Computational Methods in Economics - Final Project

Ricardo Semião e Castro

2025-04-10

1 Introduction

There is a literature on the use of *hybrid optimization methods* for using two stages to firstly deal with complex aspects of the problem, and then solve the remaining simpler problem with more efficient methods. I'll show that an unusual class of problems – where the domain of the objective function is part of the optimization choice – can be rewritten in a hybrid-aligned format, similar to more common problems. Then, I'll use this framing to motivate a generalist design and code implementation.

I consider hybrid methods, with a random first step, as they provide space for custom operators, possibly based on heuristics. This flexibility is important to solve non-standard problems.

I'll start by describing the problems of interest and rewriting all of them in the same language in section Section 2. With this generalist framing, I'll describe the method's design in section Section 3, implement it in Section 4, and apply it in section Section 5.

2 Theoretical Framework

2.1 Considered Problems and Examples

I'll consider three types of 'bad-behaviors': non-real variables domains (integer, categorical etc.); variables that affect the objective or constraints functions in a complex (non-linear) way; and endogenous domains.

2.1.1 Different Natures of Variables

Also consider variables with separable characteristics: some imply a clear global optimum and others don't – $x + \sin(2y)$; or where some are convex and others aren't – $x^2 + \text{floor}(x)^2$; or some have 'difficult' constraints and others don't.

2.1.2 Non Real-Valued Variables

Consider a central planner choosing lump-sum taxes τ to maximize some social welfare function. The higher the tax, the higher the government's utility, and smaller the population's $i \in \{1, 2, \ldots, N\}$.

$$\max_\tau u(\tau) = U_G(\tau) + \sum_{i=1}^N U_P^i(\tau), \quad u : \mathbb{R}^+ \to \mathbb{R}$$

Let's add difficulty by steps. Consider the discrete nature of money, we have a mixed-integer problem:

$$\max_\tau u(\tau) = U_G(\tau) + \sum_{i=1}^N U_P^i(\tau), \quad u : [0.01, 0.02, \ldots] \to \mathbb{R}$$

Suppose the government can choose different taxes τ_s given each person's group $s_i \in \{A, B\}$. Then, our problem is now mixed with a categorical variable:

$$\begin{aligned} &\max_{\tau,s} u(\tau, s) = U_G(\tau, s) + \sum_{i=1}^N U_P^i(\tau_i(s_i)), \quad u : (\mathbb{R}^+)^2 \times \{A, B\}^N \to \mathbb{R} \\ &\tau_i(s_i) = I\{s_i = A\}\tau_A + I\{s_i = B\}\tau_B \end{aligned}$$

2.1.3 Endogenous Domain

What if the central planner, before choosing τ, must commit to it being in some interval $[\underline{\tau}, \underline{\tau} + c]$, $c \in \mathbb{R}$? Or, maybe, the number of groups can be chosen, such that $s \in \{A, B\}$ or $\{A, B, C\}$, ...?

$$\begin{array}{ll} \max_{\tau,S,\mathcal{S}} u(\tau,S) = U_G(\tau,S) + \sum_{i=1}^{N} U_P^i(\tau_i(S_i)), & u : (\mathbb{R}^+)^2 \times \mathcal{S}^N \to \mathbb{R} \\ \max_{\tau,\underline{\tau}} u(\tau), & u : [\underline{\tau}, \underline{\tau}+c] \to \mathbb{R} \end{array}$$

Additionally, consider the researcher's problem: they have data on τ and $u(\tau;\theta)$ and want to estimate some parameter θ by minimizing some loss function $L(\theta; y - u(\tau;\theta))$. They don't know the scale of the abstract parameter θ but must choose a domain for the computational method of their choosing:

$$\min_{\theta,\Theta} L(\theta; y - u(\tau;\theta)), \quad L : \Theta \to \mathbb{R}$$

While this formulation could be interesting, it is not fundamentally different from using a general domain and creating a new constraint $x \in \{\underline{\tau}, \underline{\tau}+c\}$. The last example didn't have a rule, but I'll show that we can rewrite it too in the same way.

