ISBN: 978-3-95450-168-7 doi:10.18429/JACoW-IPAC2015-MOPJE076

MULTI-OBJECTIVE GENETIC OPTIMIZATION WITH THE GENERAL PARTICLE TRACER (GPT) CODE

S.B. van der Geer, M.J. de Loos, Pulsar Physics, Eindhoven, The Netherlands

Abstract

In a typical design process there are a large number of variables, external constraints, and multiple conflicting objectives. Examples of the latter are short pulse, high charge, low emittance and low price. The classical solution to handle such problems is to combine all objectives into one merit function. This however implicitly assumes that the tradeoffs between all objectives are a-priori known. Especially in the early design stages this is hardly ever the case. A popular solution to this problem is to switch to multi-objective genetic optimization algorithms. This class of algorithms solves the problem by genetically optimising an entire population of sample solutions based on selection and recombination operators. The output, the so-called Pareto front, only includes solutions that are fully optimized where no objective can be improved without degrading any other. Here we present numerical studies and practical test runs of the genetic optimizer built into the General Particle Tracer (GPT) code [1].

INTRODUCTION

GDFMGO is a new multi-objective global optimizer for the GPT code. Its internal algorithm evolves a population of candidate solutions towards the so-called Pareto-front. This front is defined as the set of points where it is impossible to improve one objective without making at least one other objective worse. The final result is a set of points equally spaced at or near the Pareto front, where each point is fully optimized in all its variables.

The Pareto front samples, for the given objectives, the optimal combination of variables. For two objectives and ten variables, the Pareto front is a line through two dimensional objective space and ten dimensional variable space. In other words, the original problem of having to choose the best combination of ten variables has been reduced to the one dimensional problem of selecting where on the line you want to be. This collapse of dimensionality makes multi-objective optimization such a powerful tool for decision making. Its power does not come from providing the 'the best' solution, but from reduction of the dimensionality of the problem and simultaneously providing insight in tradeoffs between the different objectives.

GDFMGO starts with a fixed size population of candidate solutions with variables either chosen randomly in a user specified interval or read from file. Typically this initial population is very poor in terms of the objectives, and probably in violation of a number of constraints if present. The idea is to create improved candidate solutions based on the solutions already present, while gradually removing the worst ones. This process is repeated over and over again, and slowly the population as a whole evolves in the desired direction. Global convergence properties come from the fact that the population is pushed gently towards the Pareto front instead of steepest descent into a potential local minimum. Although there are plenty of pathological cases where this does not work, in practice the algorithm finds the global optimum in a impressively large number of cases with the default parameters.

The key ingredients of the algorithm are a) creation of new candidate solutions based on the existing ones, b) ranking the solutions such that the worst can be removed and c) a stopping criterion. They are described in detail below.

New Species

New random samples are created from the existing population. The mechanism we use is known as Differential Evolution (DE) [2]. To create a new combination of variables **p**, first four random species from the population are selected. We denote them **u**, **v**, **w** and **x**. Subsequently, an intermediate point **t** is calculated from **u**, **v** and **w** using: $\mathbf{t} = \mathbf{w} + s\,(\mathbf{u} - \mathbf{v})$.

The idea is that the new point is close to the existing **w**, but offset with the direction from **v** to **u**. To enforce convergence, the direction is slightly reduced by a scale factor s that must be below 1. In practice using this intermediate point **t** as the new point already works very well. However, when the entire candidate solution lies on a hyperplane in all variables there is no escape from this plane thereby stalling the algorithm. This is where the last selected point **x** comes into play, where there is a mutation probability (1-ρ) that variables are selected from **x** instead of from **t**. To avoid duplicating **x** at least one of the variables must come from **t**. In equations:

$$cr = \text{random\_int}[1, n]$$
$$p_i = \begin{cases} t_i & \text{random}[0,1] < \rho \quad \vee \quad i = cr \\ x_i & \text{otherwise} \end{cases}$$

A lower ρ increases global convergence properties at the cost of a slower convergence rate. In practice neither the scale factor s nor the mutation rate ρ are critical parameters and the defaults of 0.6 and 0.9 respectively work fine in most cases.

New candidate solutions are added until the population size is doubled. Then the population is ranked and trimmed to the original size again. How this is done is the subject of the next section.

Ranking Procedure: NSGA

The initial population is based on a random selection in all variables. Obviously, such random start conditions give poor results in terms of the objectives. The idea is to slowly improve this population, but to do this a ranking procedure is needed that defines which points are 'better' and which ones are 'worse'. In case of a single objective this is trivial, since we can simply sort all candidate solutions: If the objective needs to be minimized, an ascending sorting order gives a natural rank and the 'worse' ones can simply be defined as an upper fraction.

