introduction

Solving a mathematical optimization problem involves using an INSTANCE of the problem and an abstract FORMULATION – written in a formal mathematical programming language – as inputs for a SOLVER, and then retrieving the SOLUTIONS produced by the solver itself.

Solvers' *default* parameter values are tested by the vendors to have acceptable average performance on a huge testbed of cases. On the other hand, they can yield poor performances when the instances pertain to a *specific* class of problems. Thus, solvers need **tuning**, to select appropriate parameter values.

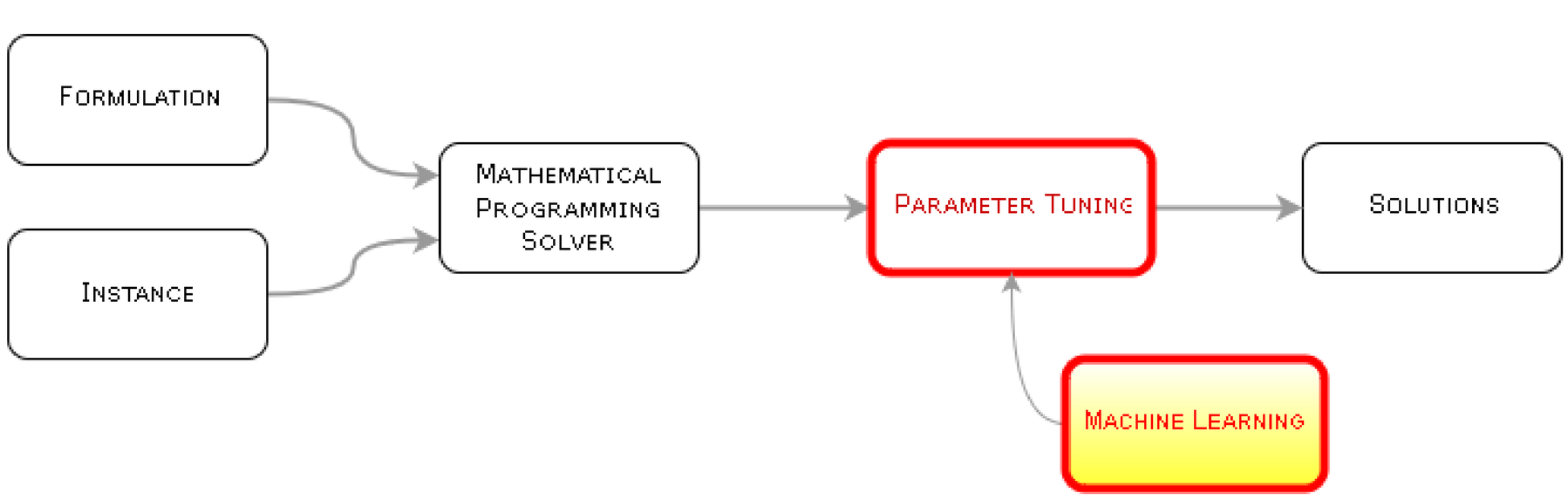

Figure 1: Automating the parameter tuning problem on a mathematical optimization solver

Manual tuning is the norm. But it turns out to be a possibly inaccurate and time-consuming process, even for experienced users. **Supervised Machine Learning** techniques can be used to automate the parameter tuning approach and, upon receiving an instance, recommend the parameter configurations that are best suited for that instance.

The approach: Phase 1

Create training set

Run the solver to produce a training set $\mathcal{S} = \{(f_i, c_i), y_i, \}$, s.t.

- f_i: vector of features representing each instance of a Hydro Unit Commitment Problem (in France)
- c_i: vector of solver's parameter values, i.e. a parameter configuration
- y_i: evaluation measure for a run of the solver (within a fixed time limit), executed on instance f with parameter setting c, i.e.

$$\frac{UpperBound\ sol. - LowerBound\ sol.}{LowerBound\ sol.}$$

The approach: Phase 2

Supervised training

Given a target function p to evaluate the performance of a pair (f, c),

$$p : \mathcal{F} \times \mathcal{C} \longrightarrow \mathbb{R}, \quad p(f, c) = y, \qquad (1)$$

build a model $\bar{p}$ that approximates p via supervised ML, trained on $\mathcal{S}$. $\mathcal{F}$ is the space of all feature vectors and $\mathcal{C}$ is the space of all possible parameter settings.