2.2 Reframing the Problems

Consider below the general problem of optimizing a real-value function, with possibly non-real domain. I'll show alterations that can be done to transform it into a generalist problem, aligned with hybrid methods.

$$\max_{x,X} f(x) \ \ s.t. \ \ g(x) \geq 0, \quad f : X \to \mathbb{R}^n, \ g : X \to \mathbb{R}^m, \ X \in \mathcal{X}$$

Where: the domain is endogenous with options in $\mathcal{X}$; $n, m \in \mathbb{N}$ are the dimensions of the result and constraints, respectively; the restrictions can be irrelevant $g(x) = 0$; and not all dimensions of x need be used in both f and g.

First, note that we can always separate, from the full x, the variables $\tilde{x}$ to be worked only on the first step. If some constraints depend only on $\tilde{x}$, we can separate them into $\tilde{g}(\tilde{x})$.

Additionally, we can map the domain options $\mathcal{X}$ into a set of indexes $\tilde{X}$. Then, we can rewrite our problem as choosing the index:

$$\max_{x,\tilde{x}} f(x) \ \ s.t. \ \ g(x) \geq 0, \quad f : X_{\tilde{x}} \to \mathbb{R}^n, \ g : X_{\tilde{x}} \to \mathbb{R}^m, \ \tilde{x} \in \tilde{X}$$

Based on the equivalence below, we can change the domain options $\mathcal{X}$ into restriction options $H_{\mathcal{X}}$. Then, we can trade the domain choosing for the constraint g_{m+1}:

$$\begin{gathered} x \in X \quad \Leftrightarrow \quad h(x) \geq 0, \ h : X = \{x : h(x) \geq 0\} \\ H_{\mathcal{X}} = \{h_{\tilde{x}} : X = \{x : h_{\tilde{x}}(x) \geq 0\}, \ X \in \mathcal{X}\} \\ g_{m+1}(x,\tilde{x}) = \sum_{i \in \tilde{X}} I\{i = \tilde{x}\} h_{\tilde{x}}(x) \end{gathered}$$

Where if $\mathcal{X}$ is uncountable, $\tilde{X} \subseteq \mathbb{R}$ and we would use an integral.

After these manipulations, the reframed problem becomes:

$$\max_{x,\tilde{x}} f(x,\tilde{x}) \ \ s.t. \ \ g(x,\tilde{x}) \geq 0, \ \tilde{g}(\tilde{x}) \geq 0, \quad f : X \to \mathbb{R}^n, \ g : X \times \tilde{X} \to \mathbb{R}^m, \ \tilde{g} : \tilde{X} \to \mathbb{R}^{\tilde{m}}$$

As $X \subseteq \mathbb{R}^k$ is not open and $g : X \to \mathbb{R}^m$ is not differentiable, KKT conditions aren't satisfied. Thus, I'll describe a method for such problem in the next section.

3 Method Design

First, one must define: the number $T \in \mathbb{N}$ of iterations (denoted by t); the number $N \in \mathbb{N}$ of samples (denoted by s) in each population S_t. Denote this step by $S_0 = \text{init}(\tilde{X}, \tilde{g})$.

3.1 First Step: Create Initial Sample of $\tilde{x}$

The first step is to create an initial population of the separated variables $\tilde{x}$, $S_0 = (\tilde{x}_s)_{s=1}^N$. This population must be created to respect the constraints of $\tilde{g}$. Denote this step by $S_0 = \text{initialize}(\tilde{X}, \tilde{g})$.

There are several methods to create an initial sample. They generally can be divided into:

- Sampling $\tilde{x}_s \sim P(.)$ from a distribution $P : \tilde{X} \to [0, 1]$.
  - One can use the uniform distribution, or a heuristic guess based on knowledge of the problem, a-la importance sampling.
- Creating an even grid of points via some rule r $\tilde{x}_s = \inf\{\tilde{X}\} + r(\tilde{X})$.
  - Examples are grid sampling, Latin Hypercube sampling, Kronecker sampling, and Sobol or Halton sequences.
- Additionally, the domain can be split into groups and sampling done within each group (stratified/cluster-based sampling).

The main goal is to guarantee that the algorithm will be able to explore the whole domain. (Kazimipour, Li, and Qin 2014) provides a comprehensive review of initialization methods.

Then, to guarantee validity of the $\tilde{g}$ constraints, there are also several methods:

- Rejecting or repairing samples that break the constraints.
  - Using the closest valid point, some projection into the valid space, amongst others.
- Using heuristics to sample only from the valid space.
  - Sampling each dimension of $\tilde{x}_s$ at a time, updating the valid space of the rest each time; choosing a distribution P that is more likely to sample valid points; amongst others.

