# *optGpSampler*: An Improved Tool for Uniformly Sampling the Solution-Space of Genome-Scale Metabolic Networks

**Wout Megchelenbrink[1,2,3]*, Martijn Huynen[2,3], Elena Marchiori[1,3]***

**1** Institute for Computing and Information Sciences (ICIS), Radboud University Nijmegen, Nijmegen, The Netherlands, **2** Centre for Molecular and Biomolecular Informatics (CMBI), Radboud University Nijmegen Medical Centre, Nijmegen, The Netherlands, **3** Centre for Systems Biology and Bioenergetics (CSBB), Radboud University Nijmegen Medical Centre, Nijmegen, The Netherlands

## Abstract

Constraint-based models of metabolic networks are typically underdetermined, because they contain more reactions than metabolites. Therefore the solutions to this system do not consist of unique flux rates for each reaction, but rather a space of possible flux rates. By uniformly sampling this space, an estimated probability distribution for each reaction's flux in the network can be obtained. However, sampling a high dimensional network is time-consuming. Furthermore, the constraints imposed on the network give rise to an irregularly shaped solution space. Therefore more tailored, efficient sampling methods are needed. We propose an efficient sampling algorithm (called *optGpSampler*), which implements the Artificial Centering Hit-and-Run algorithm in a different manner than the sampling algorithm implemented in the COBRA Toolbox for metabolic network analysis, here called *gpSampler*. Results of extensive experiments on different genome-scale metabolic networks show that *optGpSampler* is up to 40 times faster than *gpSampler*. Application of existing convergence diagnostics on small network reconstructions indicate that *optGpSampler* converges roughly ten times faster than *gpSampler* towards similar sampling distributions. For networks of higher dimension (i.e. containing more than 500 reactions), we observed significantly better convergence of *optGpSampler* and a large deviation between the samples generated by the two algorithms. **Availability:** *optGpSampler* for Matlab and Python is available for non-commercial use at: http://cs.ru.nl/~wmegchel/optGpSampler/.

**Citation:** Megchelenbrink W, Huynen M, Marchiori E (2014) *optGpSampler*: An Improved Tool for Uniformly Sampling the Solution-Space of Genome-Scale Metabolic Networks. PLoS ONE 9(2): e86587. doi:10.1371/journal.pone.0086587

**Editor:** Simon Rogers, University of Glasgow, United Kingdom

**Received** June 7, 2013; **Accepted** December 13, 2013; **Published** February 14, 2014

**Funding:** This work was supported by The Netherlands Organisation for Scientific Research (NWO) and The Netherlands Organisation for Health Research and Development (ZonMW). The funders had no role in study design, data collection and analysis, decision to publish, or preparation of the manuscript.

**Competing Interests:** The authors have declared that no competing interests exist.

* E-mail: w.megchelenbrink@cs.ru.nl (WM); elenam@cs.ru.nl (EM)

## Introduction

Modelling metabolic networks helps to unravel the complex machinery of metabolism within the cell. A classic approach is to model the reaction pathways in a dynamic fashion, using detailed kinetic data. For genome-scale models, often involving hundreds or thousands of reactions and metabolites, it is experimentally prohibitive to obtain the kinetic parameters involved. A constraint-based approach has successfully been applied to model and address a wide range of biological questions in the absence of detailed kinetic data [1,2]. By using a steady-state assumption, a first type of constraint dictates that all metabolite concentrations stay constant over time (mass-balance). A second type of constraint limits the flux rate for each reaction (flux-capacity and directionality). The relation between the $m$ metabolites and $n$ reactions is described in the $m \times n$ stoichiometric matrix $\mathbf{S}$. A positive stoichiometric coefficient $\mathbf{S}_{i,j}$ means that the metabolite $i$ is produced by reaction $j$ and a negative entry indicates that the metabolite is consumed in that reaction. At steady-state, the mass-balance and flux capacity constraints can be formulated as in eq. (1) and inequality (2) respectively.

$$\frac{d\vec{x}}{dt} = \mathbf{S}\vec{v} = 0, \tag{1}$$

$$v_{j,min} \leq v_j \leq v_{j,max}, \forall j \in \{1 \ldots n\}, \tag{2}$$

where $\vec{x}$ and $\vec{v}$ are vectors of metabolite concentrations and flux rates respectively. Each flux rate $v_j$ is bounded by inequality constraint (2). Although in some cases the bounds are known from experiments, for most reactions this is not the case and arbitrarily large values are used. Since the matrix $\mathbf{S}$ is a system of linear equations, the constraints in eq:mass-balance and inequality (2) form a bounded convex space [3], containing all possible values of $\vec{v}$. A major challenge is to characterize the biologically interesting flux distributions among all alternatives. Since the stoichiometric matrix is fixed, the remaining possibility is to add additional constraints or tighten the inequality bounds in (2). This can be done by incorporating measured flux data, which is often laborious, expensive or difficult to obtain, even for a small subset of all reactions.

Many constraint-based methods have been proposed to find flux distributions that are of biological interest. A successful approach is Flux Balance Analysis (FBA) [4], that introduces a biologically relevant objective function and uses linear programming to find a flux distribution that optimizes this objective. FBA proved to be especially useful for cell types with a well-defined objective function, such as maximum growth for unicellular species. Although FBA has successfully been applied to determine possible phenotypes and byproduct secretion in various experimental settings, it is often unable to determine the underlying (internal) flux states [5]. Furthermore, for many objective functions, a wide range of alternative optima exists [6].

