A versatile multi-objective FLUKA optimization using Genetic Algorithms

Vasilis Vlachoudis[1,a], Guido Arnau Antoniucci[1], Serge Mathot[1], Wioletta Sandra Kozlowska[1,2] and Maurizio Vretenar[3]

[1]*Dep EN, CERN CH-1211, Switzerland*

[2]*Medical University of Vienna, Austria*

[3]*Accelerator and Technology Sector, CERN CH-1211, Switzerland*

Abstract. Quite often Monte Carlo simulation studies require a multi phase-space optimization, a complicated task, heavily relying on the operator experience and judgment. Examples of such calculations are shielding calculations with stringent conditions in the cost, in residual dose, material properties and space available, or in the medical field optimizing the dose delivered to a patient under a hadron treatment. The present paper describes our implementation inside flair[1] the advanced user interface of FLUKA[2,3] of a multi-objective Genetic Algorithm[**Erreur ! Source du renvoi introuvable.**] to facilitate the search for the optimum solution.

1 Introduction

Monte Carlo programs like FLUKA[2,3] are often used to optimize complicated problems, with multiple free parameters and variable constraints. Some examples are:

- Shielding calculations, to find the optimal material configuration and geometry to reduce to some acceptable levels the dose at the same time reducing the cost and the space required;
- Target or dumps design, to find the best configuration of materials that can withstand the power of the beam, with reduce waste production and provide at the same time the best physics parameters;
- Optimize the radiotherapy treatment planning to deliver a prescribed dose to a volume of interest and minimizing the dose to nearby organs;
- and many others.

Such complicated simulation tasks, requires an experienced user to provide some initial educative guesses, and a large number of iterations until all the constraints are met and the optimum solution is found. In numerical analysis there are several optimization techniques that could be used to facilitate the task, each one with their advantages and drawbacks. The advantage of GA over the other algorithms that makes it suitable for Monte Carlo calculations is the no-dependence on gradients and derivatives, while every step is performed using random numbers. The present paper describes our implementation of a generic Genetic Algorithm [**Erreur ! Source du renvoi introuvable.**] solution that works seamlessly with the FLUKA Monte Carlo code.

2 Genetic Algorithm Implementation

In the field of artificial intelligence, Genetic Algorithm (GA) is a search that mimics the process of natural selection. It is commonly used to optimize problems using techniques inspired by natural evolution. The algorithm is based on a population of individuals, which is repeatedly modified/evolved in its environment until the target solution is reached. Our implementation of GA is written in Python as an extension to flair[1] the advanced user interface of FLUKA. Python[5] being an interpreted language is not as fast when compared to the compiled ones like C/C++, Fortran, however the CPU penalty that is added by using Python is minimal compared the time spent in FLUKA to evaluate each solution. However our implementation provides also a mechanism to fully parallelize the evaluation of each individual and takes care of the synchronization of the multiple threads. In the following sections we briefly described some of the terminology and our implementation.

2.1 Environment

The environment is the master class that contains the population, includes the basic parameters settings and provides the methods controlling the evolution of our population from generation to generation. The user can define the population size, maximum number of generations, as well as probabilities rates for crossover and mutation, and the goal for optimum solution. The class on initialization creates the initial population of random individuals, or it can be provided externally. Once everything is initialized the main loop is invoked.

[a] Corresponding author: Vasilis.Vlachoudis@cern.ch

At each iteration it is evaluating all individuals either in parallel or in synchronous order. Each individual evaluation requires one dedicated FLUKA job with the parameters defined in the chromosome of the individual. As soon as all the individuals are evaluated, if the desired goal is met the process stops.
Otherwise all individuals are sorted, based on their fitness function score, with higher preference the best individuals are recombined together (crossover) to produce new children individuals. In addition, randomly based on the mutation ration, some individuals are mutated, their chromosome are altered randomly according to a certain user defined pattern.
In our implementation and on user's choice we keep the best individuals untouched from one generation to another, the so called elitist selection.
The genetic algorithms have a tendency to converge towards the first local minimum that will find, unless if a high mutation rate is employed. However, with high mutation rates the converging process takes longer and can result in diverging from the solution. From lots of testing, we discovered that, as in the elitist approach, if we replace the worst individual with completely random one, it helped to converge faster, like in the case of smaller mutation rate but providing a better result.

2.2 Individual

Each individual is described by a chromosome which is represented as a string of alleles (genes).

