ISBN: 978-3-95450-200-4 doi:10.18429/JACoW-ICAP2018-SAPAF03

COMPARISON OF MODEL-BASED AND HEURISTIC OPTIMIZATION ALGORITHMS APPLIED TO PHOTOINJECTORS USING LIBENSEMBLE

N. Neveu[*], L. Spentzouris, Illinois Institute of Technology, Chicago, USA
J. Larson, S. Hudson, Argonne National Laboratory, Lemont, USA

Abstract

Genetic algorithms are commonly used in the accelerator community and often require significant computational resources and manual adjustment of hyperparameters. Model-based methods can be significantly more efficient in their use of computational resources, but are often labeled as unreliable for the nonlinear or nonsmooth problems that can be found in accelerator physics. We investigate the behavior of both approaches using a photoinjector operated in the space-charge-dominated regime. All model-based optimization runs were coordinated and managed by libEnsemble, a Python library at Argonne National Laboratory.

ARGONNE WAKEFIELD ACCELERATOR FACILITY INTRODUCTION

The Argonne Wakefield Accelerator (AWA) facility houses two photoinjector beamlines. Ongoing research efforts at the AWA includes emittance exchange photocathode studies and two-beam acceleration experiments [1], the latter of which motivates this work. Figure 1 shows the layout of the AWA bunker during two-beam accelerator experiments. The high-charge beamline, often referred to as the drive beam, is being modeled in this work.

CODE AND RESOURCES

The particle-in-cell code OPAL [2] is used to simulate the high charge beam line at the AWA. OPAL is an open-source parallel code with two version, OPAL-t and OPAL-cycl. The former was used for this work, the latter version is used for modeling cyclotrons. OPAL can also simulate 3D-space charge, 1D coherent synchrotron radiation, and wakefield effects. Note that the optimization methods being compared in this study are applicable to any beam-dynamics code—Parmela [3], ASTRA [4], GPT [5], all of which have been used by AWA group members—not just OPAL-t.

All simulations were run on the Bebop cluster maintained by the Laboratory Computing Resource Center [6] at Argonne National Laboratory. Bebop machine consist of 664 Broadwell nodes, and 352 Knights Landing (KNL) nodes. The simulations presented here were performed on KNL nodes, due to short queue times and readily available resources.

[*] nneveu@hawk.iit.edu

OPTIMIZATION METHODS

Heuristic Method: Genetic Algorithm

Genetic Algorithms (GA) are a popular choice for optimizing simulations in the accelerator physics community. They have been used with success on several types of accelerator physics problems that address challenges facing both linear and circular machines [7]. It is not disputed that in these solutions were found with benefit to many facilities. However, it is also well-known that GAs are computationally intensive. They can require hundreds of thousands of core hours depending on the problem being solved. Performing optimization with fewer calls to the simulation would not only save time, but would also enable facilities to accomplish more design work without access to large core hour allocations.

Model Based Methods: APOSMM + BOBYQA

Model-based derivative free methods are increasingly popular in mathematics and other scientific domains, but have not been widely used in the accelerator physics community. This may be due to the assumption that these methods may become within local minima within a bounded search space. In a sense, this is true if the algorithm is always started with the same initial conditions and weights. However, if the algorithm is started multiple times with various initial conditions, this behavior may be mitigated.

In order to achieve a multistart approach, the asynchronously parallel optimization solver for finding multiple minima (APOSMM) [8] was used. This algorithm maintains a history of all previously evaluated points, and uses this information when deciding starting points for local optimization runs. APOSMM also allows concurrent local optimization runs while honoring the amount of resources available. For details on how local optimization points are determined, see [8, Section 3].

In this paper, we use the bounded optimization by quadratic approximation (BOBYQA) [9] local optimization method. After a set of simulation evaluations are finish, the beam parameters (i.e. objectives) at the desired location are fed to BOBYQA. The algorithm then builds and minimizes a quadratic model of the objectives in order to pick the next point to evaluate.

Both APOSMM and BOBYQA implementations used are open source, freely available, and written in Python.

