Multi-Objective Optimization With an Integrated Electromagnetics and Beam Dynamics Workflow

David A. Bizzozero[a], Ji Qiang[a], Lixin Ge[b], Zenghai Li[b], Cho-Kuen Ng[b], Liling Xiao[b]

[a]*Lawrence Berkeley National Laboratory, 1 Cyclotron Rd., Berkeley, CA 94720, USA*
[b]*SLAC National Accelerator Laboratory, 2575 Sand Hill Rd., Menlo Park, CA 94025, USA*

Abstract

In particle accelerators, RF cavities are used to accelerate charged particle beams to designed high energy for physical applications. In a typical accelerator design, the optimization of RF cavities and the optimization of beam dynamics are carried out in separate studies. For a more general and unrestricted accelerator design, a coupled optimization of the RF cavities and the beam parameters is required. For this coupled optimization problem, we have developed an integrated electromagnetics and beam dynamics workflow management system. Within this system, the geometries for a set of cavity components are first adjusted; the field modes are then computed with an electromagnetics program, and imported into a beam dynamics program for beam dynamics simulation. This workflow is encapsulated into a parallel multi-objective optimizer to achieve the integrated accelerator design optimization. A multi fidelity strategy is developed to improve the speed of the optimizer. This integrated global optimization capability is illustrated using a photoinjector design example and yields an improved design.

1. Introduction

In previous studies, optimization of accelerator components and lattices have been done separately in a set of isolated optimizations. For example, in current injector optimization, accelerator components (e.g. RF cavities) are individually shape-optimized for performance subject to general requirements such as peak surface field, shunt impedance, and resonant frequency. Once these components' shapes are determined, beam dynamics simulations optimize around other output parameters such as bunch length and transverse emittance by adjusting the injector lattice parameters such as the amplitude and phase of the driving fields. However, this form of beam dynamics optimization is restricted by the fixed geometrical shape and field profile of the components.

While this form of sequential optimization may yield reasonable results, it will optimize locally only on a subset of the input parameter space at a time. By contrast, in end-to-end global optimization, all input parameters ranging from the photocathode geometric shape parameters (e.g. cathode tip radius of curvature), to the lattice phases and amplitudes can be adjusted simultaneously. The global optimization of input parameters will always yield a solution set of inputs at least as good as those obtained by sequential optimization provided enough iterations.

In this paper, we have developed a systematic integrated workflow management system, titled A3PI (ACE3P with IMPACT), written in Python which interfaces various programs with genetic algorithms and built-in parallelism to perform integrated global multi-objective optimization on high performance computing (HPC) systems.

In the following sections, after the Introduction, we present the integrated electromagnetics and beam dynamics workflow in Section II; we discuss the multi-objective global optimization with an integrated workflow in Section III;

we illustrate the multi-objective global optimization in a photoinjector application example in Section IV; and lastly we draw conclusions in Section V.

2. Integrated Electromagnetics and Beam Dynamics Workflow

2.1. ACE3P Overview

The parallel finite element electromagnetics modeling suite: ACE3P, was developed for accelerator cavity and structure design including integrated multi-physics effects in electromagnetic, thermal, and mechanical characteristics. The electromagnetic modules are discretized in the frequency domain and time domain for the computational volume inside an accelerator cavity. The thermal and mechanical solvers are formulated in the frequency domain for the computational volume of the cavity walls and their surroundings. Six simulation modules have been developed in ACE3P to address different physics aspects of accelerator applications [1, 2, 3]. The modeling capabilities of each ACE3P module [4] are summarized as follows:

1. Omega3P is an electromagnetic eigensolver in the frequency domain for calculating the resonant modes and their damping in accelerator cavities.
2. S3P is an electromagnetic solver in the frequency domain for determining the transmission of electromagnetic fields in open accelerator structures.
3. Track3P is a particle tracking code in the time domain for tracing electrons in accelerator structures under the influence of external static or dynamic electromagnetic fields for studying multipacting and dark current.
4. T3P is a time domain solver for the computation of wakefield excited by a charged particle beam and for studying transient effects from external electromagnetic excitations.
5. Pic3P is a full-wave particle-in-cell solver in the time domain for simulations of space-charge dominated devices.
6. TEM3P is a multi-physics module consisting of thermal and mechanical solvers for the analysis of integrated electromagnetic, thermal, and mechanical effects in accelerator cavities and structures.

In addition to these application modules, preprocessing tools for handling mesh formats and evaluating mesh entity statistics, postprocessing tools for visualization and analysis of simulation results, as well as a cavity shape optimization tool [5] have also been implemented.

2.2. IMPACT Overview

The IMPACT code is a parallel particle-in-cell code suite for modeling high intensity, high brightness beams in RF proton linacs, electron linacs, and photoinjectors [6, 7, 8, 9, 10, 11, 12]. It consists of two parallel particle-in-cell tracking codes: IMPACT-Z and IMPACT-T. The former uses longitudinal position as the independent variable and allows for efficient particle advance over large distances as in an RF linac, and the latter uses time as the independent variable and is needed to accurately model systems with strong space-charge as in photoinjectors. Additionally, the IMPACT suite contains an RF linac lattice design code, an envelope matching and analysis code, and a number of pre- and post-processing codes.

Both parallel particle tracking codes assume a quasi-electrostatic model of the beam (i.e. electrostatic self-fields in the beam frame, possibly with energy binning for a beam with large energy spread) and compute space-charge effects self-consistently at each time step together with the external acceleration and focusing fields. The 3D Poisson equation is solved in the beam frame at each step of the calculation. The resulting electrostatic fields are Lorentz-transformed back to the laboratory frame to obtain the electric and magnetic self-forces acting on the beam. There are six Poisson solvers in the IMPACT suite, corresponding to transverse open or closed boundary conditions with a round or rectangular pipe shape, and longitudinal open or periodic boundary conditions. These solvers use either a spectral method for closed transverse boundary conditions, or a convolution-based Green's function method for open transverse boundary conditions. The convolution for the most widely used open boundary condition Poisson solver is calculated using an FFT with a doubled computational domain. The computing time of this solver scales as $N \log(N)$, where N is the number of grid points. The parallel implementation includes both a 2D domain decomposition approach for the 3D computational domain and a particle-field decomposition approach to provide the optimal parallel performance for different applications on modern supercomputers such as the Knights Landing (KNL) compute nodes on Cori@NERSC [13]. In addition to the fully 3D space-charge capability, the IMPACT suite also includes detailed modeling of beam dynamics in RF cavities (via field maps or z-dependent transfer maps including RF focusing/defocusing), various magnetic focusing elements (solenoid, dipole, quadrupole, etc), allowance of arbitrary overlap of external fields (3D and 2D), structure and coherent synchrotron radiation (CSR) wakefields, tracking of multiple charge states, tracking multiple bin/bunches, an analytical model for laser-electron interactions inside an undulator, and capabilities for machine error studies and correction.

2.3. *A3PI Workflow Overview*

The integrated electromagnetics and beam dynamics workflow, A3PI, management system manages a workflow for optimization and interfaces several component codes including: Cubit [14], ACE3P [4, 15], and IMPACT [16, 17, 18].

