Foam: A General purpose Monte Carlo Cellular Algorithm

S. Jadach

aHenryk Niewodniczanski Institute of Nuclear Physics, ul. Radzikowskiego 152, 31-342 Cracow, Poland

A general-purpose, self-adapting Monte Carlo (MC) algorithm implemented in the program Foam is described. The high efficiency of the MC, that is small maximum weight or variance of the MC weight is achieved by means of dividing the integration domain into small cells. The cells can be n-dimensional simplices, hyperrectangles or a Cartesian product of them. The grid of cells, “foam”, is produced in the process of the binary split of the cells. The next cell to be divided and the position/direction of the division hyperplane is chosen by the algorithm which optimizes the ratio of the maximum weight to the average weight or (optionally) the total variance. The algorithm is able to deal, in principle, with an arbitrary pattern of the singularities in the distribution.

Introduction and motivation

For the problem of function minimalization of the arbitrary user-function one applies MINUIT or other program from the NAG library. One finds also many general purpose programs for integration of “arbitrary” integrand. The above general-purpose tools work, in principle, for a very wide range of the user-functions. For multi-dimensional Monte Carlo simulation problem, that is for the problem of generating randomly points according to n-dimensional distribution, there is precious little examples of the general purpose Monte Carlo Simulators (GPMCS), that is programs which work (in principle) for an arbitrary integrand.

Inevitably the GPMCS has to work in 2 stages: “exploration” and “generation”. During exploration GPMCS is “digesting” the entire shape of the n-dimensional distribution \( \rho(x_1, x_2, ... x_n) \) to be generated and memorizes it as efficiently as possible using all CPU power and memory available. Obviously, for the memorized \( \rho'(x_1, x_2, ... x_n) \) a method of the MC generation of the points \( \vec{x} \) exactly according to \( \rho'(\vec{x}) \), has to be available. The quality of the distribution of the weight \( w = \rho/\rho' \) for events in the generation (small variance, good ratio of maximum to average, etc.) is determined by the algorithm of the exploration. In other words, the quality of the “target weight distribution” in the latter generation is determining/driving the algorithm of the former exploration.

At the time of writing up this contribution a detailed description of the algorithm and the program is available in ref. [1]. The original, less advanced version was described in ref. [2]. The other old and new efforts in the area of general purpose MC methods see refs. [3–9]

Cellular algorithm of the Foam

The most obvious method to minimize the variance (or maximum weight) of the target weight distribution in generation is to split integration domain into many cells\(^2\), such that \( \rho(\vec{x}) \) is approximated by constant \( \rho'(\vec{x}) \) within each cell. This is a cellular class of general purpose MC algorithms\(^3\) which always involves a kind of cellular exploration of the user defined distribution.

The obvious questions are: what kind (shape) of cells to use and how to cover the space with cells? In Foam algorithm/program three types of cells are used: pure simplices, pure hyperrectangles and Cartesian products of them. Cells of this kind can be parametrized rather easily and the related computer memory consumption is not ex-

---

\(^1\)Work partly supported by the European Community’s Human Potential Programme under contract HPRN-CT-2000-00149 “Physics at Colliders”, by Polish Government grants KBN 5P03B09320 and 2P03B00122, and by NATO grant PST.CLG.977751.

\(^2\)This general idea is at least 40 years old.

\(^3\)“Stratified sampling”, used in the literature, has a narrower meaning.
The system of cells can be created all at once (like in VEGAS [3]) or in an evolutionary way, by the “splitting process”. The Foam algorithm does the binary split of cells. Next cell to be split is chosen by examining the “target weight distribution”, that is of the event generation in the latter phase. The binary split provides automatically the full coverage of the space, simply because the primary “root cell” is identical with the entire integration domain and there is never any mismatch between parent cell and two daughter cells.

Another important issue is the question of the variance reduction versus maximum weight reduction. In construction of the Foam algorithm we have put most effort on the minimization of the ratio of the maximum weight to the average weight $w_{\text{max}}/\langle w \rangle$. This parameter is essential, if we want to transform $w$-ted events into $w = 1$ events, at the latter stage of the MC generation. Minimizing maximum weight is not the same as minimizing variance $\sigma = \sqrt{\langle w^2 \rangle - \langle w \rangle^2}$. Usually minimizing $w_{\text{max}}$ is more difficult. In Foam minimizing variance is also implemented and optionally available. It can be useful in the case when a MC with $w$-ted events is acceptable. In fig. 1 we show two examples of the weight distribution evolution in the Foam, when adding more and more

cells.

In the Foam algorithm we define two auxiliary distributions $\rho'(x)$ and $\rho_{\text{loss}}(x)$ related to integrand $\rho(x)$. Both are constructed together step by step, simultaneously with the construction of the foam of cells, in the exploration process. When our ultimate aim is to minimize $w_{\text{max}}$, we define

$$\rho'(x) \equiv \max_{y \in \text{Cell}_I} \rho(y), \text{ for } x \in \text{Cell}_I,$$

$$R_{\text{loss}} = \int d^nx \left[ \rho'(x) - \rho(x) \right] = \int d^nx \rho_{\text{loss}}(x).$$

here $\rho_{\text{loss}}$ is the difference between the “ceiling distribution” $\rho'$ and the actual distribution $\rho$ from which it is derived. The rejection rate in the final MC run is just proportional to $\int \rho_{\text{loss}}(x)$ by construction, i.e. the rejection rate $R_{\text{loss}}/R$. Also, $\rho_{\text{loss}}(x)$ has a clear geometrical meaning, see below. When our principal aim is to minimize the ratio of the variance to the average of the weight, $\sigma/\langle w \rangle$, in the final MC generation, we are led to the following definition:

$$\rho'(x) \equiv \sqrt{\langle \rho^2 \rangle_I} - \langle \rho \rangle_I, \text{ for } x \in \text{Cell}_I,$$

$$\rho_{\text{loss}}(x) \equiv \sqrt{\langle \rho^2 \rangle_I} - \langle \rho \rangle_I, \text{ for } x \in \text{Cell}_I.$$

