ISBN: 978-3-95450-227-1 ISSN: 2673-5490 doi:10.18429/JACoW-IPAC2022-TUPOST058

BADGER: THE MISSING OPTIMIZER IN ACR*

Z. Zhang†, A. Edelen, C. Mayes, J. Garrahan, J. Shtalenkova, R. Roussel, S. Miskovich, D. Ratner
SLAC National Accelerator Laboratory, 94025 Menlo Park, USA
M. Boese, S. Tomin
Deutsches Elektronen-Synchrotron, 22607 Hamburg, Germany
G. Wang, Y. Hidaka
Brookhaven National Laboratory, 11973 Upton, USA

Abstract

Online optimization is crucial during accelerator operations to achieve satisfying machine performance. Optimization algorithms such as Nelder-Mead simplex, Gaussian process (GP), and robust conjugate direction search (RCDS) have been widely used in accelerator online optimization scenarios. The usual way of doing online optimization in accelerator control rooms (ACR) is to write a script that connects the algorithm to the problem. This approach would accrue code fragments that are difficult to maintain and reuse, plus the optimization progress can not be easily monitored and controlled. In this study, we propose an optimization platform named Badger to tackle the obstacles in ACR online optimizations. The design philosophies and features of Badger would be introduced and discussed.

INTRODUCTION

Modern large-scale accelerator facilities become more and more complicated, in consequence, the performance of the accelerators in operation usually differs from the design. Online optimization during operation is the key to bridging the gap between the designed properties and the ones in action, to achieve satisfying machine performance [1–7]. One obstacle to applying various algorithms in machine tuning tasks is that different algorithms usually work in different ways, the users have to write wrapper code to adapt the algorithms to their cases. This approach is not ideal since the number of the wrapper scripts would increase with time, and consequently makes it hard to manage and reuse these machine tuning-related scripts. Another drawback of creating a wrapper script for every new machine tuning task is code redundancy. For example, a large chunk of similar optimization progress visualization code needs to be written again and again – it is essential to see the optimization progress on the fly, although the visualization code itself is trivial and distracting.

The issues described above can be solved by employing an optimization framework that has a wide range of built-in algorithms, provides a straightforward way to create a custom optimization problem, and can monitor the optimization progress in some way. Xopt [8], Ocelot optimizer [9], and Teeport [10] are good candidates of optimizers that can be applied in online optimization scenario. However, more considerations must be taken in the accelerator control rooms (ACR), to boost the efficiency of the machine tuning tasks in daily operations: 1) The tasks are usually repeated hundreds of thousands of times, so rerunning a task should be as easy as possible, 2) optimization data for all history runs should be archived and logged properly, for future references, 3) optimizations are required to be strictly safe – no violations on variable hard boundaries would be tolerated, and 4) the optimization progress must be monitorable and controllable – the operators should be able to pause/resume/terminate a task according to the situation. We developed Badger – an online optimization framework that was designed with ACR usage in mind – that accounts for all the requirements above. Badger has been tested and verified to work in ACR of SLAC, DESY, and BNL.

CONCEPTS

Badger abstracts an optimization run as an optimization algorithm interacts with an environment, by following some pre-defined rules. As visualized in Figure 1, within an optimization routine, the environment is controlled by the algorithm and tunes/observes the control system/machine through an interface, while the users control/monitor the optimization flow through a graphical user interface (GUI) or a command-line interface (CLI).

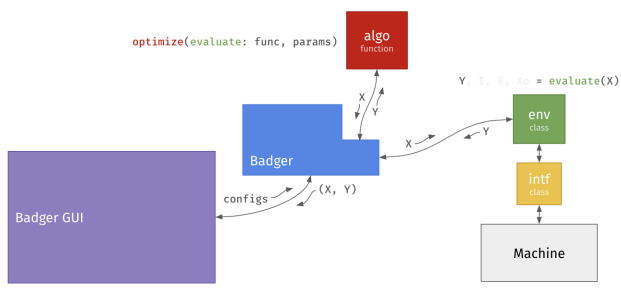

Figure 1: The architecture of Badger. The algorithm accepts an `evaluate` function provided by Badger as an argument, while the `evaluate` function sends trail solutions to be evaluated to the environment. Environment talks to the machine through an optional interface, to get or set the process variable (PV) values. The data flow during the optimization goes through the Badger core (shape in blue) and the optimization progress can be monitored/controlled by the Badger GUI or CLI (not shown in the plot).