In order to obtain an estimated probability distribution of attainable flux values for each reaction in the network, methods based on uniform sampling are used. A fast algorithm for this task is called *hit-and-run* (HR) [7]. HR collects samples by iteratively choosing a random direction and a random step size in that direction such that the next point also resides in the solution space. For the irregularly shaped solution spaces of metabolic networks the Artificial Centering Hit-and-Run (ACHR) algorithm [8] is better suited, because it is tailored to sample in the elongated directions of the solution space. Partly based on ACHR, the uniform random sampling procedure known as *gpSampler* is often used to sample metabolic networks and implemented in the COnstrained Based Reconstruction and Analysis (COBRA) Toolbox [9]. Although these algorithms are often referred to as uniform random samplers, their convergence behaviour in the context of genome-scale metabolic networks has not yet been thoroughly investigated. Besides its irregular shape, the solution space of genome-scale metabolic networks are often high-dimensional, containing hundreds to almost a thousand dimensions as in the human metabolic network reconstruction.

Therefore in this paper we investigate uniform random sampling in the context of metabolic networks. Our contributions are threefold: (1) we introduce an efficient and effective random sampling algorithm which combines the advantages of ACHR and *gpSampler*; (2) we propose a new measure to quantify the deviation between samples obtained from two independent sampling runs; (3) we perform a thorough analysis on five metabolic network models.

## Methods

### Uniform Random Sampling

One of the first attempts to sample the flux states in a metabolic network used a *rejection sampling* technique [10]. In rejection sampling, the space of interest is enclosed by a regular shape, such as a parallelepiped. Samples are drawn from the uniform distribution over the parallelepiped and rejected if they violate the constraints for the enclosed space of interest. The nice feature of rejection sampling is that the samples are uniformly distributed over the enclosed space. However, fitting a regular shape tightly around the space of interest is hard and often impossible. This means that in higher dimensions, the volume of the enclosing shape grows explosively compared to the volume of the shape of interest [7]. In this case, a very large fraction of the samples have to be rejected, making rejection sampling an inefficient method for genome-scale models.

The *hit-and-run* (HR) algorithm [7] mitigates this problem, because it samples directly from the solution space (see Fig. 1). Hit-and-run starts from a point $\vec{x}_0$ in the bounded space. It chooses an arbitrary direction $\vec{u}_1$ from the uniform distribution on the boundary $\partial B$ of the unit sphere in $\mathbb{R}^n$. The distance from $\vec{x}_0$ to the boundary of the solution space in the direction of $\vec{u}_1$ determines the maximum distance it can travel. An arbitrary step size $\lambda_1$ is selected in the (negative) direction of $\vec{u}_1$, such that the sampler does not step out of the constrained space. The next point $\vec{x}_2$ is determined by travelling distance $\lambda_1$ in the direction $\vec{u}_1$. By iterating this process, HR generates a chain of consecutive sample points. The fact that HR uses only the current point to obtain the next sample makes it a Markov Chain Monte Carlo (MCMC) method [11], which has been shown to converge towards the target distribution (uniform in our setting).

In general, the constraints in metabolic network models lead to a convex space of irregular shape which is elongated for reactions whose flux rates are loosely constrained, and is narrow for tightly constrained fluxes. A consequence of this phenomenon is that many sample points are close to the boundary of the solution space. Since HR chooses a direction $\vec{u}_i$ uniformly from all possible directions, this enforces the sampler to perform small steps, so the next generated point is close to the previous one. In practice this prevents the sampler to fully explore the rest of the solution space.

ACHR [8] alleviates the problem of getting trapped in regions close to the boundary because it tries to sample in the elongated directions, thus making larger steps possible. It iteratively generates samples by using an 'artificial' centre of the space, which is empirically estimated at each iteration using the points sampled so far. ACHR consists of two phases: warm-up and (main) sampling. In the *warm-up* phase an arbitrary initial point $\vec{x}_0$ in the solution space is selected to generate a chain $\vec{x}_0,\ldots,\vec{x}_W$ of $W \geq n$ points. The requirement $W \geq n$ ensures that after the warm-up phase, the set of directions spans $\partial B$ with probability one [8]. Then the main *sampling* phase starts from $\vec{x}_W$. By iteratively updating the empirical centre and by using a direction from a randomly chosen previous sampled point to this centre, ACHR

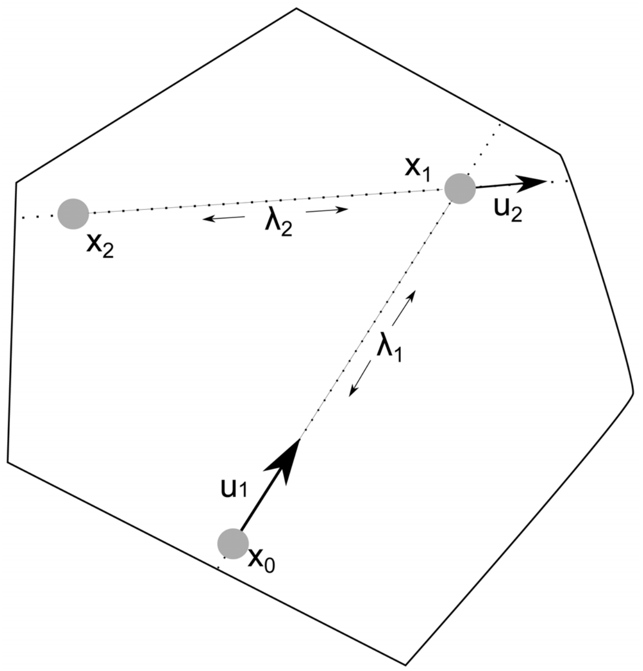

**Figure 1. Illustration of hit-and-run.** Hit-and-run starts at the point $x_0$ in the solution space $\mathbf{S}$. It chooses a random direction $u_1$ and determines the maximum distance it can travel forwards or backwards in that direction. A random step size $\lambda_1$ is chosen on the line $u_1 \in \mathbf{S}$. The next point $x_1$ is obtained by travelling $\lambda_1$ in the direction $u_1$. By iterating this process $T$ times, samples are obtained that are uniformly distributed in the space, when $T \rightarrow \infty$.
doi:10.1371/journal.pone.0086587.g001