Note that if $\tilde{x}$ contains encoded variables, as in the Researcher's Problem, any heuristics used depend on how the encoding was done.

3.2 Second Step: Solve the Reduced Problem

Denote this step by $R_t = \text{optimize}(X, g, t, S_t)$. For a given sample $\tilde{x}_s$, what is left is the reduced problem, that doesn't depend on $\tilde{x}$ nor $\tilde{g}$:

$$\max_x f_s(x;\ \tilde{x}_s) \ \ s.t.\ \ g_s(x;\ \tilde{x}_s) \geq 0, \quad f : X \to \mathbb{R}^n,\ g : X \to \mathbb{R}^m$$

One chooses the solver of their liking to this problem. One can also choose several options, that get randomly picked for each sample, which helps with generality of the solution.

Then, for each $\tilde{x}_s \in S_t$, the problem is solved into x_s^*, and an ordered set of the results - and optional meta-information I_s (e.g.: time to completion) - are stored:

$$O_s = (\tilde{x}_s,\ x_s^*,\ f(x_s^*, \tilde{x}_s),\ I_s), \quad R_t = (O_s)_{s=1}^N$$

Some set of metrics M_t are calculated from R_t. Then, the stopping criteria can be drawn from a combination of options. On top of maximum time elapsed or iterations, one can consider the best/median/other sample value $(\tilde{x}, x)$ or performance $f((\tilde{x}, x))$ (denote by o), via: threshold $o \gtrless k$; convergence $|o_{t'} - o_t| < \epsilon$; or stability $sd(o) < \epsilon$. Let $R = (R_t)_{t=1}^T$ and $M = (M_t)_{t=1}^T$.

3.3 Third Step: Update the Sample

The last step is to update the sample: $S_{t+1} = \text{update}(\tilde{X}, \tilde{g}, t, S_t, R_t)$. The general objective is to create a new population in the "direction" of the best performing samples in S_t. There are many literatures that motivate update operators, and it will be user-supplied in the coding implementation, such that the setup can encompass many options.

For the present explanation, I'll focus on operators that are more easily applied to numeric problems in economics, mainly related to the Evolutionary Algorithms literature. Consider the option of combining random tuples of samples ("crossover"), and then adding some randomization, to avoid local optima ("mutation"). In simplistic terms, a 'child' sample is created by:

- Position crossover: taking some positions/dimensions from each of the 'parents', randomly.
- Arithmetic crossover: combining the parents' values, with averages or else.
- Distributional crossover: sampling from a distribution based on the parents' values.
- Resample mutation: randomly choosing a dimension to resample from its distribution.
- Noise mutation: randomly choosing a dimension to add noise.
- The amount of crossover and mutation, amongst others, can be hyperparameters, which can even depend on the iteration t.

Again, not remotely an exhaustive list. More on crossover and mutation can be found in (Kora and Yadlapalli 2017), (De Falco, Della Cioppa, and Tarantino 2002), but many other random optimization literatures can be considered.

Note that these operators need to account for possibly non-real or non-numeric variables, and also for the constraints $\tilde{g}$. The flexibility of being able to define the operator is a major advantage of the method.

4 Coding Implementation

The coding implementation is done via an R package, `pbhy.optr` (Population-Based Hybrid Optimization in R). It is available on my GitHub. It is unfeasible to explain all the details of the implementation here, but all is explained in the package's documentation.

The main function is `optimize_pbhy()`. It, and all others, have documentation pages with the same math notation as here, and with similar argument names (e.g.: `xtil` for $\tilde{x}$ and `x_dom` for X). The function receives the main arguments:

- The functions `f`, `g`, `gtil`, which must receive the correct arguments `x` and/or `xtil`.
- The domains `x_dom`, `xtil_dom`, lists with length m and $\tilde{m}$. Will be used by the user-supplied operators and their elements can be whatever the user needs.
- The operators `initializer`, `optimizer`, and `updater`, which must receive the arguments described in the previous section, and return an output in a specific format.
- The helper `stopper`, that calculates metrics of the performance at each iteration, and sets the stopping criteria; and `logger`, that logs the performance to the console for user feedback. They are created by the helper functions `flow_stopper()` and `flow_logger()`.

The algorithm is presented below. The user-provided function calls are wrapped in `try_op` function, to catch and notify errors in a user-friendly way. The notation `x\$y` indicates an object y within an object x.