The average $\langle \cdots \rangle_I$ is over the $I$-th cell$^4$. The ratio

$^4$See ref. [1] for a detailed derivation of the above prescription.
\( \sigma/\langle w \rangle \) in the final MC generation is a monotonous ascending function of \( R_{\text{loss}} = \int \rho_{\text{loss}}(x) dx^n \) – hence, minimization of \( R_{\text{loss}} \) is equivalent to minimization of \( \sigma/\langle w \rangle \).

The basic rules governing binary split of a cell is that each split of a cell: \( \omega \rightarrow \omega' + \omega'' \) should decrease the total \( R_{\text{loss}} \) and the decrease should be as big as possible. How to get the best total decrease \( \Delta R_{\text{loss}} \)?

1. For each next cell split we choose a cell with the biggest \( R_{\text{loss}} \).
2. Position/direction of a plane dividing a parent cell into two daughter cells is chosen to get the smallest total \( R_{\text{loss}} \).

The actual procedure relies on the small MC exercise done for each single cell during its exploration, in which events are generated with the flat distribution, weighted with \( \rho \) and projected onto \( n \) (simplicial case) or \( n(n+1)/2 \) (hyperrectangular case) of the cells. Resulting projection histograms are then analysed and the best “division geometry” is found, for which the estimate of \( \Delta R_{\text{loss}} \) is calculated. How the geometry of the division algorithm looks like? In case of a \( n \)-dimensional simplex defined by \( n+1 \) vertices, a pair of vertices \( x_i \) and \( x_j \) is chosen and a new vertex \( Y \) is put somewhere on the line in between:

\[
Y = \lambda x_i + (1 - \lambda) x_j, \quad 0 < \lambda < 1.
\]

Two daughter simplices are defined with two list of vertices:

\[
(x_1, \ldots, x_{i-1}, Y, x_{i+1}, \ldots, x_j, x_{j+1}, \ldots, x_n, x_{n+1})
\]

and

\[
(x_1, \ldots, x_{i-1}, x_{i+1}, \ldots, x_{j-1}, Y, x_{j+1}, \ldots, x_n, x_{n+1}).
\]

Now, how do we choose a pair \((i, j)\) and the value of \(\lambda\)? A short sample of the MC events (100-1000) is generated \(\in\) cell. Each MC point is projected (see fig. 2) \( X \rightarrow Y \) onto an edge \((i, j), i \neq j \). For each \((i, j)\) the \(dN/d\lambda\) is histogrammed, and its “loss” functional \( R_{\text{loss}} \) is estimated. The optimal edge \((i, j)\) with the biggest loss is selected. For the optimal edge the cell division ration \(\lambda\) is deduced from its histogram (\(\lambda\) is always a rational number, \(n/N_{\text{bin}}\)).

In fig. 3 we show several examples of the evolution of the 2-dimensional foam of cells in case of simplices (triangles) and rectangles for up to 2500 cells.

The important practical question is whether hyperrectangular cells or simplicial cells are better as a building blocks. There is no simple answer. Simplices are limited to low dimensions \(n < 6\) because calculation of increasingly large number of determinants leads to much CPU time consumption. In addition, for simplices memory consumption is \(\sim 16n\) Bytes/Cell while for hyperrectangles memory consumption can be limited to below 50Bytes/Cell independently of \(n\). See ref. [1] for details. Experimenting with many testing functions has also shown that quite often hyperrectangles provide final MC efficiency as good or sometimes even better than for simplices.

The method of cell division relying on the analysis of the projection histograms requires many function calls in the cell exploration. The overall efficiency of the final MC is best for large number of cells. That points towards necessity of the large number of the function calls during the exploration, which may lead to excessive CPU time consumption. The following efficient trick was found to limit number of the function calls in the cell exploration: During MC exploration of a given new cell we continuously monitor the number of accumulated effective \(W = 1\) events, \(N_{\text{eff}} = (\sum w_i)^2 \sum w_i^2\), and stop explo-
ration when $N_{eff}/n_{bin} > 25$, where $n_{bin}$ is the number of bins in each histogram used to estimate the best division direction/edge and parameter. In this way the increase of $N_{samp}$ is not wasted for cells inside which the integrand is already varying very little.

**Programs**

At present the main development version of the Foam implementation is available in the C++ language. Two other versions written in Fortran 77 are also available. They are functionally equivalent to version in C++.5

**Conclusions**

Foam algorithm is a versatile adaptive, general-purpose type of an algorithm based on the cellular division of the integration domain. The geometry of the “foam of cells” is rather simple, cells of simplical and/or hyperrectangular shape are constructed in the process of a binary split. It works in principle for arbitrary distribution — no assumption of factorizability as in VEGAS of Ref. [3]. Encoding cells with memory-efficient methods allows up to $\sim 10^6$ cells to be built in the computer memory of a typical desktop computer.

**REFERENCES**


5Future development will be restricted to C++.