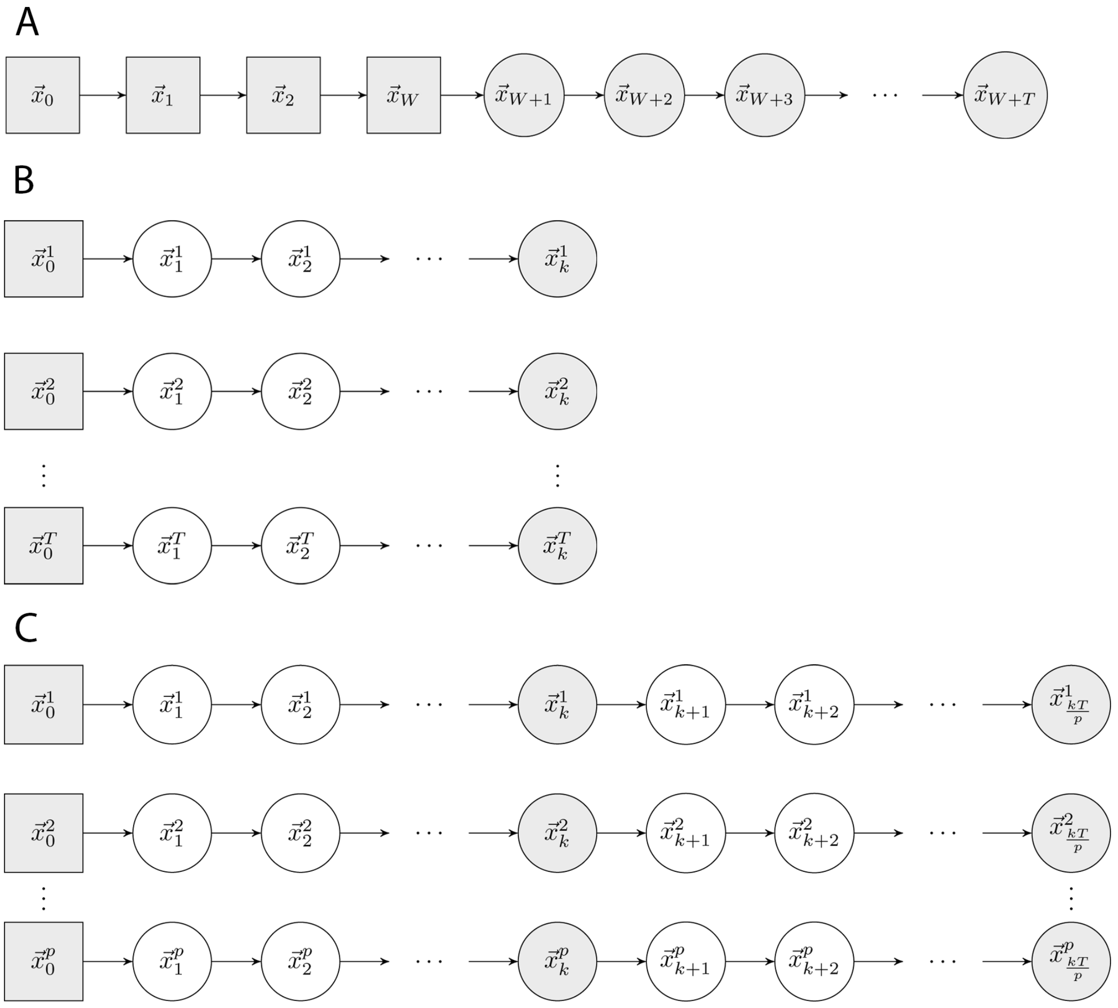

**Figure 2. Conceptual difference between the samplers.** Conceptual difference between (a) ACHR, (b) gpSampler and (c) optGpSampler. Warm-up points are depicted as gray rectangles, samples that are stored as gray circles. Uncoloured circles denote points that are visited by the sampler, but are not stored as a sample. a) The original ACHR algorithm starts at a point $\vec{x}_0$ and iteratively moves to a next point $\vec{x}_i = \vec{x}_{i-1} + \lambda_i \vec{u}_i$. One chain is used, with step count $k=1$. The chain contains $W$ warm-up points and $T$ samples. b) *GpSampler* uses the linear programming procedure described in the main text to find $T$ warm-up points. Then, each of the warm-up points is iteratively moved in the space in the same fashion as the ACHR algorithm in (a), leading to $T$ sampling chains. Each chain of length $k$ returns its end point as a sample. c) *OptGpSampler* obtains $2n$ warm-up points. For each of the $p$ processors used, a warm-up point is chosen randomly as the initial point $\vec{x}_0{}^j$, with $j \in \{1,2,\ldots,p\}$. Starting from the warm-up points, new points are found in the same fashion as for the ACHR algorithm in (a), but now only every $k^{th}$ point is kept as a sample. Again, the result is $T$ sample points, but now these have travelled $k$ up to $kT/p$ steps from a warm-up point. Compared to *gpSampler*, it uses less but much longer sampling chains.
doi:10.1371/journal.pone.0086587.g002

explores elongated directions of the space. Figure 2a illustrates the *warm-up* and the main *sampling* phase of ACHR.

