Asynchronous surrogate optimization in Python (pySOT + POAP)

David Eriksson

Center for Applied Mathematics
Cornell University

dme65@cornell.edu

June 21, 2016

Outline

Background

- Global optimization problem (GOP)

$$\begin{array}{ll} \text{minimize} & f(x) \\ \text{subject to} & g_i(x) \leq 0, \quad i = 1, \ldots, m \\ & x \in \mathcal{D} \end{array} \tag{1}$$

- $f : \mathbb{R}^d \rightarrow \mathbb{R}$ continuous deterministic expensive black-box function.
- Inequality constraints $g_i(x) \leq 0$, where $g_i : \mathbb{R}^d \rightarrow \mathbb{R}$ for $i = 1, \ldots, m$.
- $D \subset \mathbb{R}^d$ is a hypercube

Background

- Surrogate optimization methods are successful on GOP
- Use a surrogate to approximate the objective function
- Common surrogate models:
  - Radial basis functions (RBFs)
  - Kriging
  - Multivariate adaptive regression splines (MARS)
  - Polynomial regression
- Surrogate optimization methods start by evaluating an experimental design
- The initial surrogate is fitted using these points.

Surrogate optimization

Algorithm 1: Synchronous Surrogate Optimization Algorithm

Input: Optimization problem, Experimental design, Adaptive sampling method, Surrogate model, Stopping criterion, Restart criterion

Output: Best solution and its corresponding function value

```
1  Generate an initial experimental design;
2  Evaluate the points in the experimental design;
3  Build a Surrogate model from the data;
4  repeat
5      if Restart criterion met then
6          Reset the Surrogate model and the Sample point strategy;
7          go to 1;
8      end
9      Use the adaptive sampling method to generate new point(s) to
         evaluate;
10     Evaluate the point(s) generated using all computational resources;
11     Update the Surrogate model;
12 until Stopping criterion met;
```

Surrogate optimization

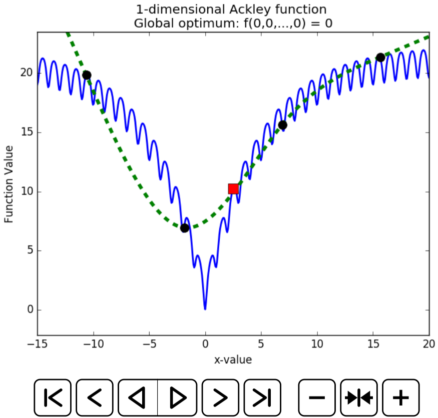

Figure: Animation of a stochastic optimization algorithm. (Solid line) is the objective function, (Dashed line) is the surrogate, (**Circles**) are the old evaluations, (Square) is the new evaluation.

Overview

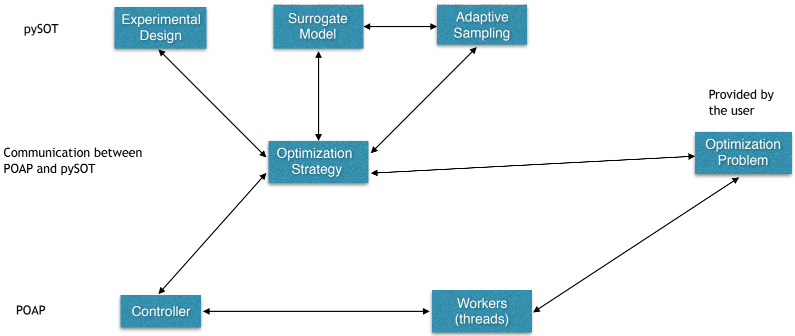

Figure: Interactions between the objects in POAP and pySOT

POAP

- POAP (Plumbing for Optimization with Asynchronous Parallelism)
- Hosted on GitHub: `https://github.com/dbindel/POAP`
- A framework for building and combining asynchronous optimization strategies
- The user can provide his own strategies
- Handles the communication with the objective function
- Supports combined strategies
- Capable of handling crashed function evaluations and workers crashing
- It is possible to retrieve partial information from the objective function evaluation

POAP

- Three main components
  1. A **strategy** for proposing new evaluations
  2. A set of **workers** that carry out function evaluations
  3. A **controller** asking workers to run function evaluations
- The controller is also responsible for
  1. Accepting or rejecting proposals by the strategy
  2. Controlling and monitoring the workers
  3. Informing the strategy object of relevant events.
- Different strategies can be composed by combining their control actions
- The workers and the strategy communicate via the controller

POAP

- The multi-threaded controller employs a set of workers
- Each worker is allowed to exploit parallelism
- There is support for communicating with an objective function that is not necessarily in Python (C, C++, Fortran, MATLAB, etc.)
- The user is responsible for implementing the optimization problem
- Workers can connect to a specified TCP/IP port to communicate with the controller