A schematic of the integrated workflow is shown in Figure 1. Here, the A3PI is used to automate a chain of tasks such as: (1) run Cubit to generate the geometry of an accelerator cavity, (2) run Acdtool (a subprogram of ACE3P) to convert the output mesh from Cubit to a format suitable for ACE3P, (3) run Omega3P to compute eigenmodes of the meshed geometry, (4) run Acdtool again to extract the modal fields on a Cartesian grid for use in IMPACT, and (5) run IMPACT with the external fields provided.

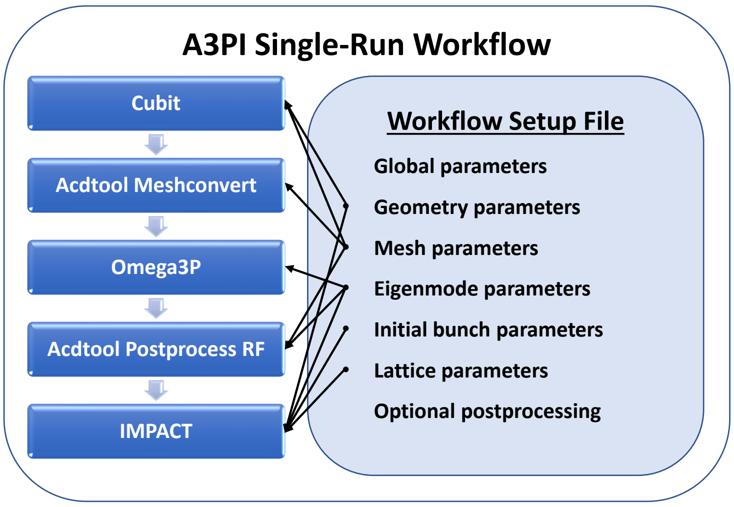


Figure 1. Example layout of a setup of A3PI to run a chain of tasks given a set of run parameters. The A3PI code will automatically generate input files for the component codes and run them sequentially.

Thus, A3PI can encapsulate the workflow as a single function evaluation where various parameters are the inputs and the IMPACT particle data are the outputs. Depending on the complexity of the desired workflow, the setup file of A3PI can be quite long as it contains all necessary information to run each code individually. However, A3PI is modular in that if a given component code (e.g. Cubit) isn't necessary, its section can be omitted in the setup file. The advantage to this approach is that A3PI can use the ConfigParser utility in Python to replace necessary values for each input file automatically. For example, if the length of a cavity is set up as a variable, then A3PI can automatically replace the appropriate values across various codes, such in the Cubit input file as well as in the lattice section for IMPACT.

2.4. *Cubit and Mesh Conversion in A3PI*

To run Cubit, a journal file must be provided which contains the instructions on how to generate a specific geometry (e.g. photocathode cavity).This journal file can be edited automatically by A3PI to update specific parameters before creating a mesh. For example, a user can create a generic geometry in Cubit and save it to a journal file. Then a keyword-value pair can be set up in the A3PI configuration file to replace a variable definition in the Cubit journal file so that updated values can be seamlessly inserted into the journal file prior to running Cubit.

After Cubit finishes meshing the geometry with the desired parameters, the ACE3P helper code: Acdtool, can be used to convert this mesh format into a NetCDF file for use in Omega3P. This NetCDF file can also be read-into MATLAB [23] for postprocessing visualization with some additional A3PI tools.

2.5. Omega3P and RF Postprocessing in A3PI

Next, Omega3P can be used to find eigenmodes with various settings from the formatted tetrahedral mesh from Acdtool. Typically, only the fundamental mode is used in accelerator design but other high-order-modes (HOMs) can be computed as well to study other effects. The A3PI workflow interfaces with Omega3P by creating an input file based on the geometry provided and other user specifications such as the finite element method (FEM) order and boundary conditions for particular surfaces.

Once a given set of eigenmodes are computed for the input geometry, the modal fields are interpolated along a 3D Cartesian grid centered along the beam axis. This is done since the beam dynamics simulation only requires fields in the neighborhood of the particle bunches' trajectory and greatly reduces the input field file size required for IMPACT. This Cartesian field interpolation procedure is done with Acdtool and uses the openPMD [19] standard with the HDF5 file format. The result is a set of 4 field files per eigenmode containing the real and imaginary parts of the electric and magnetic fields.

2.6. IMPACT in A3PI

For the final section of the workflow, IMPACT is used for a beam dynamics simulation using the fields computed from the earlier steps. The input settings for Impact are divided into two parts: the initial beam parameters, and the accelerator lattice. The A3PI workflow can interface with these sections and automatically generate Impact-formatted input files. Also, A3PI can perform phase scanning for a given IMPACT simulation by optimizing phases for each lattice element; this is based on existing IMPACT code but has been fully integrated into A3PI.

Once a given IMPACT simulation is complete, the resulting output files are then parsed by A3PI to determine final quantities of interest including: beam energy, bunch size, bunch emittance, and so forth. This parsed data is used as the output when a A3PI workflow is used as a function evaluation while the various input parameters used throughout the workflow are the function inputs.

3. Multi-Objective Optimization

3.1. Overview

In the previous section, we discussed how to set up A3PI to run a single chain of tasks for a given set of parameters. The next step is to use this workflow as a black-box function to optimize a set of input parameters with respect to desired output quantities of interest. The problem of optimization can be stated in the general mathematical form as:

$$\min \begin{cases} f_1(\vec{x}) \\ \cdots \\ f_k(\vec{x}) \end{cases} \text{subject to } g_i(\vec{x}) \leq 0, h_i(\vec{x}) = 0,$$

where $f_1, \cdots, f_k$ are k objective functions to be optimized, $\vec{x} \in X \subseteq \mathbb{R}^n$ is a vector of n control parameters, and g_i and h_i are constraints to the optimization. The goal of multi-objective optimization is to find the Pareto-optimal front in the objective function solution space. The Pareto-optimal front is a collection of non-dominated solutions in the whole feasible solution space. Any other solution in the feasible solution space will be dominated by those solutions on the Pareto-optimal front. In the multi-objective optimization, a solution A is said to dominate a solution B if all components of A are at least as good as those of B (with at least one component strictly better). Here, a component of A corresponds to one objective function value, i.e. $A_i = f_i(\vec{x})$. The solution A is non-dominated if it is not dominated by any solution within the group.

For the multi-objective optimization in A3PI, we use the DEAP [20] Python library to set up a genetic algorithm; we opt to use NSGA-II but there are other algorithms to choose from for various types of problems. Our aim was to set up A3PI as an integrated workflow manager and additional algorithms can be added with minor efforts.

While we could simply wrap a multi-objective optimizer around the A3PI workflow, if the individual workflow evaluations are costly, convergence to a Pareto front may take a very long time. To further parallelize A3PI, we use the libEnsemble [21] Python library to assist with the task distribution to multiple nodes in an HPC environment. One particular challenge is that the tasks within the A3PI workflow may use varying amounts of resources: Omega3P in particular has a large memory footprint to accurately solve for eigenmode fields. Thus, by using libEnsemble, A3PI can efficiently distribute tasks to computing resources for the optimization cycle.

To set up A3PI for use with DEAP and libEnsemble, an additional section is added to the A3PI setup file which contains optimization parameters such as population size, mutation parameters, etc. in addition to the libEnsemble-specific options desired. When A3PI is called with these additional settings, a manager process distributes a single-run workflow, as in Figure 1, to different "worker" folders for parallel evaluation. The choice of input parameters from the manager process is passed to each worker and the workflow is evaluated; the output quantities of interest from the workers are then returned to the manager process for mutation in choosing the next population generation. When the desired convergence is achieved or a maximum number of evaluations is performed, the manager process returns the results of the optimization. An overview diagram of the optimization hierarchy is shown in Figure 2.