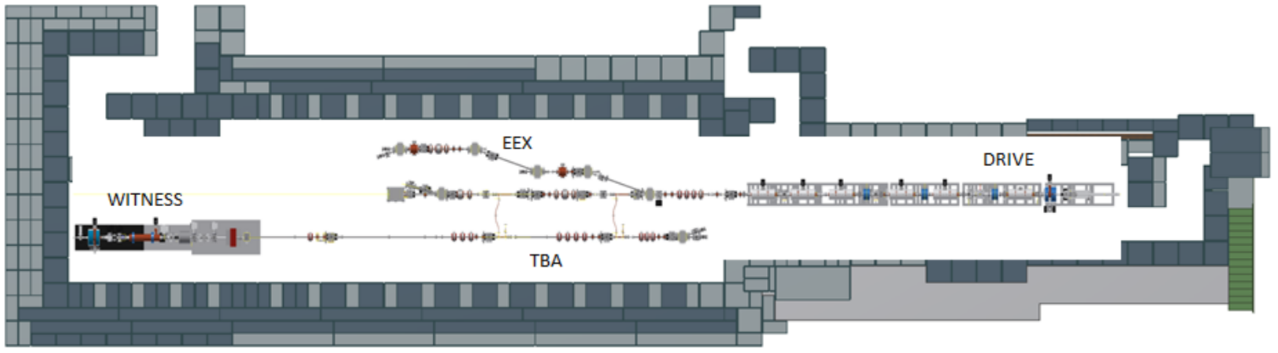

Figure 1: Layout of the AWA bunker. The drive beam with high charge is located on the right side of the image and the witness beam with lower charge and energy is shown on the left side of the image.

LIBENSEMBLE

`libEnsemble` is a library developed at Argonne for managing ensemble-like collections of computations. This library is capable of coordinating concurrent simulation evaluations and optimization algorithms. In the past, using a local optimization method in a multistart fashion required many serial runs, or much time spent on the users end to decide initial starting points and manage resources. `libEnsemble` in combination with APOSMM manages allocations and resources, so that the user can focus on optimization work and results rather than parallel programming and resource management. Key features of `libEnsemble` include:

- Automatically manages the asynchronous evaluation of calculations and, if desired, the optimization of outputs
- Helpful developers
- Open source Python code, on GitHub.
- Can run on laptops, clusters, and HPC systems

As an example of large resource management, `libEnsemble` has been used to coordinate the evaluation of 1,600 concurrent `OPAL` evaluations on 200 KNL nodes, each with 64 cores. `libEnsemble` provides many useful features for common accelerator physics use cases:

- Can gracefully kill simulation runs that loose particles *before* the end of simulation (thereby saving significant computational resources in studies with many failures)
- Gracefully kills runs that become unresponsive.
- Saves specified data into a NumPy array for easy access and storage
- Evaluates objectives based on specific beam criteria and z location.
- Allows for `OPAL-t` instances using parallel resources

This library has the potential to significantly simplify the use of model-based and multistart algorithms for optimization problems. In addition, this can save time spent on programming, since parallelization and allocation of resources for concurrent runs is managed by `libEnsemble` instead of user scripts.

Table 1: Parameter Bounds for Linac Optimization

Variable	Range	Unit
Solenoid Strength	$300 \leq S_1 \leq 550$	amps
Solenoid Strength	$180 \leq S_2 \leq 280$	amps
Phase of Gun	$-20 \leq \phi_g \leq 0$	degrees
Cavity Phase	$-20 \leq \phi_L \leq 20$	degrees

PHOTOINJECTOR OPTIMIZATION

The beamline simulated in `OPAL` consisted of the gun, two solenoids, and six linac cavities, as shown in Fig. 2. The charge of interest was 40 nC, therefore 3D space charge forces were calculated at all times, while the rf-field maps were 2D. The laser radius was set at 9 mm. Cavity gradients were set to achieve 65 MeV, with a small spread depending on the phase in each cavity. Nine design variables (shown in Table 1) were adjusted during the local optimization runs. Note ϕ_L is a vector containing the six cavity phases. Three objectives were chosen: σ_x, σ_y, σ_z. In this case, due to the 2D field maps, $\sigma_x = \sigma_y$. The three objectives reduce to two objectives, one representing the transverse beamsize, and one representing the longitudinal beamsize. Therefore the beamsize is optimized at the end of the linac.

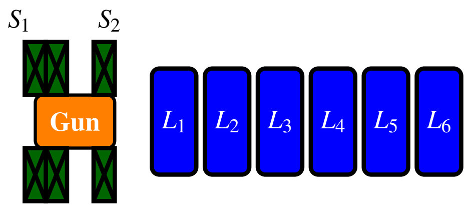

Figure 2: Simulation model used in OPAL and for optimization runs. The charge of interest was 40 nC.

First, `libEnsemble` was used to generate a 1,000 point random sample. These runs were then given by `libEnsemble` to APOSMM. BOBYQA was used as the local optimization method with a multistart approach. The objective was the sum of σ_x, σ_y, and σ_z. All starting points and subsequent points were chosen by APOSMM. Runs were initiated by `libEnsemble` and each `OPAL-t` simulation was run on four cores. `libEnsemble` maintained objective values and summary information needed by APOSMM after each simulation completed. A limit of 600 evaluations was set for the entire `libEnsemble` runs.