* This work was supported by the U.S. Department of Energy, under DOE Contract No. DE-AC02-76SF00515 and the Office of Science, Office of Basic Energy Sciences.
† zhezhang@slac.stanford.edu

ISBN: 978-3-95450-227-1 ISSN: 2673-5490 doi:10.18429/JACoW-IPAC2022-TUPOST058

Algorithms, environments, and interfaces in Badger are all managed through a plugin system and could be developed and maintained separately. The application interfaces (API) for creating the plugins are very straightforward and simple, yet abstractive enough to handle various situations.

As mentioned above, there are several terms/concepts in Badger, and their meanings are a little different from their general definitions. We will briefly go through the terms/concepts in Badger in the following subsections.

Routine

An optimization setup in Badger is called a routine. A routine contains all the information needed to perform the optimization: 1) The optimization algorithm and its hyperparameters, 2) the environment in which the optimization would be performed, and 3) the configuration of the optimization, such as variables, objectives, and constraints.

To run an optimization in Badger, the users need to define the routine. Badger provides several ways to easily compose the routine so that the users are not required to write it by hand.

Interface

An interface in Badger is a piece of code that talks to the underlying control system/machine. It communicates to the control system to 1) set a process variable (PV) to some specific value, and 2) get the value of a PV. An interface is also responsible to perform the configuration needed for communicating with the control system, and the configuration can be customized by passing a params dictionary to the interface.

The concept of the interface was introduced to Badger for better code reuse. The users don't have to copy-n-paste the same fundamental code again and again when coding the optimization problems for the same underlying control system. With this concept, users could simply ask Badger to use the same interface, and focus more on the higher-level logic of the optimization problem. One thing worth mentioning is that interfaces are optional in Badger – an interface is not needed if the optimization problem is simple enough (such as an analytical function) that can be directly shaped into a Badger environment.

Environment

An environment is Badger's way to (partially) abstract an optimization problem. A typical optimization problem usually consists of the variables to tune, and the objectives to optimize. A Badger environment defines all the interesting variables and observations of a control system/machine. An optimization problem can be specified by stating which variables in the environment are the variables to tune, and which observations are the objectives to optimize. Furthermore, one can define the constraints for the optimization by picking up some observations from the environment and giving it a threshold.

Note that in Badger, one environment could support multiple relevant optimization problems – just put all the variables and observations to the environment, and use routine config to select which variables/observations to use for the optimization.

Routine config

A routine config is the counterpart of optimization problem abstraction regarding the environment. An optimization problem can be fully defined by an environment with a routine config. On top of the variables and observations provided by the environment, routine config tells Badger which and how variables/observations are used as the tuning variables/objectives/constraints. Combining the environment and routine config, Badger would compose an `evaluate` function with the following signature:

$$\mathbf{Y}, \mathbf{I}, \mathbf{E}, \mathbf{X_o} = \text{evaluate}(\mathbf{X}),$$

where $\mathbf{X}$ denotes the variables, $\mathbf{Y}$ the evaluated objectives, $\mathbf{I}$ the inequality constraints, $\mathbf{E}$ the equality constraints, and $\mathbf{X_o}$ the readout of the variables. This `evaluate` function would be sent to the algorithm once the optimization starts.

The reasons to divide the optimization problem definition into two parts (environment and routine config) are: 1) Better code reuse, and 2) operations in ACR usually require slightly changing a routine frequently, so it's good to have an abstraction for the frequently changed configurations (routine config), to avoid messing with the optimization source code (the environment class).