2.2.1 Allele

Each allele is a python class that holds one or more parameters. In our implementation we have the following predefined alleles:

- Binary: holding a true/false value;
- Range: holding an arbitrary range of values from min to max;
- Integer: holding a integer value within a range
- Real-uniform: holding a floating point variable with a uniform distribution within a range;
- Real-Gaussian: holding a floating point number sampled from a normal distribution of values within a defined range;
- List: a list of predefined choices, can be anything of integers, reals or strings (e.g. material names)

However, the user can easily superclass the base allele class and create his own. Each allele must provide two methods, one for random assignment and one for mutation. In flair each allele is represented inside the FLUKA input file with a **#define** variable. A recent addition in flair is the extension of the pre-processor language giving the ability to perform complicated function evaluation for scalar, vectors or matrices and dynamically alter the settings inside the input file. The GA implementation is highly profiting from this addition to create parametric inputs for each individual.

2.2.2 Crossover

Crossover is the mating of two selected parent individuals for generating the child individuals of the next generation. There are numerous algorithms for that, currently we have implemented the single and two point crossover methods. Table 1 shows an example of a two point crossover method, where the two parent individuals P_1, P_2 mate to generate two new children C_1, C_2.

Table 1: Example of two-point crossover. Pivot points are represented with a double line.

P_1:	A	B	C	D	E	F	G	H	I
P_2:	a	b	c	d	e	f	g	h	I
C_1:	A	B	C	d	e	f	g	H	I
C_2:	a	b	c	D	E	F	G	h	I

Two pivot points are randomly sampled over the string of alleles and each child is composed by the sub-strings limited by the two pivot points alternating from the two parents.

2.2.3 Mutation

Mutation is a random alteration of an allele to a new random value. It is useful to enlarge the phase space of the search and to go out of local minima. Mutation typically has a very low probability rate typically 0.2-0.3%, if it is increased too much the search might fail to converge it is equivalent of increasing a lot the entropy of the system.

2.2.4 Fitness function

The evaluation or the cost of each individual is called "*fitness function*", which is the objective function we are trying to optimize (minimize or maximize) and it is user defined. To evaluate the fitness function in flair, each individual will create a dedicated FLUKA input file with the alleles appearing as **#define** cards in the beginning. Flair will launch the simulation with the input file and in the end collect and process the results. The output of the simulation is provided as input to the fitness function, which in turn evaluates the cost for the individual under study.
The fitness function is the most complicated part of the process, not from the technical point of view but from the conceptual. The user has to provide a fitness function that will summarize, classify within a single number the outcome of the simulation. For example, it is extremely difficult to relate the financial cost versus the physics optimization of the project or safety requirements. Inside flair we provide several methods of extracting the simulated results, in an easy way, however the user has to do the final python coding of the function. Multiple objectives can be described as the linear weighted sum of simpler quantities.

3 Test cases

Extensive tests have been carried out for the correct functioning of the algorithm as standalone or combined with FLUKA, for the fine tuning of the various parameters.

3.1 Parameter tuning

A simplified Monte Carlo optimization problem was devised to provide fast convergence and simulate the effect of each parameter. However, there is no generic guideline on having a unique set of parameters that works best for each problem. The individual tests where ranging from simple binary search problems, function fitting, travelling salesman problem etc..

The problem was consisting of a modified traveller salesman problem, with each node instead of having a fixed location, it includes also an uncertainty, fuzziness in the positioning. The program has to find the minimum path joining all nodes.

In the first run with 100 individuals we kept all parameters zero and with modified the cross-rate. Fig. 1 shows the spread of solutions from all individuals versus the generation number. As it is evident from the plot, the process quickly comes into convergence to the optimum solution based on the starting conditions. What is to remark here is that the higher the cross over parameter the steeper is the gradient of convergence. However, the final value stops on the first local minimum, and all solutions converge slowly to the same value. It is visible that there is no relation on the optimum reached and cross over rate.

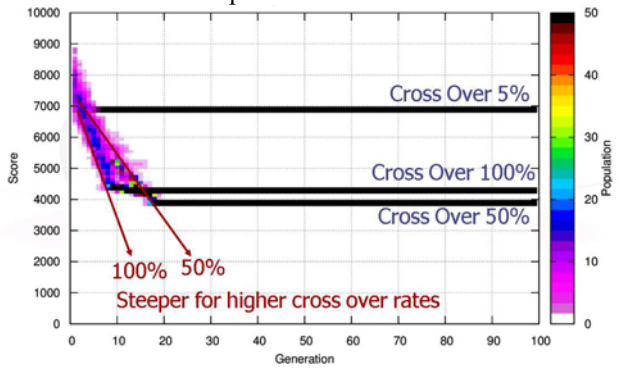

Figure 1. Cross Over parameter testing

In order to allow some individuals to escape the local minimum the mutation parameter is important, which provides extra perturbation to the system. In Fig. 2 is visible that high mutation rates of 10%, enlarges the spectrum of solutions however no good convergegnce can be achieved.