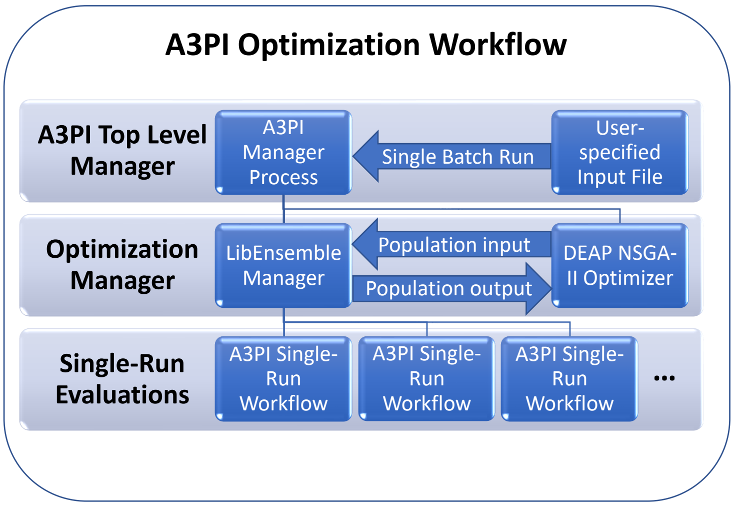


Figure 2. Example layout of a setup of A3PI to run a chain of tasks given a set of run parameters. A user supplies an input file (top right), then A3PI manager (top left) will automatically allocate resources for the component codes. The optimization cycle (middle row) uses the DEAP optimizer to select input parameters and the libEnsemble manager to create and execute single-run workflows. Each single-run evaluation (bottom row) is performed in parallel while the tasks within each run are sequential.

3.2. Optional Field Interpolation

Since Omega3P typically uses substantially more resources, roughly 10 times as many compute nodes, as the other components in the A3PI workflow task chain, we designed an interpolation method to approximate the Omega3P output. If the number of input parameters for the tasks involving Omega3P or its preceding steps is small. While this introduces an additional approximation error, our tests have found that if the parameter space is sufficiently, the field error is small. The drawback is that the electromagnetic fields need to be computed for a set of various parameters in advance but can be quickly interpolated once saved to external files.

For example, if only a few geometrical shape parameters are to be optimized over, then the *Cubit* → *Acdtool mesh convert* → *Omega3P* → *Acdtool postprocess RF* task chain can be run a few times with varying combinations of those parameters. Then, with the modal Cartesian fields for a small set of values, an interpolant can be constructed to approximate the fields at any intermediate value of that parameter.

The goal is to predict the field for a set of n control parameter inputs without running Omega3P more than m times, with m taken to be small but must satisfy $m > n$. Specifically, to interpolate the modal fields using m pre-computed fields for n parameters, we can represent the parameter combinations by the vector $\vec{x}_i \in \mathbb{R}^n$ with $i \in \{1, 2, ..., m\}$. Then to find the fields at a new value $\vec{y} \in \mathbb{R}^n$ (which must lie in the convex hull of $\vec{x}_i$) we can use the n-dimensional Delaunay algorithm from SciPy [22] to locate a simplex consisting of $n + 1$ values of $\vec{x}_i$ containing $\vec{y}$. Then we interpolate the value of the fields at the value of $\vec{y}$ using the barycentric coordinates of $\vec{y}$ inside that simplex as weights.

For a quick example, if we have two control parameters that we want to interpolate in the unit square: $(\alpha,\beta) \in [0, 1] \times [0, 1]$, we can compute the fields using a full workflow, which includes Omega3P, on the corners: $\vec{x}_1 = (0, 0)$, $\vec{x}_2 = (1, 0)$, $\vec{x}_3 = (0, 1)$, and $\vec{x}_4 = (1, 1)$. Then if we need the value of the fields for $\vec{y} = (0.2, 0.5)$, we note that this point lies in the simplex (i.e. triangle) formed by $\vec{x}_1$, $\vec{x}_2$, and $\vec{x}_3$. The barycentric coordinates of $\vec{y}$ in that simplex is $(0.3, 0.2, 0.5)$, and thus we interpolate the fields with the control parameters $\vec{y}$ by combining the field solutions for $\vec{x}_1$, $\vec{x}_2$, and $\vec{x}_3$ with the weights 0.3, 0.2, and 0.5 respectively.

One difficulty arises if the field data is inhomogeneous for different sets of control parameters. For example, if the length of a cavity is chosen as a control parameter, then by changing its value, the physical domain may change. Since the field data is always on a 3D Cartesian grid after the Acdtool postprocess step, we can map this data conformally to a unit cube. Then the field interpolation can be done with the barycentric coordinates as before and the 3D Cartesian domain can be resized afterwards to the appropriate dimensions. This procedure is not robust, but will work reasonably well if the domain's size does not vary significantly with respect to the corners of the simplex containing the query point $\vec{y}$.

For a final validation of any optimization, we can revert back to the full workflow for the field calculation. The advantage of the interpolation routine is to speed up the optimization cycle, by a factor of ∼100 in our tests (after the pre-computed fields are given), until the population is closer to the Pareto-optimal solutions.

4. Injector Optimization Example

To showcase the capability of A3PI, we use a test model of an injector lattice consisting of 200 MHz SRF photocathode gun with a movable cathode stalk (Figure 4 (top)), a focusing solenoid, and four 1.3 GHz TESLA 9-cell SRF accelerating cavities [24] (Figure 4 (bottom)). The input parameter space we optimize over is given in Table 1. Other necessary simulation parameters such as bunch charge or cavity RF power are either held fixed or determined from the input parameters; some of these additional fixed parameters are given in Table 2. The ranges for phase parameters were chosen after using A3PI in single-run mode to determine a suitable range.

For the multi-objective optimization test, we set A3PI to minimize the final transverse RMS emittance and bunch length while maintaining the constraint that the beam energy must be greater than 60 MeV. Such constraints are enforced via a penalty scaling term to objectives. Specifically, for this test we define the penalty factor F as:

$$F = 1 + \max\{E_{\text{beam}}/(1\,\text{MeV}) - 60, 0\},$$

and the penalty is applied to the final transverse emittance and bunch length by multiplying those quantities by F. A coefficient to the max function can be used to adjust the strictness of the penalty factor as needed. However, due to the restricted phases in our test following the initial phase scan, the beam energy was naturally constrained to above 60 MeV without need of the penalty factor.

Next, since all parameters except for d_{cathode} are only used by IMPACT, we can simplify the A3PI workflow by using field interpolation as defined in the previous section. For this procedure, the *Cubit* $\rightarrow$ *Acdtool mesh convert* $\rightarrow$ *Omega3P* $\rightarrow$ *Acdtool postprocess RF* task chain is initially performed only 6 times to obtain field data using $d_{\text{cathode}} = \{0, 2, 4, 6, 8, 10\,\text{mm}\}$. These field maps are then used to construct approximate fields at other values of d_{cathode} within the defined range. Thus a shortened A3PI workflow can be set up for this model by replacing the aforementioned task chain with a very fast field lookup and interpolation routine.

