On the efficiency of a parallelized Hit And Run Algorithm for convex polytopes*

Mario Vazquez Corte[1] and Luis V Montiel[2]

Abstract— **This paper presents a parallelized verios of the Monte Carlo Hit-and-Run algorithm for sampling convex bounded polytopes. Recent literature suggest that that Hit-and-Run algorithms posses better properties than the widely used methods like maximum entropy and copulas. To asses the performance of the new algorithm it is compared against the hitandrun R package that is used in operation research literature. A Docker image based in a Linux distribution is provided along with the algorithm in C language and the necessary libraries to run it.**

I. INTRODUCTION

Sampling algorithms are widely used to solve many abstract and applied mathematical problems like estimating probability distributions, optimizing under partial information[8], and approximating a polytope's volume[1][2]. This kind of problems suffer from the curse of dimensionality which means that the volume of the studied space increases so fast, relatively to the low-dimensional setting, that the available data becomes sparse in higher dimensions[3]. Sampling algorithms require sets of data that are representative of the studied space, and thus an increase in dimensionality must be accompanied with an exponential growth of samples[4]. Markov Chain Monte Carlo methods (**MCMC**), which rely on repeated random sampling, are one of the most used techniques available to solve this kind of problems [1][2][7]. The Hit-and-Run sampler (**HAR**), a subcategory of MCMC algorithms, has proven to be one of the most efficient and flexible approaches[5][6]. It presents many desirable mathematical conditions like uniform sampling[5] and flexibility to incorporate many types of constraints[6].

In general a **HAR** sampler can be used to study any n-dimensional convex bounded polytope by repeatedly sampling the polytope's interior, eventually, this sampling leads to a representative collection of points that live in the studied space. A polytope is the n-dimensional generalization of a polygon or polyhedron, or a geometric object with flat sides. Convex polytopes can be easily defined as the intersection of a set of half-spaces and thus can be represented by a set of linear inequalities. If a polytope is bounded it means there exists a ball of finite radius that contains the polytope in its entirety. For simplicity purposes any reference to the word polytope must be understood as a convex bounded polytope, and the Hit-and-Run sampler for convex polytopes will be abbreviated as **HAR** unless stated otherwise.

In recent years, the hit-and-run algorithm has been used to solve many applied decision problems under uncertainty. This include sampling the set of possible joint probability distributions given partial information[8] [14], weight generation for simulation-based multiple criteria decision analysis[6], and exploring the set of possible allocations in a negotiation game[15]. This class of problems have been studied under the scope of two closely related alternatives [7]: maximum entropy [13][11] and copulas[12][16]. This two approaches yield a single probability distribution or kernel which means they return a single interior point of the polytope. The wide spread use of both methods is due to their efficiency, and possible closed mathematical results under some assumptions. Recent literature has studied under which circumstances the assumptions made by maximum entropy algorithms, which may use copula methods, are not be adequate[9]. On one hand, the main advantage of the **HAR** over the forementioned methods is the great collection of interior points it yields which can be used to analyze different scenarios under relaxed assumptions[17]. For example, the **HAR** can give the decision maker more information about the polytope like many possible joint probability distributions or utility coefficients that satisfy the constraints. An other example is the widespread use of MCMC algorithms in Bayesian inference to obtain robust posterior probability distributions[18]. On the other hand, **HAR** is more costly in computational terms since it needs a big collection of points to represent the polytope, few samples are not necessarily useful. In general, **HAR** algorithms give the decision maker more flexibility to study the phenomena under less restrictive assumptions while guaranteeing a robust representation of the studied space but suffer from curse of dimensionality[18]. Hence, the necessity to increase the sampling performance of such algorithms.

Historically, many problems that suffer from the curse of dimensionality have been alleviated by the parallel programming paradigm. Two of the most prominent examples are Scientific Computing algorithms and Machine Learning algorithms. This procedures have taken advantage of the multicore and manycore architectures, and General Purpose Graphics Processor Units (GPGPUs) to enable simultaneous operations like massive concurrent floating point calculations at each step. The usefulness of parallel architectures have spawned a variety of libraries, such as OpenBLAS, CUBLAS and MAGMA, that focus on the optimization of floating point operations. This libraries are optimized versions of the BLAS and LAPACK open-source scientific computing libraries that are free of charge. Although, CUBLAS and MAGMA need NVIDIA GPGPUs.

This paper presents a parallelized version of the Monte Carlo hit-and-run algorithm (**PHAR**) which focuses on uniformly sampling convex compact polytopes in $\mathbb{R}^n$ defined by a set of linear constraints. This constraints may represent partial information known by the decision maker like known moments of a discrete probability distribution, partial derivatives of a polynomial or min/max values of a decision variable. The new algorithm is a modified version of the one presented in [10]. The handling and the calculation of the projection matrix is modified in order to reduce the communication between the host(CPU) and the devices (GPGPUs). Also, several random walks over the polytope are carried out simultaneously to take advantage of the efficient matrix-to-matrix operations the GPGPUs are capable of.