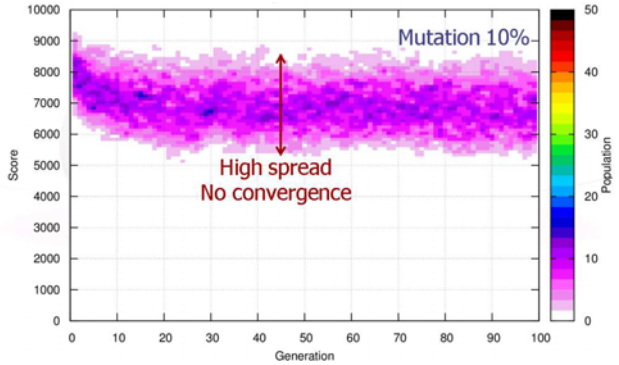

Figure 2. High mutation rate 10%, with cross-over 90%

Lowering the mutation rate, the spread becomes narrower. Also the smaller and slower rate of mutation leaves time to the cross over to minimize to the next local minimum Fig. 3. Lowering too much leads to a slower convergence.

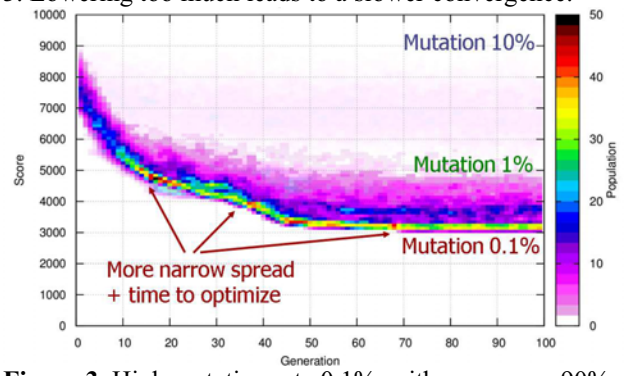

Figure 3. High mutation rate 0.1%, with cross-over 90%

The addition of elitism, where the most fitted solution is copied as such to the next generation ensure the best individual to be propagated from one generation to another. Furthermore, replacing the less fitted individuals with completely new purely random ones, helps in scanning better the phase space of solutions.

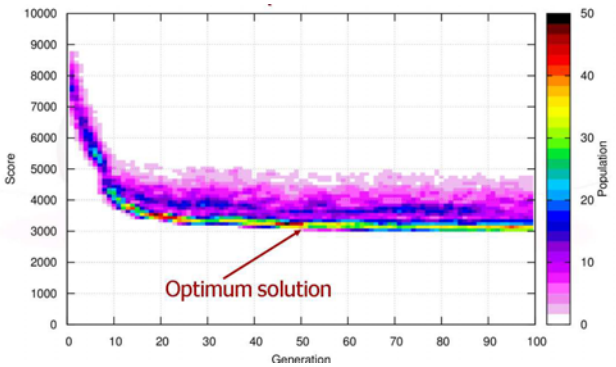

Figure 4. Combining all factors together, cross over 90%, mutation 0.1%, elitism 1, random 5.

Fig. 4 shows the convergence when all parameters are set close to the optimum, which an optimum solution reached already from the 50th generation.

Next we are going to describe two test cases in combination with FLUKA, where the code was used with success, as well as discussing the drawbacks and benefits.

3.2 HF-RFQ shielding

A possible RFQ-based accelerator[6] designed for the production of PET isotopes would be made of two directly coupled high frequency RFQs delivering a proton beam of 10 MeV in energy in a length of less than 5 meters (Fig.5). The main beam parameters are: proton energy: 10 MeV, average current: 20 μA, peak current: 500 μA, RF frequency: 750 MHz, duty cycle: 4 %.
This RFQ could be used as a compact accelerator for producing ^{18}F via the reaction $[^{18}O]H_2O(p,n)^{18}F$: 6.3 GBq/μA. Below 12 MeV, the neutron production rate is equivalent to the ^{18}F saturation yield [7], and for an average of 20 μA current we can consider a neutron

emission rate at saturation of 1.26 10^{11} n/s, with the spectrum as in Fig 6.