Back to the overview

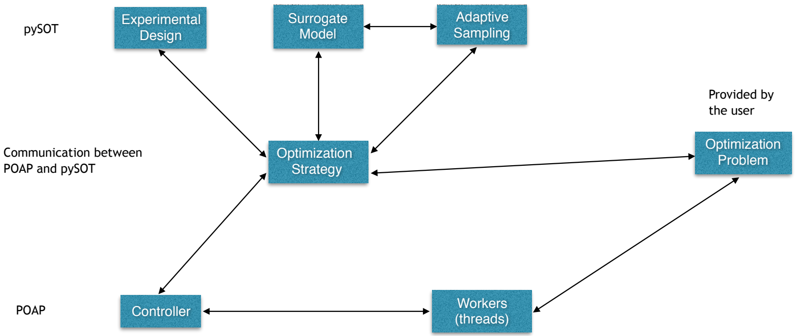

Figure: Interactions between the objects in POAP and pySOT

pySOT

- pySOT (Python Surrogate Optimization Toolbox)
- Hosted on GitHub: `https://github.com/dme65/pySOT`
- A collection of surrogate optimization strategies that can be used with POAP
- A great test-suite for doing head-to-head comparisons with different experimental designs, surrogate models, sampling techniques, strategies
- Comes with a large set of optimization test problems
- Object-orientation makes it easy for users to implement their own modules

pySOT

Main components:

(1) Optimization problem

- Number of dimensions
- Bound constraints
- Variable types
- Addition inequality constraints
- How to evaluate the objective function

(2) Experimental design generates the initial points

- Latin hypercubes (LHD)
- Symmetric Latin hypercubes (SLHD)
- Full-factorial (FF)
- Box-Behnken (BB)

pySOT

(3) Surrogate model approximates the objective function

- Radial basis functions (RBFs)
- Multivariate adaptive regression splines (MARS)
- Kriging
- Polynomial regression
- Linear combinations of the above models where the weights are determined using Dempster-Shafer theory

pySOT

(4) Adaptive sampling method decides where to evaluate next based

- Several candidate point based methods (DYCORS, SRBF, DDS, etc.)
- Minimizing the surrogate model using either a GA or a multi-start gradient method
- Minimizing the bumpiness if RBFs are used [Gutmann]
- Possible to perturb the integer or continuous variables separately
- Any cycle of the above methods, e.g., (DYCORS, DYCORS, GA, DYCORS, DYCORS, GA, ...)

pySOT

(5) Optimization strategy

- A synchronous strategy for problems with only bound constraints
- A synchronous penalty method for problems with inequality constraints
- A synchronous projection based strategy when it's easy to project an infeasible point onto the feasible region

- These three strategies have asynchronous versions, but they are yet to be added to pySOT.

The pySOT GUI

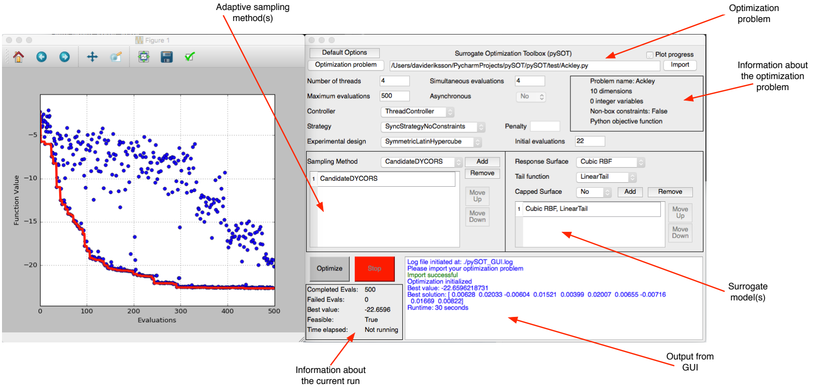

Figure: The pySOT GUI and its different components

Installing pySOT

- Install Anaconda for Python 2.7 (`https://www.continuum.io/downloads`)
- Install pySOT using the terminal command:
  pip install pySOT
- In order to use the GUI you need to install PySide:
  pip install PySide
- In order to use MARS you need to install py-earth:
  pip install https://github.com/jcrudy/py-earth/archive/master.zip
- pySOT + documentation + example code is on GitHub (`https://github.com/dme65/pySOT`)

Notebooks

- We will now go through the Python notebooks
- You can download the 7 notebooks + help files from `https://people.cam.cornell.edu/~dme65/talks.html` or `https://wakari.io/dme65`
- The notebooks make it easy to split the code into pieces
- It's also possible to save figures and outputs
- I will walk through the notebooks in detail
- You can choose to experiment with the notebooks on your own if you prefer!

Notebooks

Instructions for participants using their own laptops:

1. Download the zipped folder from `https://people.cam.cornell.edu/~dme65/talks.html`
2. Unzip the pySOT_notebooks and cd your way into the folder
3. Open the first notebook using: **jupyter notebook Example1.ipynb**

Notebook content

- Notebook 1: Introductory example, serial + threaded
- Notebook 2: 1D sampling pattern
- Notebook 3: How to make an optimization problem
- Notebook 4: MATLAB objective function
- Notebook 5: Non-bound constraints
- Notebook 6: Equality constraints
- Notebook 7: C++ objective function

POAP

- A framework for building and combining asynchronous optimization strategies
- Hosted on GitHub: `https://github.com/dbindel/POAP`
- The three main components are a controller, a strategy, and a set of workers
- Support for objective functions written in many programming languages
- Handles worker and objective function crashes
- The user supplies the optimization problem

pySOT

- A collection surrogate optimization strategies that can be used with POAP
- Hosted on GitHub: `https://github.com/dme65/pySOT`
- Comes with many different experimental designs, surrogate models, adaptive sampling techniques, strategies, test problems
- Easy to add your own components

Thank you for your attention!