FEATURES

Once the users create their custom environments for Badger, Badger provides 3 modes to investigate, inspect, manage, compose, and run the optimization routines. Various features are available for each mode, most of the core functionalities are shared across the 3 modes, while a few features are exclusive in the particular mode. The following subsections would introduce the key features of the 3 modes, respectively.

CLI mode

In the command line mode, users could view the Badger meta information and settings by running:

```
badger
```

To list all the available plugins, or inspect a specific plugin, this command is provided:

```
badger algo/env/intf [PLUGIN NAME]
```

To view all the predefined routines, or examine/run one particular routine:

```
badger routine [ROUTINE NAME] [-r]
```

And finally, to create and run a routine:

```
badger run -a ALGO_NAME [-ap ALGO_PARAMS] \\
    -e ENV_NAME [-ep ENV_PARAMS] \\
    -c CONFIG_FNAME [-s ROUTINE_NAME]
```

When the routine is running, the evaluated trail solutions would be printed as a table in the terminal. The current optimal ones would be highlighted.

ISBN: 978-3-95450-227-1 ISSN: 2673-5490 doi:10.18429/JACoW-IPAC2022-TUPOST058

GUI mode

In a terminal where Badger is available, execute the following command:

```
badger -ga
```

would launch the Badger GUI.

Badger GUI provides a more comprehensive feature set compared to the other two. As shown in Figure 2, users could navigate through the history runs, filter out the interesting predefined routines, inspect the details regarding one routine, run and monitor a new optimization, and control the optimization progress (pause/resume/terminate/etc).

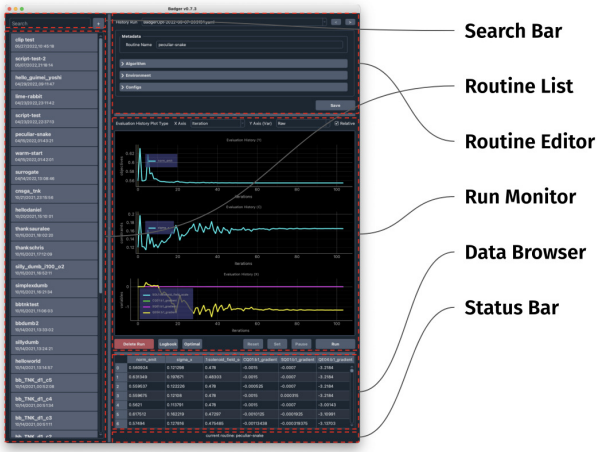

Figure 2: Badger main GUI. Users could browse/search through predefined routines, view the routine details, navigate through the history optimization runs, copy the interested optimization data, etc.

Users can also create a new routine based on a predefined one or from the ground up, in the Badger routine editor as shown in Figure 3.

With the Badger GUI, users could easily exploit the benefits that Badger core provides. It's the recommended mode to use in general.

API mode

Since Badger is a python package, it can be used the usual way as any other python package – being imported and used as a library. Badger provides `get_algo()`, `get_env()`, and `get_intf()` interfaces to grant the users access to the available plugins in Badger, so that the users could integrate Badger into their own workflows.

EXTENSIBILITY

Plugin system

Algorithms, interfaces, and environments are all plugins in Badger. A plugin in Badger is a set of python scripts, a YAML config file, and an optional README.md. Plugins can be developed and maintained separately and once

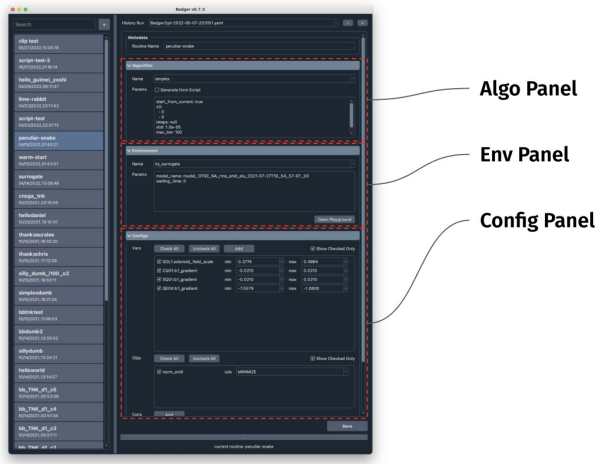

Figure 3: Badger routine editor. With the routine editor, users could select the optimization algorithm and environment to use, change the hyper-parameters for the algorithm and the environment, and configure the variables, objectives, and constraints (VOCs) of the routine.