The sequence of ACHR iterates is not a Markov chain, due to the dependence of directions on prior iterates. Thus the sequence of iterates is not guaranteed to converge to a uniform distribution [8].

ACHR is at the core of *gpSampler* [9], a popular sampling algorithm for metabolic network analysis. Figure 2b illustrates how *gpSampler* works. The highly irregular shape of the solution space of metabolic networks makes a uniform direction choice on $\partial B$ a poor choice. Therefore, in order to generate a number of $T$ samples, *gpSampler*'s warm-up phase uses linear programming in a two parts procedure. In part (1) $2n$ warm-up points are generated by consecutively minimizing and maximizing the flux rate of each reaction. In part (2) the remaining $T-2n$ warm-up points are generated by assigning random weights to the fluxes that should be optimized. The optimal solutions (and thus the warm-up points) for a linear program reside on the boundary of the constrained space [12]. Often, this causes the allowed step sizes to become very small, which makes it hard to move away from the boundary. Therefore, *gpSampler* uses a linear transformation to 'pull' the warm-up points more into the interior of the solution space. Then, each moved warm-up point is used in the main sampling phase to generate $T$ separate chains of length $k$. The user provided *step count* parameter $k$ determines the number of ACHR iterates between the starting point $\vec{x}_W$ and the end point $\vec{x}_{W+k}$ of each chain. Finally, the $T$ end points of the chains are returned as samples.

## optGpSampler

We propose to combine a part of the warm-up phase of *gpSampler* and ACHR. The resulting algorithm is called *optGpSampler* (see Table 1. Algorithm 1). First, $2n$ warm-up points are generated using part (1) of *gpSampler*'s warm-up procedure by successively minimizing and maximizing the flux through each reaction. We do not generate $T$ warm-up points as in *gpSampler* because running a linear program on a large network is much more time-consuming than random sampling. Moreover, although the weight vector for the linear program is randomly chosen, the constraints in eq. (1) and inequality (2) often lead to the same or a similar optimal solution. This could bias the starting points and directions choice of our sampler.

The sampling phase of *optGpSampler* is similar to that of ACHR (see Fig. 2c), but only selects a sample at each $k$ iterates. Furthermore, instead of generating one chain of consecutive sample points, *optGpSampler* exploits $p$ processors and generates $p$ chains in parallel. In practice, the desired number of sample points $T$ is much larger than the number of available processors $p$. This makes the length of the chains generated by *optGpSampler* a factor of $T/p$ longer than those generated by *gpSampler*.

We implemented *optGpSampler* in C++ and used Armadillo [13], a fast linear algebra library. OpenMp [14] was utilized to start a separate chain on $p$ processor cores in parallel. Interfaces for Matlab and Python enable users to easily sample existing models with *optGpSampler* or integrate our sampler in new methods.

## Experiments

### Datasets

We benchmarked *gpSampler* and *optGpSampler* on five publicly available reconstructions of genome-scale metabolic networks [14]. All reactions and associated metabolites that could not carry a flux were removed from the models prior to the sampling. The network size remaining after this preprocessing step, i.e. the number of $m$ metabolites and $n$ reactions is given between brackets. In ascending size order, we used E. coli central metabolism (68, 87), C. thermocellum iSR432 (288, 351), S. cerevesiae iND750 (479, 631), E. coli iAF1260 (1032, 1532) and H. sapiens recon 1 (1587, 2469).

### Evaluation

We assessed efficiency and quality performance of *gpSampler* and *optGpSampler*. Efficiency was measured using *runtime*. Quality performance in our context amounts to measure the capability of a sampler to effectively generate points uniformly distributed in the solution space. Since both *gpSampler* and *optGpSampler* have no theoretical convergence guarantees, we use two methods: empirical convergence diagnostics and a new method called xy-deviation.

**Table 2.** Runtimes for the networks analysed.

| Network name | m | n | N(S) | Time *Gp* (SD) | Time *optGp* (SD) |
|---|---|---|---|---|---|
| E. coli core | 68 | 87 | 24 | 137.16 (0.62) | 3.63 (0.05) |
| C. therm. iSR432 | 288 | 351 | 70 | 258.57 (0.54) | 10.20 (0.05) |
| S. cerev. iND750 | 479 | 631 | 180 | 496.57 (3.07) | 21.81 (0.04) |
| E. coli iAF1260 | 1032 | 1532 | 525 | 1474.01 (6.78) | 95.78 (2.62) |
| H. sapiens recon 1 | 1587 | 2469 | 932 | 2910.26 (43.57) | 349.05 (0.48) |

The number of metabolites and reactions is denoted by *m* and *n* respectively. The dimensionality of the nullspace of *S*, is given by *N(S)*. *Time gp (SD)* is the mean runtime (seconds) and standard deviation for sampling $T = 50.000$ points using *gpSampler*. *Time optGp (SD)* denotes the same figures for *optGpSampler*. Experiments were performed on a 16 GB RAM AMD Phenom desktop pc.
doi:10.1371/journal.pone.0086587.t002

### Empirical Convergence Diagnostics

Empirical convergence diagnostics are used to test whether the sampled distribution converges towards a stationary one. There is not always good agreement between different convergence diagnostic methods [16]. Therefore we use the following three convergence diagnostics tests: Gelman and Rubin [17], Geweke [18] and Heidelberger and Welch (HW) [19]. They are available in the Convergence Diagnosis and Output Analysis (CODA) Toolbox for MCMC [20].

The Gelman and Rubin test is multivariate: it returns a so called $R$ value (with $R \geq 1.0$). $R$ values smaller than 1.2 indicate convergence, with values closer to 1.0 indicating better results. The other two tests are univariate. The Geweke test returns a z-value for each reaction, with lower values indicating better convergence. The HW test returns whether a sample distribution for a given reaction converged (value = 1) or not (value = 0).