In the case of multiple and conflicting objectives things become more complicated. When a candidate solution has, compared to another one, shorter pulse-duration (good) but larger emittance (bad) it is not immediately clear if this one is better or worse. The idea followed in GDFMGO is based on the NSGA algorithm [3], where the ranking is based on domination: A point dominates another if it is better in at least one objective, and not worse in all other objectives. All points that are not dominated by any other point are given rank 0. This is the set that lies closest to the Pareto front, and these are the 'best' points in our set. The ones with rank 1 are the ones that are not dominated by any other point, not taking into account points with rank 0. They are the 'next best'. And so forth and so on. The final result is a candidate set that is fully ranked based on all objectives. An example is shown in Fig 1 where stdx and stdy are conflicting objectives to be minimized. Points with small stdx but large stdy are correctly assigned the same rank as points with large stdx but small stdy.

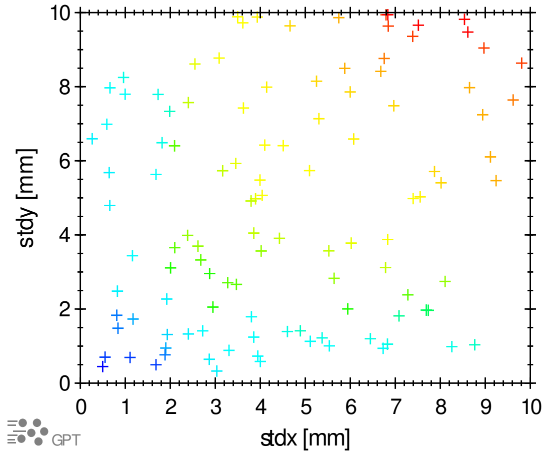

Figure 1: Multi-objective ranking shown in colours ranging from blue, via green and yellow, to red.

The ranking procedure is used to trim the candidate solution. Initially points with the highest rank are removed since they are furthest away from the Pareto front. When more than 50% of the points have rank 0, also particles with rank 0 are removed. This is done in such a way that the front is sampled as uniformly as possible by removing points with near neighbours while keeping the extremities.

Additional Constraints

An extra complication in the ranking procedure is that in many cases there are additional constraints. An example is a spot size as small as possible, under the condition that the total charge is larger than 1 pC. With just one extra constraint, an intuitive ranking procedure is that all candidates that violate the constraint are dominated by candidates that do not violate the constraint. This allows the rest of the procedure to remain identical, and indeed this works just fine. With more constraints however it is impossible to define something like the sum of all violations. If the spotsize must be smaller than 10 micron and we are 1 micron off, and if the charge must be larger than 1 pC and we are 0.1 pC off, then the total violation is 1 micron plus 0.1 pC. These can only be added with additional scaling factors, and these typically cannot easily be defined. The chosen solution is to only count the number of violations, and completely disregard information related to how far off a candidate solution lies.

The consequence of the above solution is that when most candidates violate several constraints there can be too much evolutionary pressure. The reason is that because of the elitist nature of the algorithm it will not temporarily give up on one constraint to fulfil another one. This fully undermines the global optimization properties of the algorithm. To solve this problem GDFMGO uses dynamic constraints: The actual constraints are relaxed until at most half the population violates the constraint. This causes a much more gentle evolutionary push towards zero constrained violations, restoring the global convergence properties and fully eliminating the need for scaling factors.

EXAMPLE

The example shown below describes the optimization of a small sub-relativistic UED beamline [4] consisting of a 100 keV electrostatic gun, two solenoids and an rf-compression cavity, as shown in Fig 2. The difficulty in the design of this beamline is that the dynamics of the first part is entirely space-charge driven, and this in turn couples the transverse and longitudinal beam dynamics in a non-linear way.

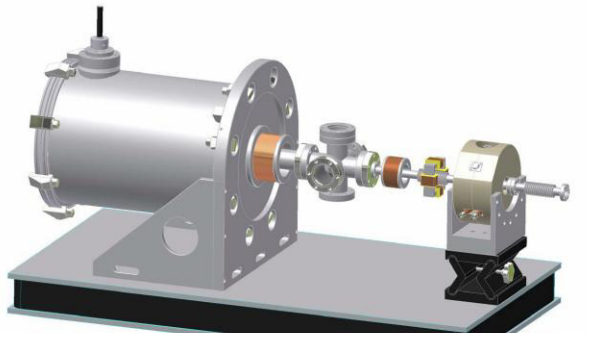

Figure 2: UED beamline to be optimized.

ISBN: 978-3-95450-168-7 doi:10.18429/JACoW-IPAC2015-MOPJE076

Objectives

In the beamline shown above we have the following objectives at the target: a) short pulse length, b) large amount of charge, c) large coherence length = low beam emittance, and d) small spot-size. The above situation with many conflicting objectives is typical, but from experience we learned that this is not the most useful way to start the design process. It is better to rewrite a few of these objectives into 'good enough' constraints. For example, for all applications where <100 fs pulse duration is 'short enough' it is possible to drop this objective and rewrite at as a constraint. Shown in Fig 3 is a colour coded representation of the ranking at an intermediate step during the optimization, where the constraint at 100 fs is clearly visible as an abrupt colour change.