Algorithm 1 Hybrid Optimization Algorithm

```
Require: Helpers were created with flow_*(); $f$, $g$ and $\tilde{g}$ are valid functions
 1: Initialize containers for $R_t$ (encompassing $S_t$), and $M_t$
 2: $S_0 \leftarrow$ try_op(initialize($\tilde{X}, \tilde{g}$))
 3: for $t \in \{1, \ldots,$ stopper$iter_upper$\}$ do        ▷ iter_upper: unfeasible limit
 4:     $R_t \leftarrow$ try_op(optimize($X, g, S_t, t$))        ▷ Optimize
 5:     $M_t \leftarrow$ stopper$get_metrics($R_t$)        ▷ Calculate metrics
 6:     logger$log($R_t, M_t, t$)        ▷ Log results
 7:     if stopper$check_metrics($M_t$) then
 8:         break        ▷ Check stoppage criteria
 9:     end if
10:     $S_{t+1} \leftarrow$ try_op(update($\tilde{X}, \tilde{g}, R_t, t$))        ▷ Update sample
11: end for
12: return $(R, M)$
```

It is very important to read the `example.Rmd` ("Using the Pb-Hybrid Algorithm") vignette, which thoroughly explains the package logic and usage. In its core, it is a wrapper to 'organize' the operators into a two-step hybrid procedure. It is the user that has 'responsibility' for making smart choices on these operators, but the package guarantees (i) thorough tests and error handling to help debugging, and (ii) a very flexible/generalist/agnostic setup, that doesn't restrict the user. The tricks to allow (i) any size of m and $\tilde{m}$; (ii) any kind of operator; (iii) any kind of metric M_t and stopping criteria are specially important.

As one of the goals of the class is good coding practices, I spent lots of effort on the package (more than I should've). To cite some of the features, it:

- Has thorough documentation with follows exactly the notation of this paper, which will be attached to the docs. The argument syntax also follows that notation.
- Follows the tidyverse style guide using the `styler` and `lintr` packages for consistency.
- Near-perfectly (minor some dependency shortcuts I took) passes the R CMD check, a necessary step for CRAN submission.
- Has a website built with `pkgdown` (in the "docs/" folder)[1].

Some improvements that I intend to do:

- Post it in my GitHub (it might be there by the time you correct this).
- The package include pre-done metrics for the `stopper` object. I intent to expand those, and also do pre-done operators for the common optimization problems.
- Add an `on.exit()` call to `optimize_pbhy()`, such that it safely returns what it had before reaching an eventual error.
- The `testthat` package structure is created for the package, but I haven't had the time to write unir tests yet.
- Create `summary()` and `plot()` methods for the `optimize_pbhy()` result.

5 Application

The `example.Rmd` has an application for a specific problem, using a random (uniform) sampler for initialization, a golden search for optimization, and a genetic algorithm's approach of crossover+mutation for the updater.

6 Conclusion

I am very happy with the theoretical framework, algorithm design, and especially the coding implementation. I put lots of effort into a good quality code, with intuitive use, well explained, and with helper tests, functions, and error handling. The flexibility of the algorithm is also a major advantage.

But, I focused too much on that aspect and not enough on the economic motivation. There were few examples at the beginning of this paper, limited explanation for the endogenous domain problem, and only one specific application in the `example.Rmd` vignette, which was not an endogenous domain problem. I got too caught up in the coding implementation and organized my priorities poorly. Still, the setup is very solid, making it easy to add the endogenous domain example.

[1]Because of a known issue in the mathjax rendering, the math notation might not be correctly rendered.

References

De Falco, I, A Della Cioppa, and E Tarantino. 2002. “Mutation-Based Genetic Algorithm: Performance Evaluation.” *Applied Soft Computing* 1 (4): 285–99. https://doi.org/https://doi.org/10.1016/S1568-4946(02)00021-2.

Kazimipour, Borhan, Xiaodong Li, and A. K. Qin. 2014. “A Review of Population Initialization Techniques for Evolutionary Algorithms,” 2585–92. https://doi.org/10.1109/CEC.2014.6900618.

Kora, Padmavathi, and Priyanka Yadlapalli. 2017. “Crossover Operators in Genetic Algorithms: A Review.” *International Journal of Computer Applications* 162 (March): 34–36. https://doi.org/10.5120/ijca2017913370.