While this interpolation shortcut introduces a small error in the optimization cycle, the Pareto-optimal parameters from the shortened workflow can be validated and re-optimized by using the complete workflow for a few additional population generations. Figure 3 (top) shows the on-axis longitudinal electric field profile for the photocathode for varying values of the cathode stalk position d_{cathode}. As a comparison, Figure 3 (middle) shows a zoomed-in view of E_z near the cathode for $d_{\text{cathode}} = 4\,\text{mm}$ along with interpolated approximations using the data from neighboring solutions $d_{\text{cathode}} = \{0, 10\,\text{mm}\}$ or $d_{\text{cathode}} = \{2, 6\,\text{mm}\}$. The relative errors for these interpolated solutions are shown in Figure 3 (bottom).

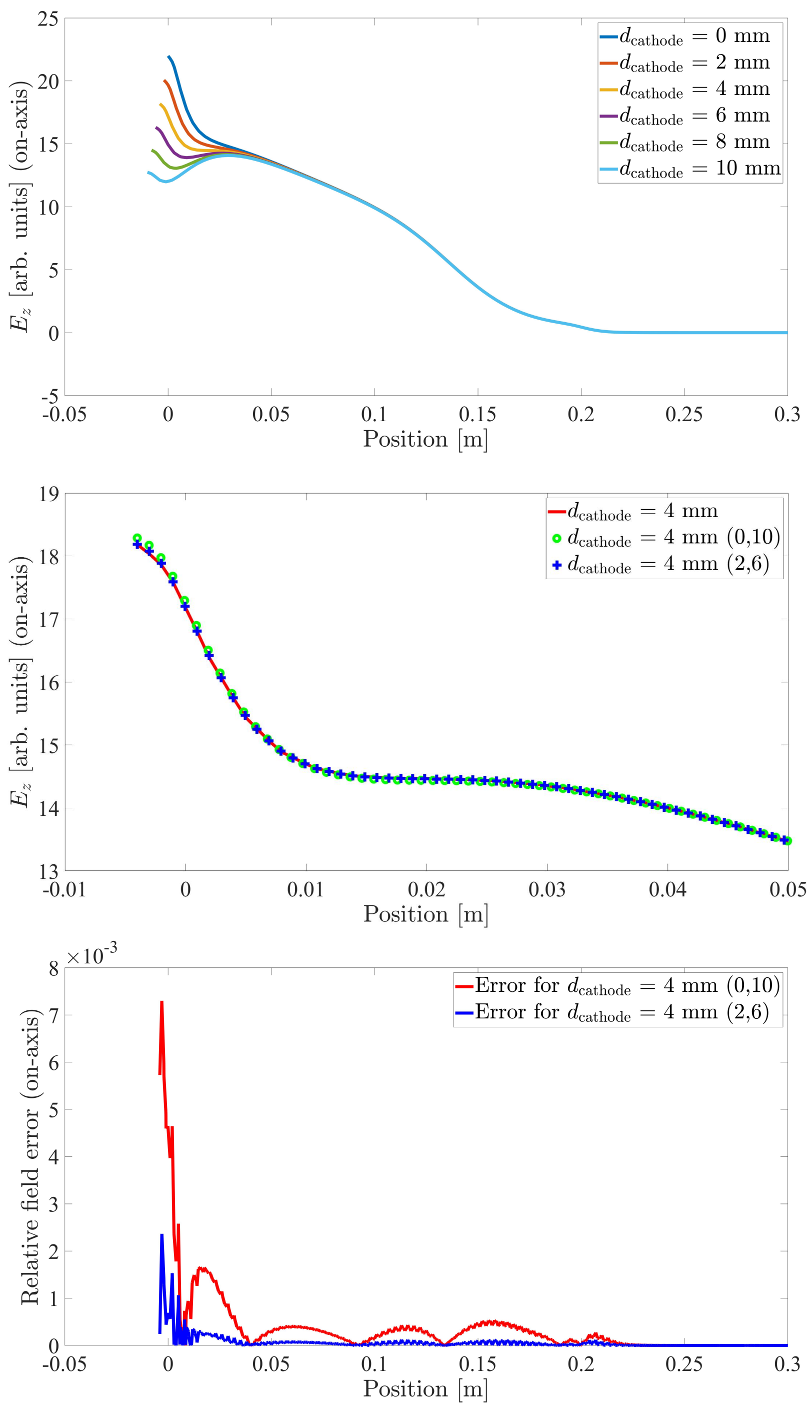


Figure 3. (Top) On-axis E_z field profile for varying values of d_{cathode}. (Middle) Zoomed-in comparison view of on-axis E_z field profile for $d_{\text{cathode}} = 4\,\text{mm}$ with the actual field data, compared to using the interpolation method with only the two points: $d_{\text{cathode}} = \{0, 10\,\text{mm}\}$, and the interpolation method with only the two points: $d_{\text{cathode}} = \{2, 6\,\text{mm}\}$. (Bottom) Relative error between the actual field data for $d_{\text{cathode}} = 4\,\text{mm}$ and the interpolated fields. This plot indicates that our field interpolation routine using the 6 pre-computed solutions of $d_{\text{cathode}} = \{0, 2, 4, 6, 8, 10\,\text{mm}\}$, to construct a field solution at an arbitrary value of $d_{\text{cathode}} \in [0, 10\,\text{mm}]$, results in less than $\approx 0.1\%$ field error.

To test the optimization model, we set the population size to 128 individuals, using 128 workers with 1 KNL node allocated to each, and ran an A3PI workflow for a total of 50 generations with the NSGA-II optimizer. Each individual evaluation using the shorter interpolation-based workflow takes approximately 1 minute on a KNL node with most of that time used for the IMPACT evaluation. Thus using 128 KNL nodes, this full optimization can be completed in less than an hour. However, when using the full task chain workflow, since Omega3P requires more CPU and memory resources such that a single-run evaluation uses approximately 10 minutes with 10 KNL nodes, each generation with the full workflow takes approximately 100 times more CPU-node-hours. Therefore, it is not recommended to use the full workflow until the population is close to the Pareto-front; in our example, after 40~50 generations.

A final validation was done using the full workflow for 2 generations using the same population but with 80 KNL nodes, using 8 workers with 10 nodes allocated to each, for approximately 5 hours but did not show significant improvement over using the interpolated workflow. We conjecture that since only a single geometry parameter was used, and that the field was sampled sufficiently with the 6 values of d_{cathode}, the interpolation error had a very small effect on the workflow optimization. However, if a large number of geometrical parameters are required, and therefore the number of input dimensions is higher, the use of an interpolation workflow may suffer from a greater pre-computing overhead since many more samples are required.

As shown in Figure 5, the individuals approach an approximate Pareto-optimal boundary after a few dozen generations. Each individual reflects a choice of the 9 parameters given in Table 1 which are then evaluated with the A3PI single-run workflow; the output final transverse emittance and bunch length are plotted.