**Table 1.** Algorithm 1.

```
  Input: S: the solution space; $T$: sample count; $k$: step count, $p$: number of processors;
  output: P: sequence of $T$ sampled points;
  /* Warm-up phase                                                          */
1 Generate $2n$ warm-up points as in part (1) of gpSampler's warm-up phase;
  /* Sampling-phase                                                         */
2 $\mathbf{P} = <>$;
3 $L = \lceil Tk/p \rceil$ ;
4 for $i = 1$ to $p$ do
5     $\vec{x}_0 =$ a point randomly chosen from the $2n$ warm-up points;
6     for $j = 1$ to $L$ do
7         $\vec{x}_j =$ point generated from $\vec{x}_{j-1}$ by performing one iterate of the ACHR sampling phase;
8         if $j$ mod $k = = 0$ then
9             $\mathbf{P} = <\mathbf{P}, \vec{x}_j>$;
10        end
11    end
12 end
```

optGpSampler(**S**, $T$, $k$, $p$).
doi:10.1371/journal.pone.0086587.t001

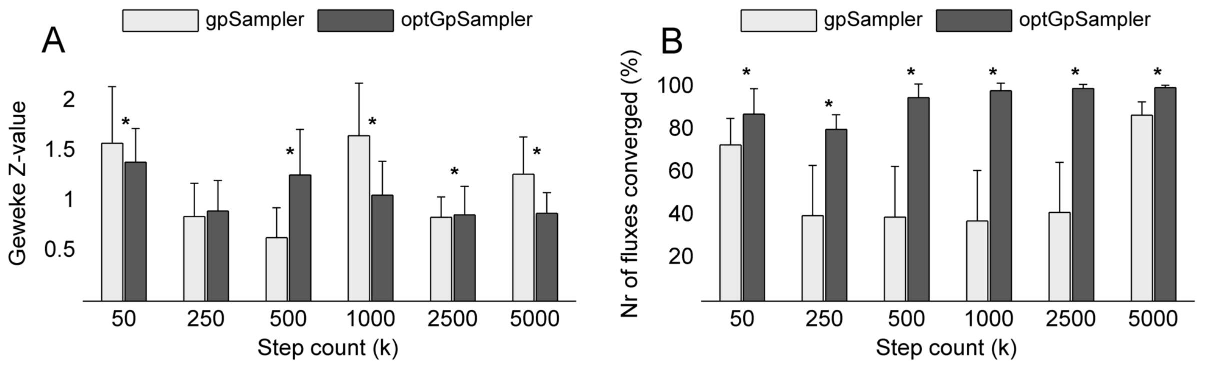

**Figure 3. Empirical convergence for C. thermocellum iSR432.** (a) Convergence according to the Geweke diagnostic. (b) Convergence according to the Heidelberger-Welch diagnostic. Convergence for *optGpSampler* is observed at approximately 500 steps. For *gpSampler*, both diagnostics only agree on convergence after $k = 5000$ steps. Notice the higher HW convergence fraction of the latter at $k = 50$ steps and at $k = 5000$ steps compared to the steps in between.
doi:10.1371/journal.pone.0086587.g003

## xy-Deviation

In general, a larger step count gives a sample distribution that is closer to the target distribution, at the expense of a longer runtime. Therefore, we introduce a measure called $xy$-deviation that quantifies how samples generated by sampler $x$ using a given step count $k_1$ deviate from those generated by sampler $y$ using a *much larger* step count $k_2$. Specifically, a small deviation indicates that the sample distribution generated by sampler $x$ converged empirically to the target distribution of $y$.

Given samplers $x$ and $y$, step counts $k_1$ and $k_2$, with $k_2$ very large ($k_2 = 5000$ in our experiments), the number $c$ of runs, and the number $T$ of samples, $xy$-deviation is computed as follows.

1. Perform $c$ runs of sampler $x$ with step count $k_1$, and $c$ runs of sampler $y$ with step count $k_2$, where each run generates $T$ samples.
2. Sort the points in each run (e.g., in increasing order), producing $c$ sorted chains of points for sampler $x$ and $y$: $<x^j_{b,i}>$, $<y^j_{b,i}>$, $b \in \{1,\ldots,T\}$, $j \in \{1,\ldots,n\}$, $i \in \{1,\ldots,c\}$.
3. For each reaction $j$, normalize the flux rates: we divide $x^j_{b,i}$ (resp. $y^j_{b,i}$) by the difference between the upper and lower bounds of $v_j$:
$$X^j_{b,i} = \frac{x^j_{b,i}}{v_{j,max} - v_{j,min}}, \qquad Y^j_{b,i} = \frac{y^j_{b,i}}{v_{j,max} - v_{j,min}},$$
$i \in \{1,\ldots,c\}, j \in \{1,\ldots,n\}, b \in \{1,\ldots,T\}$.
4. Compute the mean chain of the $c$ runs of $y$:
$$\bar{Y}^j_b = \frac{1}{c}\sum\nolimits_{i=1}^{c} Y^j_{b,i}, \; j \in \{1,\ldots,n\}, b \in \{1,\ldots,T\}.$$
5. Our assumption is that chains generated by sampler $x$ that converge will have small deviation from the ‘average-chain’ generated by $y$. For each reaction $j$, we measure such deviation by computing the mean absolute deviation over the chains generated by $x$ from $\bar{Y}^j_b$: $D^j = \frac{1}{c \times T}\sum\nolimits_{i=1}^{c}\sum\nolimits_{b=1}^{T} |X^j_{b,i} - \bar{Y}^j_b|$, $j \in \{1 \ldots n\}$.