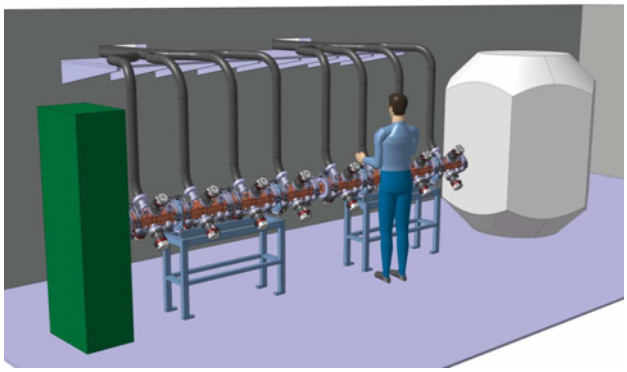

Figure 5. HF-RFQ conceptual design

The aim of the FLUKA simulation study was to find an adequate compact shielding, minimizing the dose from the emerging neutrons so the generator could be installed inside a hospital respecting the radioprotection requirements.

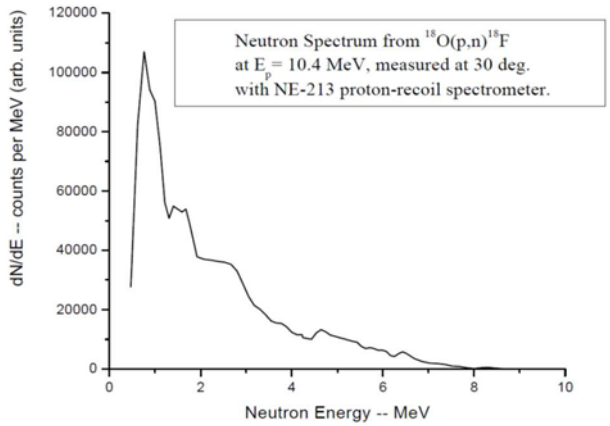

Figure 6. Neutron spectrum from $^{18}O(p,n)^{18}F$ reaction at E_p=10.4 MeV

3.2.1 *Setup*

A special source routine was written to model the outgoing neutron spectrum from Fig. 6 assuming isotropic angular distribution. The neutron target was placed in a shell structure, onion like spherical geometry like in Fig. 7, where the thickness of each layer as well the material was dynamically assigned via **#define** cards Fig. 8. Several materials were considered Polyethylene, Borated-Polyethylene, Iron, Copper, Tungsten, Concrete etc.
To save CPU time, the electromagnetic thresholds were raised, which leads to underestimation of the final dose. However it will not affect the concept of the shielding, since the neutrons are the ones responsible for the dose at the outer layers. Once a solution was attainted, the full simulation using also the electromagnetic part with normal thresholds was performed. Importance biasing was also used to enhance the results in the outer layers. Finally, a spherical scoring was used to enhance the statistics, profiting from the spherical symmetry.

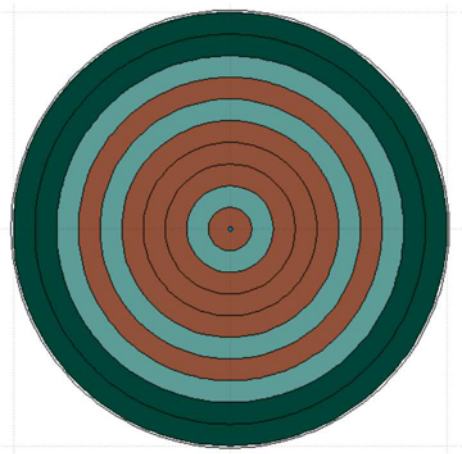

Figure 7. Geometry of the optimum solution. Cyan=Polyethylene, Brown=Iron, Green=Borated Polyethylene, every ring increases by 10cm the radius

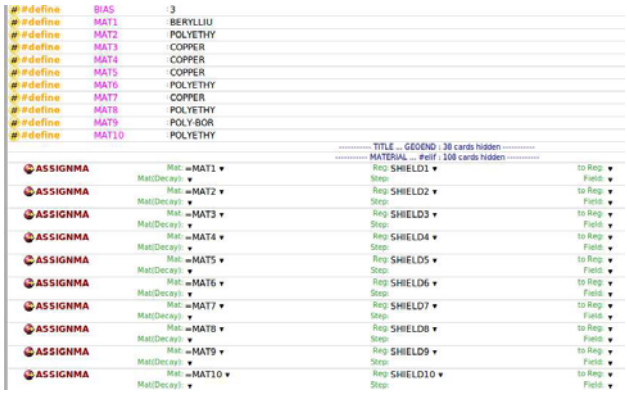

Figure 8. Snippet of the input file showing the dynamic definition of the materials to the various shells.