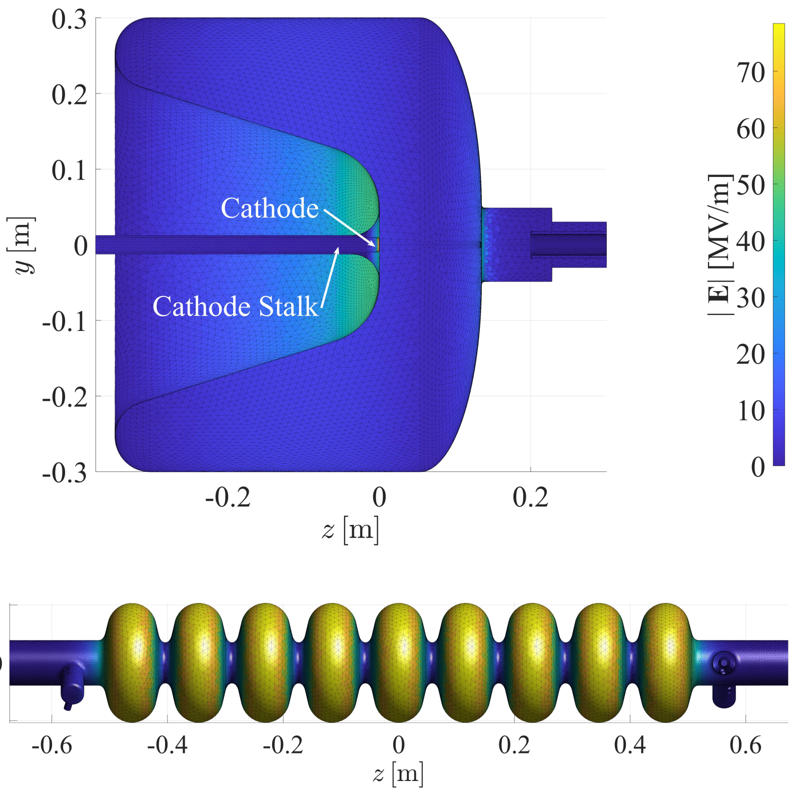


Figure 4. (Top) Slice-view of model RF photocathode with movable stalk. As the cathode depth d_{cathode} increases, the cathode can move inwards which changes the E_z on-axis field. Note: this geometry is for testing purposes and does not have an optimized design with regards to the peak surface field on the cathode plug outer diameter and the gun cavity nose. (Bottom) Mesh geometry of one of the 9-cell 1.3 GHz accelerating cavities.

Table 1. Model Injector Input Parameters

Parameter	**Range**	**Description**
d_{cathode}	$0-10$ mm	Cathode stalk position
θ_{cathode}	$150-170$ deg	Cathode driving phase
$\sigma_{x,y}$	$400-800$ µm	Laser transverse spot size
σ_z	$25-40$ µm	Laser pulse length
B_{solenoid}	$100-200$ mT	Focusing solenoid strength
θ_{accel1}	$280-310$ deg	Accelerating cavity 1 phase
θ_{accel2}	$280-310$ deg	Accelerating cavity 2 phase
θ_{accel3}	$280-310$ deg	Accelerating cavity 3 phase
θ_{accel4}	$280-310$ deg	Accelerating cavity 4 phase

Table 2. Model Injector Additional Parameters

Parameter	**Value**	**Description**
q_{bunch}	200 pC	Electron bunch charge
ω_{cathode}	200 MHz	SRF gun cavity frequency
W_{total}	8.85 µJ	Cathode cavity total energy*
ω_{accel}	1.3 GHz	Accelerating cavity frequency
$E_{z,\text{accel}}$	30 MV/m	Accelerating cavity gradient

*: Cathode E_z field varies from 25~45 MV/m with stalk position.

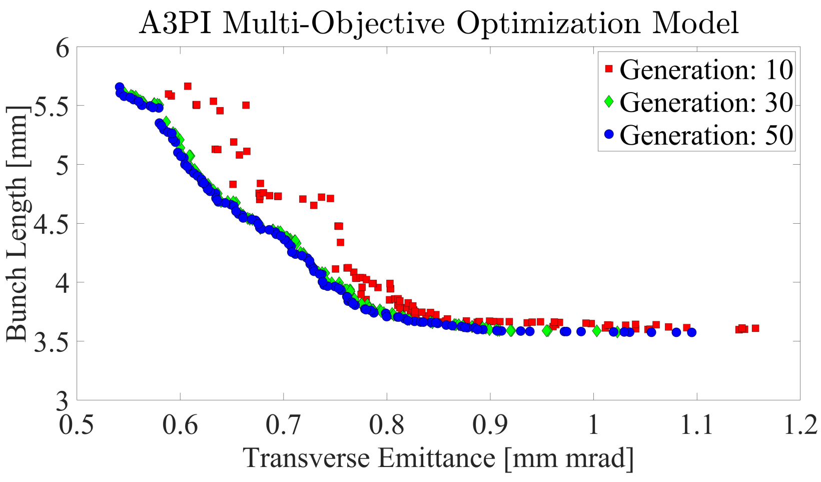


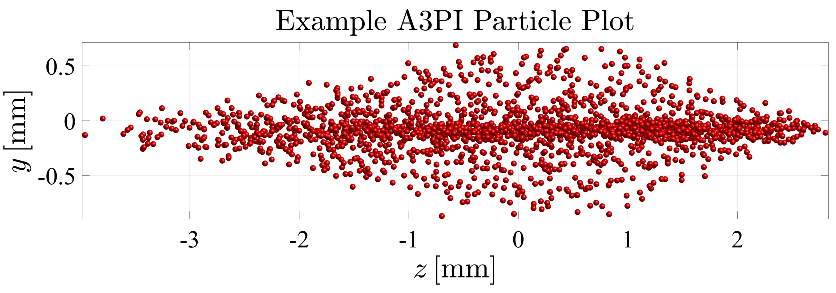
Figure 5. The best 128 individuals at varying generations from the NSGA-II multi-objective optimization of the A3PI injector model.

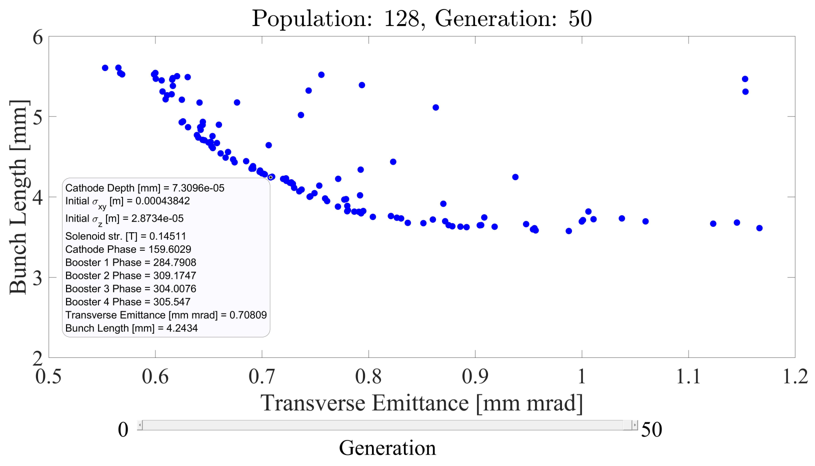
5. Additional A3PI Capabilities

We also would like to introduce a set of MATLAB tools we developed for the purposes of visualizing various outputs. We have included several visualization scripts with the A3PI workflow library: plotting routines for fields and meshes using the Acdtool and Omega3P NetCDF outputs, for particles using the IMPACT outputs, and for optimization populations using the libEnsemble history array outputs.

These visualization scripts have various options but in particular can be used to create exportable video animations for time-series data such as particle evolution, or generation-series data from the multi-objective optimization cycles. Additionally, the data can be viewed in an interactive mode to quickly view particle bunch shapes or a set of input parameters for a particular optimization population individual. A demo of particle and history plots are shown in

Figure 6. Additionally, more details on the A3PI workflow management system are given in Appendix A and the A3PI user guide [25].