For comparison, we used the NSGA-II [10] implementation within `OPAL`. Examples of how to use the built-in optimizer can be found on the OPAL wiki[1]. The optimization problem was defined within an `OPAL-t` input file. The results of all simulations are shown in Fig. 3.

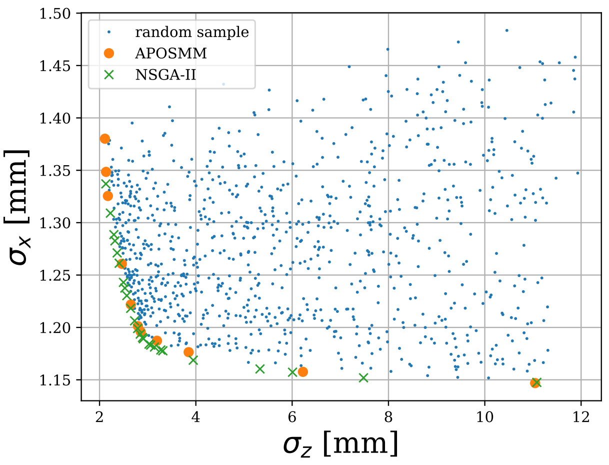

Figure 3: Comparison of GA and APOSMM optimization results. Note only σ_x is plotted, because σ_x and σ_y are equal. This is a result of using 2D symmetric field maps.

CONCLUSION

There is no significant difference in the Pareto fronts generated by NSGA-II, APOSMM, and the random sample. The GA performed 32,967 simulation evaluations to complete 200 generations. APOSMM completed 600 evaluations with a seed of 1,000 random points. Based on this case alone, it is not clear yet what the time to solution difference is possible for more complicated problems. This was a test case, where the design variables are bounded to regions of acceptable solutions. (The bounds were chosen based experienced gain in prior work.) This point is illustrated by the Pareto front of the 1,000 point random sample. It closely resembles the two optimization methods, which indicates the problem is well bounded. For this optimization problem, running either NSGA-II or APOSMM provided only marginal improvement of the solutions. This suggest a random sample may be sufficient in estimating Pareto fronts for some problems. A significant amount of computational resources could be saved in such cases; if a heuristic method is not used.

[1] `https://gitlab.psi.ch/OPAL/Manual-2.0/wikis/optimiser`

FUTURE WORK

Extension of this work is ongoing and will be published. That work includes optimization problems that are not well bounded or defined by a random sample, like the case shown here.

ACKNOWLEDGMENTS

We gratefully acknowledge the computing resources provided on Bebop, a high-performance computing cluster operated by the Laboratory Computing Resource Center at Argonne National Laboratory. This material is based upon work supported by the U.S. Department of Energy, Office of Science, under contract number DE-AC02-06CH11357 and grant number DE-SC0015479.

REFERENCES

[1] J. Shao *et al.*, “Recent Progress towards Dielectric Short Pulse Two-Beam Acceleration,” in *Proc. IPAC’18*, Vancouver, Canada, Apr-May 2018, `doi:10.18429/JACoW-IPAC2018-TUYGBE3`

[2] A. Adelmann *et al.*, “The OPAL (Object Oriented Parallel Accelerator Library) framework,” PSI, Zurich, Switzerland, Rep. PSI-PR-08-02, 2008-2017.

[3] L. M. Young, “PARMELA,” Los Alamos National Laboratory, Los Alamos, NM, USA, Rep. LA-UR-96-1835 (Revised April 22, 2003).

[4] ASTRA Manual, April 2014, `http://www.desy.de/~mpyflo/Astra_manual/Astra-Manual_V3.2.pdf`

[5] General Particle Tracker, `http://www.pulsar.nl/gpt`

[6] Laboratory Computing Resource Center, `https://www.lcrc.anl.gov`

[7] A. S. Hofler *et al.*, “Innovative applications of genetic algorithms to problems in accelerator physics,” *Phys. Rev. ST Accel. Beams*, vol. 16, p. 010101, 2013.

[8] J. Larson and S.M. Wild, “Asynchronously parallel optimization solver for finding multiple minima,” *Math. Prog. Comp.*, vol. 10, p. 1-30, Feb. 2018.

[9] M. Powell, “The BOBYQA algorithm for bound constrained optimization without derivatives,” University of Cambridge, U.K., Rep. NA2009/06, Oct. 2009.

[10] K. Deb, A. Pratap, S. Agarwal and T. Meyarivan, “A Fast and Elitist Multiobjective Genetic Algorithm: NSGA-II,” *IEEE Trans. on Evolutionary Comp.* , vol. 6, no. 2, pp. 182-197, Apr 2002.