A Docker image along with the code in C language and the necessary libraries to run it is presented. This code is capable of adapting to many specific architectures. For example, matrix-vector operations may be carried out using the OpenBLAS library or CUBLAS hinge on the type and availability of the GPGPUs but implemented with MAGMA routines if there are multiple GPUs available. The number of walks is automatically calculated by the algorithm depending on the available resources. The optimized libraries mentioned before are used along with the LPSOLVE library, depending on the context and the requirements of the operations. The **PHAR** also includes the thinning factor suggested in [6] to yield uncorrelated samples. We managed to obtain an speed up factor of ...(i need to run more experiments to obtain some viable results and finish this paragraph)

A non parallelized version of the hit-and-run algorithm is presented in [6] as an R-package called hitandrun. This package contains many other useful functions along with a hit-and-run sampler. This function will be abbreviated as (**RHAR**) and used as our main criterion to compare **PHAR** against. Four metrics are used to asses the uniformity of the sample produced by the **PHAR** algorithm which are also used in [6]. This metrics are Friedman-Rafsky two-sample Minimal Spanning Tree (**MST**) test, Coefficient Variation, Component Wise Error (**SCE**) and sample autocorrelation.

To show the usefulness and efficiency of the **PHAR** it is tested in an other scenario. The **PHAR** is used for approximating continuous functions using Bernstein polynomial under incomplete information, as long as the Bernstain coefficients can be represented as a set of linear constrains. This was motivated by the fact that estimating multiple attribute utility functions is one of the fundamental problems in Decision Theory, and also suffers from the curse of dimensionality. Approximating such functions using **HAR** methods is computationally costly. Furthermore, in this experiment the maximum entropy method, which is the most widely methodology used in this context, is compared against **PHAR**. For reproducibility every experiment was carried out using the Docker image mentioned before using specified AWS EC2 instances with access to zero, one or two NVIDIA GPGPUs.

The contributions of this work are as follows:

- The **PHAR**, a parallelized version of the **HAR** for sampling convex polytopes.
- A Docker image based on an Ubuntu distribution with the **PHAR** coded in C ready to use in any computer capable of running OpenBLAS and/or CUBLAS binaries. This instance of the algorithm allows the user to input the number of GPUS present (0 to 8) and the linear restrictions as a text file. See the online supplement for more information.
- An experimental analysis for approximating continuous functions via Bernstein polynomial given a set of linear restrictions that identify the search space. In particular, an experimental comparison between the maximum entropy approach and the **PHAR** for characterizing multiple attribute utility functions.

The remaining sections of the paper are organized as follows. Section 2 formalizes the notion of polytope and the **HAR** algorithm. Section 3 is dedicated to the formulation of **PHAR** and how it compares to the **HAR** algorithm presented in [10]. Section 4 focuses on reproducing the metric based experiments presented in [6] to confirm the validity of **PHAR**. Section 5 compares the performance of **PHAR** against **RHAT**. Section 6 explores the new methodology for approximating continues functions using Bernstein Polynomial, focusing on multi attribute utility functions and compares it with maximum entropy. Section 7 gives a brief description of the adaptable **PHAR** presented in the Docker image. Section 8 establishes future work directions, like an hybrid **PHAR** with accept and rejection methods for sampling convex hyperplanes. Section 8 gives some concluding remarks.

II. Problem Statement

A polytope is a geometric object with flat sides. It can be defined with a series of linear restrictions in a $\mathbb{R}^n$ space, the formal definition is:

$$\Delta^I = \{x \in \mathbb{R}^n \mid A^I x \leq b^I\}$$
$$\Delta^E = \{x \in \mathbb{R}^n \mid A^E x = b^E\}$$
$$\Delta = \Delta^I \cap \Delta^D$$

Where A^I, A^E are matrices in $\mathbb{R}^{i \times n}$ and $\mathbb{R}^{e \times n}$ respectively, and b^I, b^E are vectors in $\mathbb{R}^{i \times 1}$ and $\mathbb{R}^{e \times 1}$.

The first two sets are defined by a series of linear inequalities and equalities, respectively. The third one is the intersection of the two previous sets and forms the polytope we will study. Note that all three sets are convex by construction, since they are defined by linear inequalities, but the non-emptiness and bounded assumptions are not implied by definition. To ilustrate how a polytope is graphically represented in $\mathbb{R}^3$,in Figure II we show the 2-simplex.