Our choice to use a mean deviation over the standard deviation is motivated by the fact that the mean deviation is more efficient than the standard deviation in the realistic situation where some of the measurements are in error, and more efficient for distributions other than perfect normal [21]. Note that the range of $D_j$ is [0,1], and can be expressed as a percentage, which makes it convenient to compare deviations across different reactions.

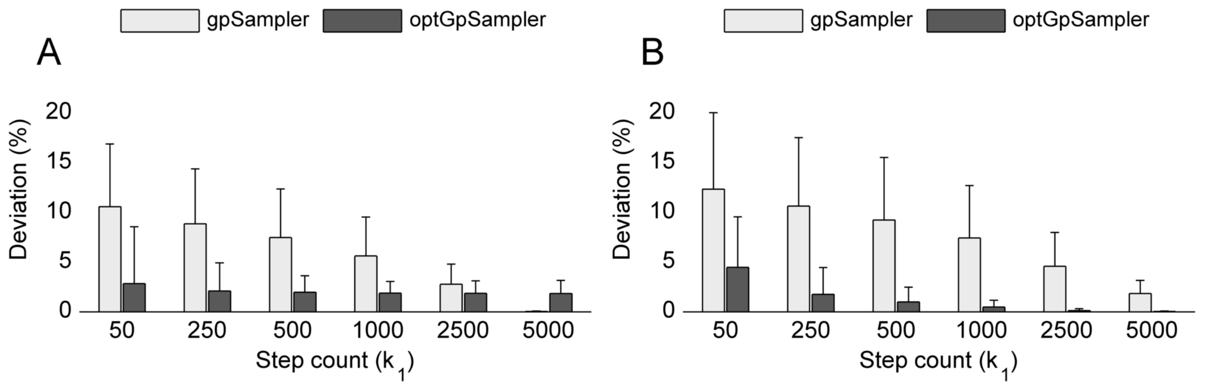

**Figure 4.** ***xy*-deviation for C. thermocellum iSR432.** $xy$-deviation from samples obtained with sampler $x$ at step count $k_1$ to sampler $y$ using $k_2 = 5000$. (a) Deviation to samples obtained by $y = gpSampler$. (b) Deviation from samples obtained by $y = optGpSampler$. In both cases *optGpSampler* converges much faster to sampler $y$.
doi:10.1371/journal.pone.0086587.g004

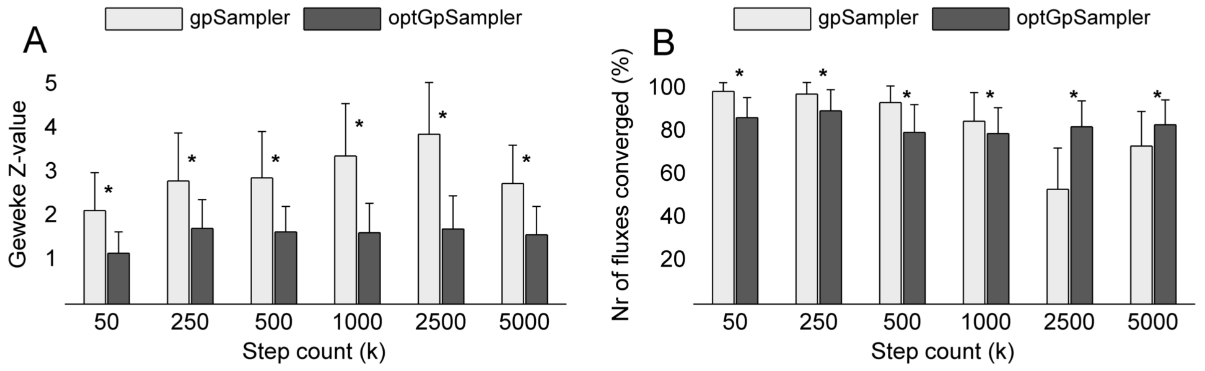

**Figure 5. Empirical convergence for E. coli iAF1260.** (a) Convergence according to the Geweke diagnostic. (b) Convergence according to the Heidelberger-Welch diagnostic. For both convergence tests, *gpSampler*'s performance deteriorates when the step count is increased up to $k = 2500$. Especially the good scores at low values of $k$ seem unrealistic and could indicate convergence towards a non-uniform distribution. Results for *optGpSampler* seem more stable and more reliable.
doi:10.1371/journal.pone.0086587.g005

6. Then the *xy-deviation* over $c$ chains with respect to $k_1$ and $k_2$ is defined as the average of the reaction deviations:

$$\overline{D} = \frac{1}{n}\sum_{j=1}^{n} D^j \tag{3}$$

Small values of *xy-deviation* indicate convergence of $x$ to $y$. In the experiments we analysed self-deviation ($xx$-deviation) and cross-deviation ($xy$-deviation, with $x$ and $y$ different samplers) for different values of $k_1$.

## Results

We used the *gpSampler* implementation in the COBRA Toolbox. Results of extensive experiments are given in supplementary material. Tables S1–S5 in File S1 provide results for all the runtime experiments performed. Figures S1–S6 in File S1 provide the results of the convergence diagnostic tests and $xy$-deviation. We visualized how a small or large $xy$-deviation translates to a similar or (highly) dissimilar sample distribution in figures S7–S11 in File S1.

### Efficiency

We sampled all networks four times using $T = 10$, 50 and 100 thousand samples, with step count $k = 50$. Both *gpSampler* and *optGpSampler* were executed in parallel mode on an AMD desktop computer using $p = 4$ processor cores in parallel. Efficiency results, summarized in Table 2, show that *optGpSampler* is roughly 6 to 40 times faster than *gpSampler*.