3.2.2 *Fitness function*

As a fitness function we used a weighted combination of two goals: *i)* minimizing the external diameter of the shielding, and *ii)* ensuring that the dose stays below the limit of 1μSv/h.

3.2.3 *Results*

A first run was performed with a population of 60 individuals and 40 generations, which resulted in a total of 2400 FLUKA runs. Using our cluster, each run lasted about 10min without the electromagnetic part. With 60 jobs running in parallel the total time for the first step of the optimization was about 8h, even though as it can be seen on Fig. 9, the fittest solution was already reached already from the 10th generation. The elitist selection guaranteed to keep this individual until the last generation. The first step provided a possible solution, which in later had to be refined with a complete run, including the electromagnetic transport, generating an initial population with individuals nearby the optimum solution of the first step (Fig 10). The total time of the second step was 10 times longer. The final solution resulted from the GA was a sandwich with the materials from inside to outside, polyethylene, copper and borated polyethylene (Fig. 7).

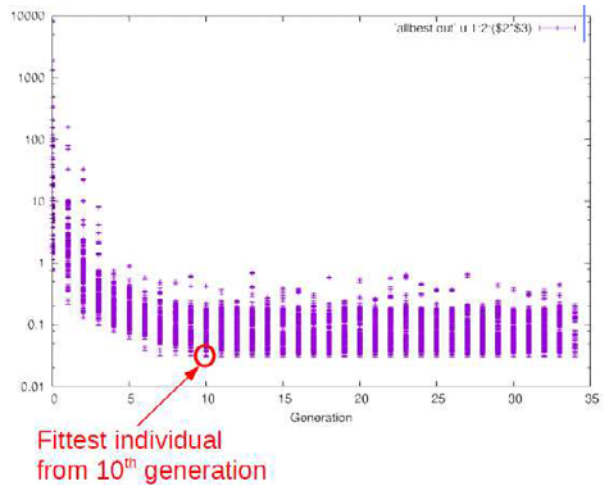

Figure 9. Fitness score evolution of 1st step transporting only neutrons versus the generation number

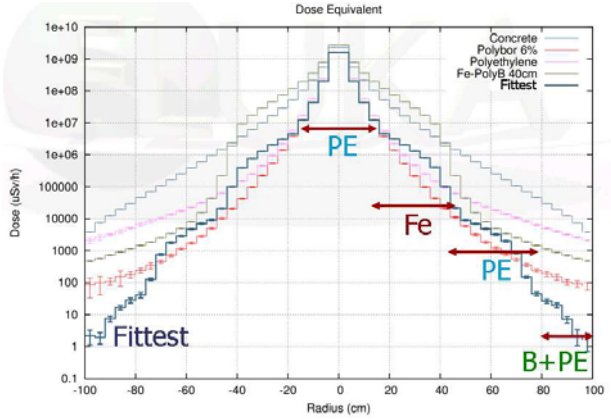

Figure 10. Dose equivalent versus radius for the optimum solution of the 2nd step, performing a full simulation with no cuts or simplifications.

3.3 Treatment Planning System optimization

Commercial Treatment Planning Systems for hadron therapy provide fast results using analytical models and parametric fits of the physical processes. On the other hand Monte Carlo provides superior results due to the detailed description of the physics, however it suffers from long execution times. In this project we are working in enhancing further the optimization result from the TPS using FLUKA based on the previous work [8]. The goal was to fine tune the particle intensity per spot to better approach the prescribed dose on the Planning Target Volume (PTV) and the same time minimizing the dose received by the Organs At Risk (OAR).

3.3.1 Simulation Setup

The simulation setup consisted on calculating the dose per each spot position on a 3D mesh binning, using the real voxel geometry generated from the DICOM-CT and the RT-PLAN. The output of the simulations was provided as input to the pre-optimizer [9]. The pre-optimizer is merging the 3D mesh binning's weighted by the individual intensity for each spot which we want to optimize.