Cathode Depth [mm] = 7.3096e-05
Initial σ_{xy} [m] = 0.00043842
Initial σ_z [m] = 2.8734e-05
Solenoid str. [T] = 0.14511
Cathode Phase = 159.6029
Booster 1 Phase = 284.7908
Booster 2 Phase = 309.1747
Booster 3 Phase = 304.0076
Booster 4 Phase = 305.547
Transverse Emittance [mm mrad] = 0.70809
Bunch Length [mm] = 4.2434

Figure 6. (Top) Example particle plot from IMPACT data extracting during the time-stepping routine. A subset of the total number of particles are shown and the plot can be adjusted to display momentum coordinates as well. (Bottom) Example A3PI history plotting routine in interactive mode. The code includes a generational slider (example shown) can be used to select a generation and the custom data tip displays the set of input parameters and output objectives of that particular simulation.

6. Conclusion

Multi-objective optimization of an end-to-end accelerator structure is an important aspect in the design of next-generation particle accelerators. For this task, we created A3PI, our novel integrated workflow manager written in Python and visualization tools written in MATLAB, to interface existing electromagnetics and beam dynamics codes. The visualization tools are built on existing internal MATLAB commands for simplicity, but similar tools could be developed in the future using only Python with third-party modules. We have demonstrated the integrated optimization capability with a test model injector but aim to use A3PI for a fully realistic photoinjector design in a future study.

Our future goals include adding compatibility of other codes to A3PI and further improvements to the A3PI utilities in more versatile computing environments. Since A3PI is modular, more codes can be added to a task chain by following a common procedure of writing a template to create an input file for a given code, and interfacing that template with A3PI by adding new sections to a configuration file.

7. Acknowledgments

This work is supported by the Director of the Office of Science of the US Department of Energy under contracts DE-AC02-05-CH11231 and DE-AC02-76-SF00515 and used computer resources at the National Energy Research Scientific Computing Center.

Appendix A. A3PI Technical Details

The A3PI Workflow is designed with the Python ConfigParser class and its extended interpolation option. The ConfigParser class lets Python manage changes to a given workflow input file before sending it to workers (e.g. CPU nodes) for evaluation. The extended interpolation option allows the ConfigParser to copy values which are changed (e.g. physical length of a component) to all appropriate places (e.g. cubit geometry, Acdtool postprocess routine, Impact-T input file, etc.).

The workflow configuration file is generally long, but it combines all necessary inputs for each code in a single place. In this appendix, we will overview an example workflow input file and explain each keyword section. Each keyword section must be surrounded by square brackets `[]`. Within a given section, keyword entry pairs must be separated by an equals = sign. If more than one value is to be assigned, then the arguments for the keywords should be separated by a comma `,` (spaces work for some sections too). Comments may be made using the # symbol.

Folder structure with A3PI is handled as follows: the workflow configuration file is located in the *base* folder. When this file is parsed by A3PI, it will accordingly generate workflow *worker* folders in which a copy of the workflow configuration file is placed, with corresponding adjustments to desired variables. To clarify, the workflow configuration file is *unique* for each use of A3PI, but when evaluating multiple workflows (e.g. for optimization), each workflow *worker* folder will have its own copy of the *worker* configuration file but with differences in the variable section `[VARS]` and a few other settings.

Appendix A.1. Common Workflow Sections

The `[VARS]` section is used to store all variables which are expected to change for multiple workflow runs. For the example below, the variable `d_cath` is set initially to `0.005`. However, during optimization, this value can be changed and the numerical value of `d_cath` will be automatically inserted elsewhere in the workflow accordingly. Note: variable names in this example are not built-in and can be added, changed, or removed as desired; the only requirement is that the variable names are consistent throughout the file.

```
[VARS]
#Define the various paramters that are updated each workflow

d_cath = 0.005
cath_phase = 160.0
sig_xy = 4.00e-4
sig_z = 2.50e-5
sol_str = 0.150
b1_phase = 300.0
b2_phase = 300.0
b3_phase = 300.0
b4_phase = 300.0
```

Next, the `[PATHS]` section defines all the code paths needed for the workflow to run. Each code can have a different path but should all be accessible from the workflow input file's directory (absolute path names are preferred). The `workflow_path` is special in that it defines the folder to create for a given workflow.

The idea is that A3PI is run from a base directory with this workflow configuration file. Next, this configuration file is interpolated with the appropriate numeric replacements, such as `d_cath` in the previous example section, and a new folder named `workflow_folder` is created. The interpolated workflow input file (which is now static) is placed into the new folder for evaluation by a worker (e.g. CPU node). When using LibEnsemble to run multiple concurrent workflow evaluations, each folder will have an automatically appended number unique for each worker (e.g. workflow_folder_0, workflow_folder_1, etc.).

```
[PATHS]
#Define paths to where the executables are found

cubit_path = mypath/cubit
acdtool_path = mypath/acdtool
omega3p_path = mypath/omega3p
impactt_path = mypath/ImpactTexe_knl_gnu
workflow_path = mypath/workflow_folder
```

The next section includes special platform-specific run options such as how many cores to use for Acdtool post-process, Impact-T, and various input data files. The static_files list includes all necessary data files needed to run the workflow. For example, this list can include the base geometry files for Cubit, static field maps for Impact-T, or any other files that need to be copied into the workflow_folder. These static files must be in the base directory with the workflow configuration file. Lastly, the A3PI_mode option can be set to single to run an A3PI workflow once, or optimize to use A3PI for multi-objective optimization.

```
[RUN_PARAMETERS]
#Define run command, HPC options, and list of static files needed for workflow

A3PI_mode = optimize
acdtool_cores = 32
omega3p_cores = 640
impactt_cores = 64
static_files = rfdata1, rfdata2, cubit_geometry.jou, cubit_acis_file.sat
```

Appendix A.2. LibEnsemble and DEAP Workflow Sections

This section defines all the necessary parameters and settings for multi-objective optimization using libEnsemble with DEAP. A typical usage would be to optimize some parameters (e.g. d_cath, sig_xy, etc.) in the [VARS] section with respect to multiple objectives such as xy_rms_emittance and z_rms_size while remaining within some constraints on other parameters such as beam_energy. To accomplish this, we can set up the [LIBENSEMBLE] section as follows.

We start by defining the num_workers value which should be at most 1 less than the number of nodes provided. This is because libEnsemble reserves one node for the managing thread. We can also set the number of nodes_per_worker here, but the default is 1. The current version of A3PI does not support dynamic node balancing yet which would allow workers to share nodes as needed if certain steps in a workflow require more resources than other steps.

For the genetic algorithm, the population size and maximum number of generations is specified. During the evolutionary process, the NSGA-II algorithm used by A3PI enforces a constant population to ensure nodes are used more efficiently.

Next, the input variables are listed as strings and must match the names of variables listed in the [VARS] section. This is to ensure the variables are properly linked and updated when the workflow is run. Each variable needs its own lower and upper bound in the parameter range; this is provided as a list which follows the order the variables were listed in the vars entry within the [LIBENSEMBLE] section. The number of input variables listed in the vars line must match the number of bounds provided. In this example, the number of input variables is 9.