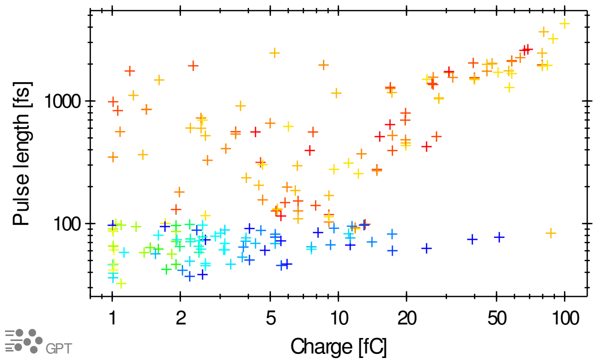

Figure 3: Ranking with the additional constraint that pulse duration must be below 100 fs. Red points are all worse compared to blue and green points.

In the results shown hereafter we analogously constrain the spot size to be below 250 micron. The remaining objectives are Charge and Coherence Length.

Variables

There are many variables such as locations and strengths of solenoids but in this paper only the rf-phase and rf-field are shown. An interesting intermediate step in the optimization process is shown in Fig 4, where the optimizer shows that there is a non-linear tradeoff possible between rf-phase and rf-field: The same amount of compression is attained by a slight misphasing and increasing the field. The final result, not shown, reveals that a slight deceleration is beneficial for the compression process by compensating higher order effects.

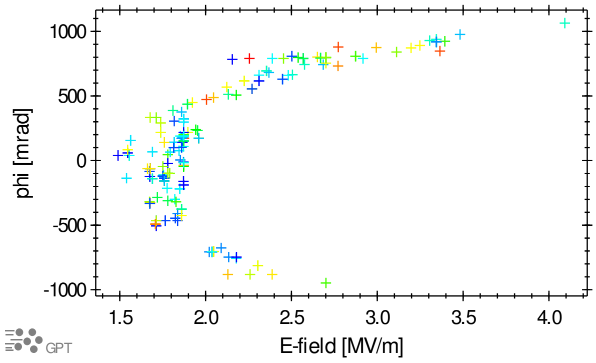

Figure 4: Intermediate result for rf-field versus phase.

Converged Result

The final result of the optimizer is shown in Fig 5, where a nicely converged Pareto front for Coherence length versus Charge is given. This collapses the *N*-dimensional variable space into a 1D parametric curve where for each point along the curve all settings (solenoids, cavity settings, etc) are known. The problem of finding the correct settings for all variables is now reduced to deciding on the 'operating point' along a 1 dimensional curve.

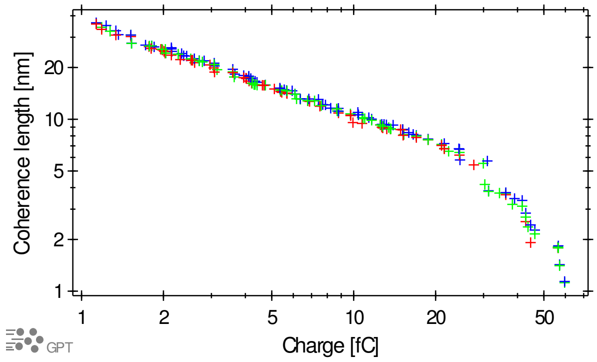

Figure 5: Final Pareto front for Charge versus Coherence length, taking into account that pulse duration must be below 100 fs and spotsize below 250 micron.

MPI IMPLEMENTATION

There is an MPI implementation of the genetic optimizer that spawns GPT runs load-balanced on MPI clusters. In order to do this, we had to relax the notion of generations and immediately spawn a GPT run the moment a new computational node is available. This can result in generations being mixed. We have seen no performance degradation because of this, and achieved almost 100% load levels on a 100-node MPI cluster for optimization problems with dozens of variables, several conflicting objectives and many constraints.

CONCLUSION

Multi-objective genetic optimization is a valuable tool to aid the design of accelerators and beamlines, particularly when there are non-linear effects and when there is mixing between the different phase-planes. We expect that continuing advances in computing power will further increase the usefulness of this approach in the foreseeable future.

REFERENCES

[1] The General Particle Tracer (GPT) code, Pulsar Physics, The Netherlands: http://www.pulsar.nl/gpt/.
[2] R. Storm, K. Price, Journal of Global Optimization, Kluwer, Vol. 11, pp. 341 (1997).
[3] Kalyanmoy Deb, IEEE Trans. on. Evolutionary comp., Vol 6, No 2, pp 182 (2002).
[4] T. van Oudheusden, P. L. E. M. Pasmans, S. B. van der Geer, M. J. de Loos, M. J. van der Wiel, and O. J. Luiten, Phys. Rev. Lett. 105, 264801 (2010).