GIOTTO: A GENETIC CODE FOR DEMANDING BEAM-DYNAMICS OPTIMIZATIONS

A. Bacci, INFN/Milan, Italy
V. Petrillo, M. Rossetti Conti, Univ. of Milan, Milano, Italy

Abstract

GIOTTO is a software based on a Genetic Algorithm (GA). Its development started in 2007 with a work published on NIMB (263, 2007, 488-496) and presented at PAC07 (THPAN031). When the parameters, defining an acceleration machine beam line, are strongly correlated in nonlinear way, the GAs are a powerful tool to coup with these difficulties. These conditions are typically generated by space-charge, as in the high brightness e-beam photo-injectors or when the Velocity Bunching compression technique (VB) is used. The power of GIOTTO is the adaptability to different cases, given by its own structure that permits to drive different external codes in series, the possibility to define a user dependent multi objective fitness function and function constraints on the beam dynamics, as well as the possibility to turn off the genetic optimization to perform statistical analysis (machine jitters). Up today it has been used in Thomson/Compton sources, ultra-short e-bunches generation by VB, focusing channel and dog-leg lines optimizations.

INTRODUCTION

In the last few years, in the field of accelerator physics and beam dynamics (BD), many works on problem solving and optimizations based on GAs have been produced. Even though the GAs have been introduced in the '70s with the well-known first monograph of John Holland (1975) [1], and already in the '90s GAs in various forms have been applied to problems in topics spanning from engineering, economics and artificial intelligence [2-4], in the accelerator physics and beam dynamics these optimization techniques are relatively young.

The improvement in the high brightness electron beams of the last few years has produced really important results in different fields, as for instance: the implementation of XUV and X rays Free Electron Lasers [5], the acceleration in plasma wave [6] or the Thomson scattering sources [7]. The production of such high quality brightness beams is always relate to an outstanding capability to tune the electron injector [8] and to cope with the bunch space charge. The space charge effects couples in a non-linear way the beam-line parameters. Giving an example, a good emittance compensation [9], downstream a Gun photon-injector, is mainly connected to the following issues: a) the laser pulse shaping at the cathode, b) the extracted charge, c) the Gun gradient, d) the Gun's solenoid intensity and e) the position of the first booster cavity. All these parameters are coupled each other in a complex way by the space-charge, making impossible a sequential optimization of the parameters; this fact is true not only from the experimental point of view, but even for the simulations. Such kind of problems, non-linear and multidimensional, are typically well faced by using GAs. Their stochastic nature prevents the convergence on weak local minima and, furthermore, the parallel structure is appealing for time consuming optimizations.

GIOTTO (Genetic Interface for OpTimising Tracking with Optics) is the evolution of a previous software, based on a GA [10, 11]. References [10, 11] have been commonly used to cite GIOTTO itself and are among the first works based on GAs for performing beam dynamics optimizations, driving simulation codes.

THE SOFTWARE

GIOTTO is based on a homemade GA. The development, done in modern Fortran, has been carried out by using high level structures, like ad hoc methods and operators, which give to the code an easily modifiable or upgradable structure [12]. GIOTTO, in most of its applications, starts by driving the tracking code Astra [13] and its generator and by using all their parameters as possible knobs for optimizations. The interface between GIOTTO and Astra, or Astra's generator, hosts in itself the capability to easily write and to easily manipulate input files. This capability has been implemented in a generalized way, making GIOTTO potentially able to drive different codes. Inside the code is in fact is present an Input Names Data Base (INDB) that can be expanded as needed, inserting all the parameters (the Input Names) necessary to write a code's input file. This methodology has been developed by using the Fortran namelist NML, so if a code is driven with NML input files, that code can be naturally used, or driven, by GIOTTO. In different way, for managing other ASCII input files it is necessary an interface.