The objectives, are listed next in a similar form but should also include signed weights for the relative behavior desired for each parameter. In the example, both objectives are to be minimized so the weights are set to -1. The number of output quantities listed in the objectives line must match the number of weights provided. In this example, the number of objectives is 2. The complete list of available objectives are given in the Table A.3. The descriptions include the IMPACT output filename and and column number where the parameter is located.

Table A.3. Supported Objective and Constraint List

Parameter	Description
`beam_energy`	Beam energy [MeV] (column 4 in fort.18)
`x_rms_size`	Transverse x-size [m] (column 4 in fort.24)
`x_rms_emittance`	RMS x-emittance [m] (column 8 in fort.24)
`y_rms_size`	Transverse y-size [m] (column 4 in fort.25)
`y_rms_emittance`	RMS y-emittance [m] (column 8 in fort.25)
`xy_rms_size`	Transverse xy-size* [m]
`xy_rms_emittance`	RMS xy-emittance* [m]
`z_rms_size`	Longitudinal z-size [m] (column 3 in fort.26)
`z_rms_emittance`	RMS z-emittance [m] (column 7 in fort.26)

*: Transverse xy options use the $\|\cdot\|_2$ norm to represent the average radial component.

Optionally, constraints can be provided and A3PI uses a simple linear penalty factor for objective functions which a constraint parameter does not satisfy the bounds provided. In this example, if the `beam_energy`, which is determined automatically by reading a corresponding Impact-T output file, does not lie in the prescribed range, a penalty factor is applied. This penalty factor F is defined for constraints C_i with corresponding bounds $[C_{\min}, C_{\min}]$ and constraint penalty scaling terms P_i as:

$$F = 1 + \sum_i [\max\{C_{\min} - C_i, 0\} + \max\{C_i - C_{\max}, 0\}] \cdot P_i$$

Thus, for the example, if the `beam_energy` value is 66 MeV (default units from Impact-T), then the penalty factor $F = 1$ (no penalty). However, if the beam energy drops to 55 MeV, then the penalty factor $F = 6$. This is then applied to the objective function evaluations y (e.g. y_1 = `xy_rms_emittance` and y_2 = `z_rms_size`) by the formula:

$$y_j^{\text{penalized}} = y_j^{\text{original}} \cdot F^{-\text{sign}(y_j) w_j}$$

For minimization objectives, the weight factor w_j is negative and thus can be penalized by increasing y_j (or decreasing y_j if it is negative). Likewise, for maximization objectives, with $w_j > 0$, y_j can be penalized by decreasing it (or increasing y_j if it is negative). The number of constraint penalty quantities listed in the `const_penalty` line must match the number of `constraints` provided. In this example, the number of objectives is 2.

Next, for the NSGA-II algorithm in DEAP, we provide the mutation crossover probability `cxpb`, mutation parameter `eta`, and independent mutation paramter `indpb`. These parameters are covered more in detail on the DEAP documentation [20]. Lastly, we can optionally define termination criteria for the optimization. Currently, only `sim_max` is supported by A3PI, but more options can be added in a future version.

```
[LIBENSEMBLE]
#Define the parameters needed for libEnsemble multi-worker evaluation (with MPI)
#This section also defines parameters for DEAP multi-objective optimization

#If workers set to auto, number of workers is MPI.COMM_WORLD size -1
num_workers = auto
nodes_per_worker = 1

#Define population size, max number of generations, input dimensions, objectives
pop_size = 128
num_gen = 50

#Define the inputs and their respective bounds (uniformly random initially)
#Variable names must be string names of variables defined in the [VARS] section
vars = d_cath, cath_phase, sig_xy, sig_z, sol_str, b1_phase, b2_phase, b3_phase, b4_phase
```

```
lower_bounds = 0.0, 150.0, 4.0e-4, 2.5e-5, 0.100, 280.0, 280.0, 280.0, 280.0
upper_bounds = 10.0, 170.0, 8.0e-4, 4.0e-5, 0.200, 310.0, 310.0, 310.0, 310.0

#Define outputs to optimize and their weights (-1 to minimize, +1 to maximize)
objectives = xy_rms_emittance, z_rms_size
weights = -1, -1

#Define constraints as penalty, if a constraint parameter is outside bounds, the
#objective functions are multiplied by a factor of dist(const,bound)*penalty
constraints = beam_energy
const_lower_bounds = 60
const_upper_bounds = 100
const_penalty = 1.0

#Define crossover probability, mutation parameter, and independent mutation prob.
cxpb = 0.8
eta = 20.0
indpb = 0.09

#Define optional exit criteria [default is sim_max = pop_size*(num_gen+1)]
sim_max = 6528
```

Appendix A.3. A3PI Code-Specific Sections

The remaining sections needed to run A3PI depend on which tasks are necessary in the task chain for a single-run workflow as in Figure 1. For example, if using an interpolated workflow without Cubit, Acdtool, or Omega3P, then the input sections pertaining to those codes are not required. The implementation of IMPACT-T is described in the following paragraphs.

The input parameters for IMPACT are based on the 'ImpactT.in' text file structure found in [26]. The parameters are separated into two sections: a beam parameters section, and a lattice section. In the beam parameters section `[IMPACTT_PARAMETERS]`, the 9 entries correspond to the first 9 rows in the IMPACT input file. The A3PI workflow will parse these entries and generate an automatic IMPACT input file to be run. The benefit of this automated file generation is to allow for the use of the variables stored in the `[VARS]` section to coordinate an entire workflow when changing various parameters with the $ symbol in the configuration file.

Familiarity with IMPACT, and its guide [26], is recommended when using `[VARS]` variables to be interpolated with the ConfigParser. Another restriction is that the size of the processor array `procs` (in this example 8×8) must be compatible with the `impactt_cores` value defined in the `[RUN_PARAMETERS]` section (in this example 64).

```
        [IMPACTT_PARAMETERS]
        #Define the ImpactT input parameters (line numbers 1-9 here)
        #See ImpactT user guide for more information

        #processors = col row
        procs = 8 8

        #steps = dt Nstep Nbunch
        steps = 4.0e-12 2000000 1

        #parts = PSdim Npart integF errF diagF imchgF imgCutOff
        parts = 6 5000 1 0 2 0 0.02

        #mesh = Nx Ny Nz bcF Rx Ry Lz
        mesh = 32 32 32 1 0.15 0.15 1.0e5

        #dist = distType restartF substepF Nemission Temission
        dist = 112 0 0 1300 6.5e-11

        #*dist = sig* sigp* mu*p* *scale p*scale xmu* xmu*
        xdist = ${VARS:sig_xy} 0.001 0.0 1.0 1.0 0.0 0.0
        ydist = ${VARS:sig_xy} 0.001 0.0 1.0 1.0 0.0 0.0
        zdist = ${VARS:sig_z} 0.0014 0.0 1.186189e-6 1.0 0.0 0.002

        #beam = I/A Ek/eV Mc2/eV Q/e freq/Hz phs/rad
        beam = 0.26 1.0 0.511005e6 -1.0 1.3e9 0.0
```