released, they could be integrated into Badger by simply putting the whole plugin folder under a specific directory managed by Badger.

One interesting property of Badger plugins is that plugins can be nested – the plugin developer can use any available plugins inside the newly created ones. Say, one could combine two environments and create a new one effortlessly, thanks to this nestable nature of Badger plugins. The users could explore the infinity of possibilities by nesting plugins together with their imagination.

Extension system

The extension system is another way to extend Badger's capabilities, and in a sense, it is more powerful than the plugin system, since it could make a batch of existing algorithms available in Badger in a few lines of code.

With the extension system, Badger could use any existing algorithms from another optimization package. Currently, Badger has an extension for Xopt [8]. More extensions are planned to be implemented soon (for example, Teeport [10] extension for remote optimization).

CONCLUSION

We developed a plugin-based optimization platform Badger, which is designed specifically for online optimization scenarios in the accelerator control rooms. Badger provides CLI, GUI, and API modes to satisfy the needs under various circumstances. New optimization problems can be incorporated into Badger by creating custom environment plugins. Badger's capability could be easily extended through its plugin and extension system. More information can be found on the Badger homepage https://slac-ml.github.io/Badger.

ISBN: 978-3-95450-227-1 ISSN: 2673-5490 doi:10.18429/JACoW-IPAC2022-TUPOST058

REFERENCES

[1] X. Huang, J. Corbett, J. Safranek, and J. Wu, "An algorithm for online optimization of accelerators," *Nucl. Instrum. Methods Phys. Res., Sect. A*, vol. 726, pp. 77–83, 2013.

[2] X. Pang and L. Rybarcyk, "Multi-objective particle swarm and genetic algorithm for the optimization of the lansce linac operation," *Nucl. Instrum. Methods Phys. Res., Sect. A*, vol. 741, pp. 124–129, 2014.

[3] X. Huang *et al.*, "Development and application of online optimization algorithms," in *Proc. North Amer. Part. Accel. Conf.(NAPAC)*, 2016, pp. 1–5.

[4] W. F. Bergan, I. V. Bazarov, C. J. Duncan, D. B. Liarte, D. L. Rubin, and J. P. Sethna, "Online storage ring optimization using dimension-reduction and genetic algorithms," *Phys. Rev. Accel. Beams*, vol. 22, no. 5, p. 054 601, 2019.

[5] K. Tian, J. Safranek, and Y. Yan, "Machine based optimization using genetic algorithms in a storage ring," *Phys. Rev. Accel. Beams*, vol. 17, no. 2, p. 020 703, 2014.

[6] J. Duris *et al.*, "Bayesian optimization of a free-electron laser," *Phys. Rev. Lett.*, vol. 124, no. 12, p. 124 801, 2020.

[7] D. K. Olsson *et al.*, "Online optimisation of the MAX-IV 3 GeV ring dynamic aperture," *Proc. IPAC'18*, vol. 2281, 2018.

[8] C. Mayes, *Christophermayes/xopt: Flexible high-level optimization in python*, 2022. `https://github.com/ChristopherMayes/Xopt`

[9] S. Tomin, "Automated optimization of the european xfel performance with ocelot," in *ICFA Beam Dynamics Mini-Workshop: Machine Learning*, 2018.

[10] Z. Zhang, X. Huang, and M. Song, "Teeport: Break the wall between the optimization algorithms and problems," *Frontiers in big Data*, vol. 4, 2021.