For the sake of clarity, let we think to perform an optimization on a beam dynamic problem that copes with different laser pulse shaping (at the photocathode) and with different electron beam line settings downstream the gun. To perform such optimization, GIOTTO writes both the Astra's generator and Astra input file, then runs the two codes in sequence. This procedure has been thought to be really easily extendible to more in cascade codes. For example it has been already tested in the sequence:

ISBN 978-3-95450-147-2

Astra's generators → Astra → Fluid tracking code for plasma accelerators or Astra → Parmela [14] → ThomXcode. Moreover, GIOTTO is a parallel code, developed by using the MPI library, so just to mention, when GIOTTO runs a cascade of codes, it means that N sequences are executed, in relation to the number of nodes it is running on. GAs by their own nature are ad hoc to be parallelized.

Nowadays many GA types of software are based on the Multi Objective typology (referred to as MOGA) [15]. These kind of GAs are useful when several optimization criteria are present simultaneously and it is difficult or impossible to combine them in a single value (referred to as fitness function). In these cases, by peculiar evolutionary technique, referred as VEGA, MOEAs, MOGA, NSGA, NPGA, where acronyms and techniques are presented in Ref. [16], it is possible to find the Pareto optimal (P-optimal), which represents the ensemble of non-dominated solutions. Within the P-optimal, a user can choose the nominal machine working point, corresponding to one single element of the P-optimal. When plotted in 2D (two objectives) the non-dominant points curve (the Pareto front) is a powerful tool to better understand the machine behaviour. Usually this curve is the solution of a differential equation.

Conversely, GIOTTO has been developed following the single criterion or single objective function, and cannot be classified as a MOGA. This choice, which seems to buck the trend, brings instead some important advantages, as for example the fact that the niching strategy, inside the population, became unnecessary, simplifying the algorithm and its execution. Furthermore, in MOGA the concept of elitism is more complex and not straightforward as for a single criterion approach and the convergence on one single point simplifies all the optimization strategy. The drawback in case of more than one criteria, as typical in BD problems or beam line optimizations has been circumvented by a peculiar definition of the GIOTTO's fitness function, as explained in the following section.

THE FITNESS FUNCTION

The GIOTTO fitness function is a single criterion with respect to the goodness of a solution (of a chromosome, in GA's language). This function returns one real value which permits the chromosome sorting inside a generation and which enters in the selection rules. The chromosome showing the higher value in one generation is the best of that generation itself, and it reproductions in the next offspring gives rise, directly, to the elitism.

The fitness function is defined into the GIOTTO's input file through an equation (or more), by using the reverse polish notation (rpn) for a practical computational point of view (data stacking) and for a neater visualization. The equation can be split in more lines, simplifying complex expressions. During the code execution, each line, which is itself an equation, can be plotted, showing on line the optimization evolution. Because the GIOTTO's input file is reloaded each new generation, it is possible to change the fitness function in real time, giving to the user a really great advantage. Fig. 1 shows a section of the input file in relation to the fitness function insert mode. A routine unrolls the input-file looking for the key-word "idoneity", and, once it is found, recognizes every consecutive line as a rpn piece of the equation. The routine counts the consecutive lines, to know in how many pieces the function is split, then brings together the pieces using the last operator on each line, exactly as in the rpn. In the example reported in Fig. 1, the equation is spit in three pieces that are added up by the '+' operators at the end of the second and third expressions.

```
!*********************************
[idoneity] !must be used in rpn
emitX 2.5 / sqr -1. * exp 50 *
sigZ 0.150 / sqr -1. * exp 50 * +
sigX 2.5 / sqr -1. * exp 50 * +
```

Figure 1: Section of the GIOTTO's input file in relation to the fitness function input method

One important feature this equation insertion method is based on, and that can be fully controlled by the user, is the possibility to implement the following strategy: each piece of the equation can be related to a single optimization criterion, which moves on a Gaussian curve (Fig. 2).