The approach: Phase 3

Mathematical Optimization

When a new instance has to be solved (its features $\bar{f}$ are given, and thus deemed as **fixed**), we aim to find the optimal configuration $c^* \in C$ that minimizes $\bar{p}(\bar{f}, \cdot)$, as this is our best guess at the configuration that minimizes $p(\bar{f}, \cdot)$. Therefore, we want to solve the problem

$$c^* \in \text{argmin} \{ \ \bar{p}(\bar{f}, c) \ : \ c \in \mathcal{C} \} \qquad (2)$$

Observations

We expect C to be in the several tens and possibly with a strong combinatorial structure. Moreover, (2) will have to be solved quickly enough so that the benefit of finding c^* is not negated by the effort required to find it. Keeping that in mind, the chosen ML technique shall be able to achieve an acceptable compromise between building a good model $\bar{p}(\bar{f}, \cdot)$ of $p(\bar{f}, \cdot)$, and solving (2) efficiently enough.

Support Vector Regression (SVR)

SVR is attractive for two reasons: it can be written as a structured convex QP, for which countless solution methods exist; its dual formulation leads to the corresponding *kernel trick*, that allows to deal with nonlinearity. With SVR, optimization model (2) becomes

$$c^* \in \text{argmin} \left\{ \ \sum_{i \in \mathcal{S}} (\bar{\alpha}_i^+ - \bar{\alpha}_i^-) \boldsymbol{\kappa}\big([\bar{\boldsymbol{f}}_i, \bar{\boldsymbol{c}}_i] \, , \, [\bar{\boldsymbol{f}}, \boldsymbol{c}] \big) + b \ : \ c \in \mathcal{C} \ \right\}, \qquad (3)$$

s.t.

- $\bar{\alpha}_i^+$ and $\bar{\alpha}_i^-$ for $i \in \mathcal{S}$: optimal dual multipliers, identified after training the SVR
- $\kappa(\bar{x}_i, x) = \langle \Phi(\bar{x}_i), \Phi(x) \rangle$, $\bar{x}_i = [\bar{f}_i, \bar{c}_i]$: the so-called *kernel* where the arbitrarily nonlinear Φ maps the training points from the *input space* $\mathcal{X} = \mathcal{F} \times \mathcal{C}$ into some arbitrary *feature space*. We chose to employ a Gaussian Kernel $\kappa(\bar{x}_i, x) = e^{-\|\bar{x}_i - x\|_2^2 / 2\sigma^2}$, where $\|\bar{x}_i - x\|_2^2 = \left\| \begin{pmatrix} \bar{f}_i - \bar{f} \\ \bar{c}_i - c \end{pmatrix} \right\|_2^2$
- $\bar{f}_i$ and $\bar{c}_i$ are the support vectors found during the training phase, consequently they are fixed during the optimization phase; $\bar{f}$ are features identifying a new instance, so they are fixed too during the optimization phase.

On choosing a kernel

Too simple a k – in addition to (possibly) only yielding a poor approximation to p – is likely to fail to obtain an optimal solution to (3). This is precisely the case of a *separable* kernel $\kappa\big([\bar{f}_i, \bar{c}_i] \, , \, [\bar{f}, c] \big) = \kappa_f(\bar{f}_i \, , \, \bar{f} \,) + \kappa_c(\bar{c}_i \, , \, c \,)$: here f would only account to a constant in the problem, and could not change its optimal solution. In this situation, it would be hopeless to try and learn relationships between f and c; on the contrary, one would be left to seeking "the best configuration for all possible instances". Hence the kernel choice described before.

École Polytechnique — Università di Pisa