3.3.2 Fitness function

The fitness function is the sum of the square of differences from the prescribed dose in the PTV plus the differences of the dose on the OAR times the Heaviside θ step function.

$$\chi(\boldsymbol{N}) = \sum_{i\,\epsilon PTV} \left(\widehat{D_{PTV}} - D_i\right)^2 + \sum_{j\,\epsilon OAR} \left(\widehat{D_{OAR}} - D_j\right)^2 \times \theta\left(\widehat{D_{OAR}} - D_j\right) \quad (1)$$

3.3.3 Results

Two algorithms have been tested, a steep descent optimization and the genetic algorithm. The steep descent is calculating the maximum gradient direction of the fitness function, and using the golden-bisection algorithm the minimum is searched along this direction.
When a minimum is reached along the maximum gradient direction, the maximum gradient is recalculated at the minimum position, and the algorithm is repeated until a convergence is achieved.

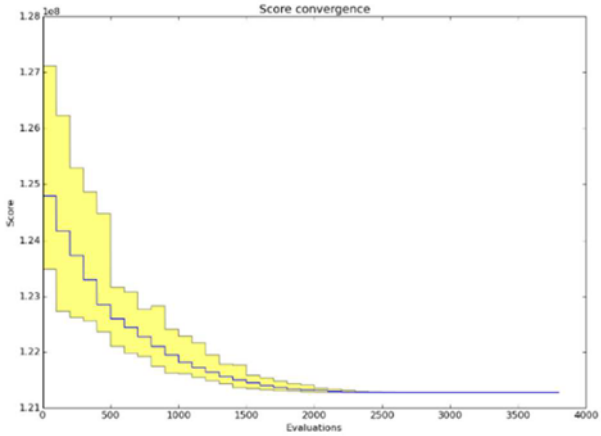

Figure 11. Solution convergence versus the number of evaluations

In the Genetic Algorithm each chromosome is represented as a vector of the beam intensities per spot. Running 60 individuals for 30 generations resulting in 1800 evaluations (Fig.11). The GA optimized the result much faster and gave a better convergence than the steep descent which stuck in a local minimum.

4 Conclusions

The present paper discusses the work in implementing a generic Genetic Algorithm approach directly coupled with the FLUKA Monte Carlo code via the flair interface. The implementation proved to be versatile enough to perform with ease optimization simulations. The main complications arises from the definition of the fitness function, which highly depends on the problem. It can range from either easily defined or being extremely complicated, especially when completely different requirements have to be merged together and represented by a single number.
The current implementation is able to work either in online mode like in the presented case of the HF-RFQ that the fitness function is calculated from the outcome of each simulation, or in offline mode like in the TPS optimization

case that first a large number of simulations is performed and then the GA is optimizing the combination of the simulations.
The main drawback of the GA is the requirement of a lot of iterations with each one requiring a full simulation with reasonable statistics. The CPU time needed can be considerable, unless if drastic optimizations in the simulation input are performed.
The benefit of using GA is that once the fitness function is defined the optimization procedure is almost autonomous. Since GA follows a random path approach, the optimization is converging as a Poisson distribution, with faster convergence when dealing with multiple free degrees of freedom to optimize.

References

1. V.Vlachoudis "*FLAIR: A Powerful But User Friendly Graphical Interface For FLUKA*" Proc. Int. Conf. on Math, Comp Meth & Reactor Physics, Saratoga Springs, NY 2009
2. G. Battistoni et al., "*The FLUKA code: Description and benchmarking*" Proc. Hadronic Shower Simulation Workshop, Fermilab 6-8 Sept 2006, IP Conf Proc 896, 31-49, (2007)
3. A. Fassò et al., "*FLUKA: a multi-particle transport code*" CERN-2005-10 (2005), INFN/TC_05/11, SLAC-R-773
4. Holland, John H. "*Adaptation in natural and artificial systems. An introductory analysis with application to biology, control, and artificial intelligence*" Ann Arbor, MI: University of Michigan Press (1975)
5. http://www.python.org
6. A.M.Lombardi et al. "Beam Dynamics in a High Frequency RFQ", Proc. of IPAC2015, Richmond, VA, USA (2015)
7. L.R.Carroll, "*Estimating the Radiation Source Term for PET Isotope Targets*", Workshop of Targetry and Target Chemistry, Triumf, CA, 2002
8. A.Mairani et al., "*A Monte Carlo-based treatment planning tool for proton therapy*", IOP Publishing 2013, Physics in Medicine and biology, Vol 58.
9. G.Arnau Antoniucci, "*FLUKA Pre-Optimizer for a Monte Carlo Treatment Planning System*", Openlab CERN Summer Student Report Sep 2015.