This strategy is the technique used by GIOTTO to deal with multi objective problems. It consists in to assign one single optimization criterion to each piece of the equation, as shown in the Fig. 1 where, referring to an electron bunch distribution, the emittance (emitX) is assigned to the first piece, the rms longitudinal dimension (sigZ) to the second one and the rms transversal dimension to the third piece of the equation. The three pieces (themselves equations), has said above, are added up, returning one single value. The trick is to use Gaussian equations type, in such a way to define regions where the objective can be considered achieved or close to be, and regions where the objective is far to be reached. These regions are a practical way to define different weights. An almost reached objective means to be close to the goal, which corresponds to the region close to the top of the Gaussian curve, where the slope versus the parameter variation is very small. The "far region", always referring to the objective optimization, corresponds instead to the maximum slope of the Gaussian curve. With this definition of the fitness function it is very easy to change different weights in multi objective problems, driving all the optimizations by only one real value. It is straightforward to understand the potentiality of being able to change the fitness function in real time. This capability is still only in the actual GIOTTO beta version, but outstanding results [8,17,18] has already been found also keeping the fitness function unchanged, the only caution is to check that the objectives, at the optimization start, (red stars in Fig. 2, a) case) are not on the Gaussian tails.

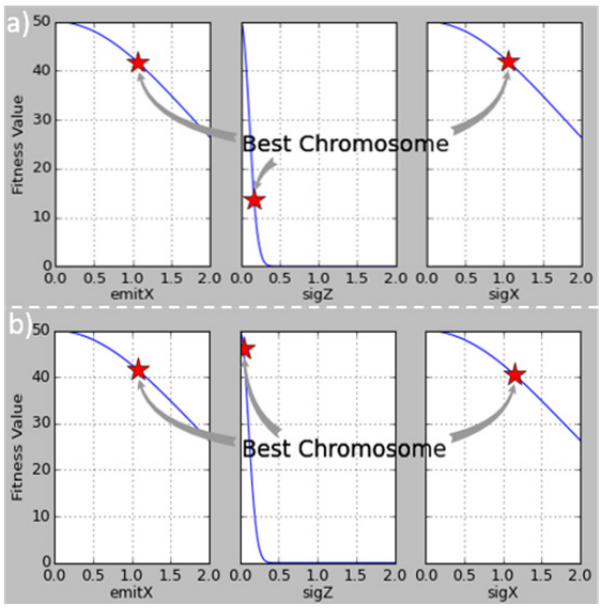

Figure 2: Graphical representation of the fitness function equation pieces. a) Offspring not yet optimized, b) offspring satisfying the optimization

Another characteristic that deserves to be mentioned is the free control of each Gaussian height, which changes the optimization slope and also defines the maximum reachable value. As shown again in Fig. 1 (pay attention to the rpn formalism), or in Fig.2, all the three Gaussian curves have a maximum height of 50, so the maximum reachable value is 150, which is reached asymptotically and defines the method that stops the optimization procedure.

This implementation of a fitness function by adding up Gaussian-like curves, one per objective and obtaining one single value, is named in this paper GAMOGA (Gaussian Multi-Objective GA). Hints of the method can be found the Ref.'s [19, 20]

GIOTTO GA'S FEATURES

GIOTTO is a Real-coded GA and uses a roulette-wheel selection, with a single point *crossover operator*. If during the selection two equal chromosomes are chosen, the selection is redone. This is a quite rare event, except for peculiar cases, when the offsprings lack in differentiation.

The optimization process starts from a first generation of chromosomes (solutions of the problem), which can be given by the user with sharp values (known good solutions of the problem) or can be generated around a central value with a variation range. Also in the case of sharp values by one known solution, the variation ranges have to be provided, because they are used by the *mutation* and *regenerator operators* (see below).

The *mutation operator* – which intervenes each new offspring, with a very small probability, but a little higher than for binary-coded GA (as commonly known) – works only on one *allele* (one parameter) per chromosome and, when the optimization stagnates, the probability to intervene is enhanced, step by step, up to a maximum value where it stays constant. The sampling around the starting *allele* value (first population) can be done with a plane probability or with a Gaussian one.

Periodically, after a number of user-defined new generation steps, the population is regenerated (*regeneration operator*) around the best chromosome, keeping alive the best chromosome itself (*elitism operator*). The regeneration, similarly to the mutation, uses the starting central *allele* and the user-defined range. To force the convergence on close solutions, a small variation of the genetic material (*alleles*) is necessary. This is done shrinking the *allele's* variation range during the regeneration. After a user-defined number of regenerations, the ranges are reopened to the original value, avoiding the stagnation in local optimization. The *regeneration operator* is very powerful and usually speeds up significantly the optimization process, but it can be also dangerous, and the parameters has to be controlled accurately. The parameters accessible to the user are: 1) after how many new generations the operator starts up, 2) the range shrink percentage and 3) after how many refining the ranges is reopened. For the sake of completeness let us see some numbers: considering a population composed by 16 chromosomes, 6 *alleles* each, and two objectives, with the three *regeneration operator* parameters set, following the above order, 25, 5, 4, after about 300 offsprings it is possible to reach the convergence; working on 2009, 16 cores, shared memory, machine, the computation takes about 3-5 hours. Without the *regeneration operator* and with a so little population (16 chromosomes) it should be impossible to reach the convergence, except with a very long computation time.