In the IMPACT lattice section `[IMPACTT_LATTICE]`, each line will correspond to a lattice line in the IMPACT input file. The keys with prefix `elem_` are not name-specific in the automated IMPACT input file generator, but each element must listed in the workflow configuration file in order. The `elem_` lines are helpful for the user when setting up the lattice, but a given element can be commented out with the # symbol without requiring a renumbering of the other elements. Lattice elements with a lot of text input, such as `elem_03` in the example, can be either a single long line, or separated into multiple indented lines.

```
        [IMPACTT_LATTICE]
        #Define the ImpactT lattice (line numbers 10+ here)
        #See ImpactT user guide for more information

        #Start 3D SC
        elem_00 = 0.0 1 20 -5 -2.0 -2.0 -2.0

        #SRF gun element
        elem_01 = 0.3 105 20 111 0.0 2.0e6 2.00e8 ${VARS:cath_phase} 300 0.1 0.0 0.0 0.0 0.0

        #Focusing solenoids
        elem_02 = 0.48 105 20 105 0.250e0 0.0 0.0 0.0 5 0.1 0.0 0.0 0.0 0.0 0.0 ${VARS:sol_str}

        #Boosting cavities
        elem_03 = 1.346 10 20 105 2.6702 3.00000e+07 1.3e9 ${VARS:b1_phase}
                400 0.1 0.0 0.0 0.0 0.0 0.0 0.0
        elem_04 = 1.346 10 20 105 4.0538 3.00000e+07 1.3e9 ${VARS:b2_phase}
                400 0.1 0.0 0.0 0.0 0.0 0.0 0.0
        elem_05 = 1.346 10 20 105 5.4374 3.00000e+07 1.3e9 ${VARS:b3_phase}
                400 0.1 0.0 0.0 0.0 0.0 0.0 0.0
        elem_06 = 1.346 10 20 105 6.8210 3.00000e+07 1.3e9 ${VARS:b4_phase}
                400 0.1 0.0 0.0 0.0 0.0 0.0 0.0
```

Other sections, such as for Cubit, Omega3P, or Acdtool, are organized in a similar manner and variables can be cross-linked using the same ConfigParser interpolation method using the $ symbol. Once every section that is required has been defined with the appropriate keyword value pairs, the A3PI workflow requires a short final section which contains the task chain information. See the complete A3PI guide [25] for more details.

References

[1] L. Ge, K. Ko, O. Kononenko, Z. Li, C.-K. Ng, L. Xiao, and J. Qiang, “Advances in Parallel Finite Element Code Suite ACE3P,” Proc. of 6th International Particle Accelerator Conference, Richmond, VA, USA May 3-8, 2015.

[2] K. Ko, A. Candel, L. Ge, A. Kabel, L-Q. Lee, Z. Li, C. Ng, V. Rawat, G. Schussman, and L. Xiao, “Advances in Parallel Electromagnetic Codes for Accelerator Science and Development,” LINAC2010, Tsukuba, Japan, 2010.

[3] O. Kononenko, C. Adolphsen, Z. Li, C.-K. Ng, and C. Rivetta, “3D Multiphysics Modeling of Superconducting Cavities with a Massively Parallel Simulation Suite,” Phys. Rev. Accel. Beams 20, 102001 (2017).

[4] https://portal.slac.stanford.edu/sites/conf_public/cw16/Pages/default.aspx, Code Workshop for ACE3P at SLAC: CW16 .

[5] V. Akcelik, L.-Q. Lee, Z. Li, C. Ng, L. Xiao, and K. Ko, “Large scale shape optimization for accelerator cavities,” J. of Phys.: Conference Series, 180, 012001 (2009).
[6] J. Qiang, C. Mitchell, M. Venturini, “Suppression of microbunching instability using bending magnets in FEL linacs,” Phys. Rev. Lett., vol. 111, 054801 (2013).
[7] J. Qiang, J. Corlett, C. E. Mitchell, C. F. Papadopulos, G. Penn, M. Placidi, M. Reinsch, R. D. Ryne, F. Sannibale, C. Sun, M. Venturini, P. Emma, S. Reiche, “Start-to-end simulation of x-ray radiation of a next generation light source using the real number of electrons,” Phys. Rev. ST Accel. Beams, vol. 17, 030701 (2014).
[8] J. Bahng, E. Kim, and J. Qiang, “Design study of low-energy beam transport for multi-charge beams at RAON,” Nuclear Instruments & Methods in Physics Research A 804, p. 99, 2015.
[9] J. Qiang, Y. Ding, P. Emma, Z. Huang, D. Ratner, T. O. Raubenheimer, F. Zhou, and M. Venturini, “Start-to-end simulation of the shot-noise driven microbunching instability experiment at the Linac Coherent Light Source,” Phys. Rev. Accel. Beams 20, 054402 (2017).
[10] J. Qiang, “Mitigation of envelope instability through fast acceleration in linear accelerators,” Phys. Rev. Accel. Beams 21, 114201 (2018).
[11] J. Qiang, “Long-term simulation of space-charge effects,” Nuclear Instruments & Methods in Physics Research A 918, p. 1 (2019).
[12] J. Qiang, L. Brouwer, and R. Teyber, “Fixed field phase shifters for a multi-pass recirculating superconducting proton linac,” Phys. Rev. Accel. Beams 24, 030101 (2021).
[13] National Energy Research Scientific Computing Center. Website: http://www.nersc.gov/
[14] Roshan W. Quadros et al, “The CUBIT Geometry & Meshing Toolkit”, Sandia National Laboratories, https://cubit.sandia.gov
[15] L.-Q. Lee “A Parallel Finite-Element Eigenmode Analysis Code for Accelerator Cavities”, SLAC Technical Report, SLAC-PUB-13529, (2009).
[16] J. Qiang, R. Ryne, S. Habib, and V. Decyk “An Object-Oriented Particle-In-Cell Code for Beam Dynamics Simulation in Linear Accelerators”, J. Comp. Phys. vol. 163, 434 (2000).
[17] J. Qiang, S. Lidia, R. Ryne, and C. Limborg-Deprey “Three-dimensional quasistatic model for high brightness bream dynamics simulation”, Phys. Rev. ST Accel. Beams **9**, 044204 (2006).
[18] J. Qiang et al, “High resolution simulation of beam dynamics in electron linacs for x-ray free electron lasers”, Phys. Rev. ST Accel. Beams **12**, 100702 (2009).
[19] A. Huebl et al. “openPMD: A meta data standard for particle and mesh based data, technical specification (CC-BY 4.0)”, November 2015, 10.5281/zenodo.591699
[20] Félix-Antoine Fortin et al, “DEAP: Evolutionary Algorithms Made Easy”, J Mach Learn Res, vol 13, pp 2171–2175, July 2012.
[21] Stephen Hudson et al, “libEnsemble User Manual”, Argonne National Laboratory, Rev 0.7.1, 2020.
[22] Pauli Virtanen et al. (2020) “SciPy 1.0: Fundamental Algorithms for Scientific Computing in Python” Nature Methods, 17(3), 261-272. Package scipy.spatial.Delaunay v0.9 retrieved May 2021 from https://docs.scipy.org/doc/scipy/reference/generated/scipy.spatial.Delaunay.html
[23] MATLAB version 8.5.0.197613 (R2018a). The MathWorks Inc., Natick, Massachusetts, United States.
[24] B. Aune et al, “Superconducting TESLA cavities”, Phys. Rev. ST Accel. Beams, vol. 3, 092001 (2010).
[25] D. Bizzozero, “A3PI User Guide v0.5”, May 2021. Retrieved on September 3, 2021, from: ?
[26] J. Qiang, “IMPACT-T User Document Beta Version 1.8”, LBNL-62326, 2007. Written April 2015. Retrieved on September 3, 2021, from: https://portal.nersc.gov/project/m669/IMPACT-T/documents/