### Quality

To asses the quality of the results, we performed four independent runs with each sampler. Each run collected $T = 50.000$ samples and was repeated for six different step counts ($k \in \{50,250,500,1000,2500,5000\}$).

We ran all convergence diagnostics with the default settings in the CODA package. The convergence tests results were averaged over the four runs. A univariate test for a sampler outputs a vector of length $n$. Significance of the difference between *gpSampler* and *optGpSampler* was assessed by applying the Wilcoxon signed-rank test to the corresponding vectors. The results for the Gelman and Rubin test indicated convergence for all experiments, in disagreement with results of the Geweke and HW tests.

The obtained chains were also used to compute the $xy$-deviation for the above given step count $k_1$, and $k_2 = 5000$. All

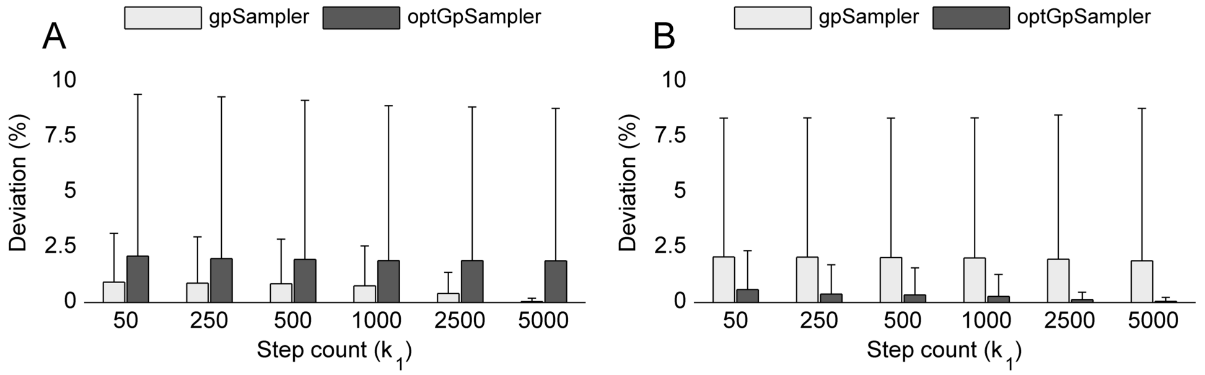

**Figure 6.** $xy$**-deviation for E. coli iAF1260** $xy$**-deviation from samples obtained with sampler** $x$ **at step count** $k_1$ **to sampler** $y$ **using** $k_2 = 5000$**.** (a) Deviation to samples obtained by $y = gpSampler$. (b) Deviation from samples obtained by $y = optGpSampler$. Self-deviation ($x = y$) is small for both samplers, but there is a large cross-deviation ($x \neq y$). For this large network, the we do not observe convergence of *gpSampler* to the samples obtained by *optGpSampler* or vice versa as in Fig. 4.
doi:10.1371/journal.pone.0086587.g006

sample chains were compared by the average of the four chains obtained at the highest step count of $k = 5000$.

**Small networks (less than** 500 **reactions).** On the *E. coli central metabolism* network, the Geweke test returned relatively low z-values for both *gpSampler* and *optGpSampler* (see figure S1 **in** File S1). Results of the HW test indicated that *optGpSampler* converged rapidly, at $k = 50$ step count. Both the Geweke and HW tests showed that *gpSampler* needs a step count close to 500 to converge. Results of xy-self-deviation demonstrate a large deviation of samples generated by *gpSampler* with a small step count from those generated with a higher step count. In general, results showed that *optGpSampler* with a low step count generates samples that are both close to those generated using a higher step count with the same sampler and to those generated by *gpSampler* with a higher step count.

On the *C. thermocellum iSR432* network, the HW test showed significantly higher convergence rates for *optGpSampler* (see Fig. 3). Both the Geweke and HW tests showed that *optGpSampler* converged at step counts bigger than 1000. The situation for *gpSampler* is different: the values of the HW test dropped at step count 250 and then increased significantly when the step count reached 5000. This indicates two distinct points of convergence, one at small step count values and one at large values. This behaviour can be explained by the way *gpSampler* collects its samples. It starts at a warm-up point and uses $k$ iterates of the ACHR algorithm to obtain a sample point. It repeats this process, each time starting from a different warm-up point (see Fig. 2b). These warm-up points turn out to be close to each other, as a consequence of the linear programming procedure used. Thus *gpSampler* often starts the sampling from the same area of the solutions space. In higher dimensions, a small step count can prevent *gpSampler* to 'travel' far from the warm-up points. In this case the small step count causes *gpSampler* to show a bias towards the regions close to the warm-up points and the sampling chain converges towards these regions. The large $xy$-deviation (see Figure 4) shows that the samples collected are different from those sampled at a higher step count.

The results for *gpSampler* at large step count ($k = 5000$ steps) again indicates convergence. In this case the small $xy$-deviation indicates that an extended region is covered by the samples. Since *optGpSampler* does not restart at a warm-up point, it is better able to 'escape' from the regions near the warm-up points because it effectively uses much longer chains. Therefore, its convergence and $xy$-deviation results are better.

**Large networks (at least** 500 **reactions).** The convergence and results for the *S. cerevisiae iND750* (see figure S3 **in** File S1) and especially the *E. coli iAF1260* network in Fig. 5 showed an even more surprising result. For these larger models, the convergence results for *gpSampler* declines when the step count is increased. For *optGpSampler*, we still observed an increasing convergence performance for the yeast network, although a much larger step count ($k \sim 2500$) was required. The more stable results for the larger *E. coli iAF1260* network could be an effect of the minimum glucose setting of this network, which significantly reduces the attainable flux states. The $xy$-deviation results in Fig. 6 indicate that the samplers give completely different sampling results. The self-deviation shown in Fig. 6, reveals a small variability within the four independent runs for each sampler. This means that both samplers give relatively stable results, and thus that the deviation results observed between the samplers must be due to the difference between the algorithms.