OTHER CAPABILITIES

An exhaustive description of important characteristics and capabilities of GIOTTO cannot been explained in few pages and it is not in the purposes of this proceeding. Differently, the full description of the fitness function and GA's features, as given in the previous sections, is essential to understand basic features, as for instance the fact that GIOTTO is not a MOGA and consequently the approach to multi objective problems.

It is important to point out that it is possible to insert evolution constraints, again by using equations into the input file, which have to be satisfied. Due to the parallel implementation, GIOTTO can be switched from the genetic optimization to a statistical analysis; by using this characteristic the full jitters analysis of the ELI-NP injector has been performed. GIOTTO is fully portable, by a standard Fortran native source and at the moment the MPI Windows and Linux version are released.

ISBN 978-3-95450-147-2

REFERENCES

[1] J. H. Holland, Adaptation in Natural and Artificial Systmes. Ann Arbor, MI: Univ. Michigan, 1975.

[2] S. Bengley, "Software au naturel" Newsweek, pp. 70.71, May 8, 1995.

[3] R. A. McIntyre, "Bach in Box: The evolution of four part Baroque harmony using the genetic algorithm" in Proc. IEEE Conf. Evolutionary Computat., Orlando, FL June 1994, pp., 852-857.

[4] J. Ventrella, "Disney meets Darwin – The evolution of funny animated figures", in Proc. Comput. Animat. Geneva, Switzerland, Apr. 1995, pp.35-43.

[5] Y. Ding, et al. PRL 102 (2009).

[6] M. Litos, et al, NATURE, Vol. 515, 6 Nov. 2014, pag. 93.

[7] arXiv:1407.3669 [physics.acc-ph].

[8] A. Bacci et al., J. Appl. Phys. 113 (2013) no.19.

[9] L. Serafini, J.B. Rosenzweig . PRE, Vol. 55, N. 6, Jun 1997.

[10] A. Bacci, V. Petrillo, A. R. Rossi, L. Serafini, Proceedings of PAC07, Albuquerque, New Mexico, USA, THPAN031.

[11] A. Bacci et al., Nucl. Instrum. Methods Phys. Res. B, 263,488-496 (2007).

[12] Ed Akin, book "Object Orineted Programming via Fortran 90/95", Cambridge Univ. Press, (2003).

[13] K. Floettmann, ASTRA User`s Manual, http://www.desy.de/~mpyflo/Astra_manual/.

[14] L. M. Young, LANL Report No. LA-UR-96-1835.

[15] S. B. van der Geer, M.J. de Loos, MOPJE076, Proceeding of IPAC2015, Richmond, VA, USA.

[16] M. Laumanns, et al., "Combining Convergence and Diversity in Evolutionary Multiobjective Optimization" Evolutionary computation Vol 10 No 3, pp 263 (2002).

[17] A. Bacci, A. R. Rossi, NIM-A 740 (2014) 42-47.

[18] C. Vaccarezza, et al., "OPTIMIZATION STUDIES FOR THE BEAM DYNAMIC IN THE RF LINAC OF THE ELI-NP GAMMA BEAM SYSTEM", presented at IPAC'16, Busan, Korea, May 2016, paper TUPOW042, this conference.

[19] D. Büche, N. N. Schraudolph, P. Koumoutsakos, IEEE TRAN. ON SYSTEMS, MAN, AND CYBERNETICS – PART C, APP. AND REVIEWS, VOL 35, NO. 2, MAY 2005.

[20] M. Zhang, W. Smart, Pattern Recognition Letters 27 (2006) 1266-1274.