The **HAR** algorithm proposed in [10] to obtain a collection X of T inner points of the polytope defined by A^I, b^i, A^E, b^e can be described as follows:

```
$i \leftarrow 0$
$X = \emptyset$
**Find** an strictly inner point of the polytope and label it $x_i$.
**while** $i \leq T$ **do**
    **Generate** a random uniform i.i.d. vector $d_i \in \mathbb{R}^n$ that represents the direction to move.
```

```
        **Find** the line set $L := \{x | x = x_i + \lambda d_i, x \in \Delta \ \& \ \lambda \in \mathbb{R}\}$.
        **Generate** a random point uniformly distributed in $L$ and label it $x_{i+1}$.
        $X = X \cup x_{i+1}$
        $i \leftarrow i + 1$.
    **end while**
    **return** $X$
```

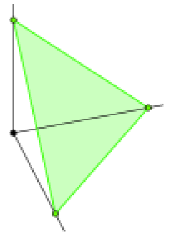

Fig. 1. Simplex in $\mathbb{R}^3$
[19]

To use the thinning factor φ proposed in [6] it suffices to store x_i every $(n-1)^3$ iterations or every 25 iterations if the autocorrelation factor is preferred[6]. This factors will be further analyzed in Section 4.

III. Extra

——————————————- IGNORE ANYTHING BELOW THIS —————————— —————————— The second one compares the **PHAR** against maximum entropy and copula methods commonly used in financial literaturefinancenetropy

.

The first one is based on the environments presented in which compares the (**HAR**) against other 6 approximation methods, including maximum entropy. The second scenario i

This paper presents four metrics to asses the uniformity of the sample produced by the **PHAR** algorithm which are also used in [6]:

1) Friedman-Rafsky two-sample Minimal Spanning Tree (**MST**) test which evaluates the obtained sample against the target distribution.
2) Coefficient Variation

Coefficient Variation, Component-wise error (SCE), y Sample Autocorrelation.

Consider a set of linear constraints that represent a n-dimensional hyperplane. At the beginning, the algorithm finds an inner point of the polytope x0. x0 is set as xt-1. Then it draws a direction d from the standarized multivariate normal distribution. Proceeds move from the previous point in direction d and -d until it violates some linear constraint, by doing this the algorithm can calculate an straight line, that spawns form x0 and moves in direction d and -d, that is entirely contained in the polytope. A random inner point from such line is uniformly drawn, stored and relabeled as xt-1. This will allow the algorithm to draw a new direction and a new inner point by repeating the process. The parallelized version of the algorithm represents the linear constraints as matrices, equalities and inequalities, and makes use of the level 1, 2 and 3 operations along with the OLS solver from CUBLAS and MKL libraries. One of the main features of the algorithm is the computation of a numerically stable (** check method later**) projection matrix at the begging of the algorithm which is stored in the GPGPU memory to reduce communication latency between host and device.

References

[1] Hastings, W. K. 1970. ”Monte Carlo Sampling Methods Using Markov Chains And Their Applications”. Biometrika 57 (1): 97. doi:10.2307/2334940

[2] Clarkson, Kenneth L., and Peter W. Shor. 1989. ”Applications Of Random Sampling In Computational Geometry, II”. Discrete Computational Geometry 4 (5): 387-421. doi:10.1007/bf02187740.

[3] T. Bengtsson, P. Bickel, and B. Li, “Curse-of-dimensionality revisited: Collapse of the particle filter in very large scale systems,” Institute of Mathematical Statistics Collections Probability and Statistics: Essays in Honor of David A. Freedman, pp. 316–334, 2008.

[4] D. Crisan and A. Doucet, “A survey of convergence results on particle filtering methods for practitioners,” IEEE Transactions on Signal Processing, vol. 50, no. 3, pp. 736–746, 2002.

[5] https://arxiv.org/abs/1505.00579

[6] Hit-And-Run enables efficient weight generation for simulation-based multiple criteria decision analysis.

[7] A survey on stochastic multicriteria acceptability analysis methods

[8] Approximations, Simulation, and Accuracy of Multivariate Discrete Probability Distributions in Decision Analysis

[9] Exploring the Accuracy of Joint-Distribution approximations given partial information

[10] Generating a Random Collection of Discrete Joint Probability Distributions Subject to Partial Information

[11] Maximum Entropy Distributions between Upper and Lower Bounds

[12] Multiattribute Utility Copulas Abbas

[13] Maximum entropy modeling of species geographic distributions

[14] https://pubsonline.informs.org/doi/abs/10.1287/opre.1080.0600

[15] NEGOCIACIONES DE MÁXIMA PROBABILIDAD PARA JUEGOS COOPERATIVOS ENTRE N-ACTORES

[16] Copula methods in finance

[17] https://www.researchgate.net/profile/Luis$_Montiel3/publication/275937754_{AG}ener$ $Generalized - Sampling - Approach - for - Multilinear - Utility - Functions - Given - Partial - Preference - Information.pdfhttps : //arxiv.org/pdf/1710.05053.pdf$

[18] https://www.mdpi.com/1099-4300/15/11/4909/htm

[19] https://en.wikipedia.org/wiki/Simplex/media/File:2D-simplex.svg