Finally, the results for the human network reconstruction (see figures S5 and S6 in File S1) indicate that *gpSampler* converges already at low $k$. Although we believe that the sample distributions indeed converged in this case, it seems unlikely that they represent a uniformly distributed sample. First, the high dimensionality of almost a thousand and the declining results for the Geweke test make this unlikely. Next, the huge deviation with samples obtained by *optGpSampler* indicate a non-uniform distribution, especially since we saw that *optGpSampler* performs better on the smaller networks. The convergence results for *optGpSampler* seem more realistic, with a relatively large z-value and a HW test result that indicates that around 60% of the sampled flux distributions converged.

## Discussion

We proposed a new algorithm for uniform sampling of the steady-state solution space of metabolic networks. Our algorithm also implements ACHR, but in a different manner than the state-of-the-art sampling method for metabolic networks (*gpSampler*). We compared the runtimes with those of *gpSampler*, and showed its superior efficiency. We investigated empirical convergence using different diagnostics and showed faster convergence of *optGpSampler* on the two smaller networks studied. Moreover, by using the here introduced $xy$-deviation measure, we compared the sampled distributions. For smaller networks, the samples obtained by *optGpSampler* using a small step count are close to those obtained with a high step count by both *optGpSampler* and *gpSampler*.

On three large networks the convergence performance of *gpSampler* diminishes when the step count increases. We hypothesized that the approach *gpSampler* takes by starting each sample chain at a warm-up point, together with the high dimensionality of the solution space, restrains its ability to move from the vicinity of these warm-up points. Because our method continues the ACHR procedure from the last collected point, it effectively uses much larger step counts. Therefore, *optGpSampler* is more likely to escape the regions near the warm-up points, leading to a better sampling result.

For the larger networks, results showed that the convergence observed at lower step counts does not reflect a convergence towards the target distribution, since the sample distributions deviate significantly from those generated using a large step count. Therefore, especially larger networks should be sampled with a high step count.

To the best of our knowledge there is no method to assess whether samples are truly uniformly random distributed in a convex space of unknown shape. Since the chains generated by ACHR are not Markov chains, asymptotic convergence guarantees also do not hold for both *gpSampler* and *optGpSampler*. Therefore convergence results should be interpreted with caution. The accelerated convergence of ACHR towards a uniform distribution was demonstrated by [8] for convex polytopes of known shape. However it remains uncertain to what extent the samples obtained by ACHR for the irregular solution space of metabolic networks are truly uniformly distributed. As expected, our experiments indicate that convergence results deteriorate when the dimensionality of the solution space increases, and that for the large genome-scale metabolic networks, using a large step count is advisable.

We envisage the provided implementation of *optGpSampler* will be beneficial to constraint-based metabolic network analysis, as it provides an efficient and versatile algorithm for sampling the irregular solution space of metabolic networks.

## Supporting Information

**File S1 Runtimes, convergence results and $xy$-deviation for the other metabolic models considered.**
(PDF)

## Acknowledgments

We thank Sergio Rossell and Robin van der Lee for testing our implementation and providing us with valuable feedback. We thank the reviewers for their comments.

## Author Contributions

Conceived and designed the experiments: WM EM. Performed the experiments: WM. Analyzed the data: WM EM. Wrote the paper: WM MH EM.

## References

1. Kauffman KJ, Prakash P, Edwards JS (2003) Advances in ux balance analysis. Current opinion in biotechnology 14: 491–496.
2. Raman K, Chandra N (2009) Flux balance analysis of biological systems: applications and challenges. Briefings in bioinformatics 10: 435–449.
3. Price ND, Reed JL, Papin JA, Wiback SJ, Palsson BO (2003) Network-based analysis of metabolic regulation in the human red blood cell. Journal of Theoretical Biology 225: 185–194.
4. Varma A, Palsson BO (1994) Metabolic ux balancing: Basic concepts, scientific and practical use. Bio/technology 12.
5. Suthers PF, Burgard AP, Dasika MS, Nowroozi F, Van Dien S, et al. (2007) Metabolic ux eluci- dation for large-scale models using 13c labeled isotopes. Metab Eng 9: 387–405.
6. Mahadevan R, Schilling C (2003) The effects of alternate optimal solutions in constraint-based genome-scale metabolic models. Metabolic engineering 5: 264–276.
7. Smith RL (1984) Efficient monte carlo procedures for generating points uniformly distributed over bounded regions. Operations Research 32: 1296–1308.
8. Kaufman DE, Smith RL (1998) Direction choice for accelerated convergence in hit-and-run sampling. Oper Res 46: 84–95.
9. Schellenberger J, Palsson BØ (2009) Use of randomized sampling for analysis of metabolic networks. J Biol Chem 284: 5457–5461.
10. Wiback SJ, Famili I, Greenberg HJ, Palsson BØ (2004) Monte carlo sampling can be used to determine the size and shape of the steady-state ux space. Journal of theoretical biology 228: 437–447.
11. Gilks WR, Richardson S, Spiegelhalter D (1995) Markov Chain Monte Carlo in Practice (Chapman & Hall/CRC Interdisciplinary Statistics). Chapman and Hall/CRC.
12. Winston WL, Goldberg JB (2004) Operations research: applications and algorithms. Thomson/Brooks/Cole Belmont.
13. Sanderson C (2010) Armadillo: An open source C++ linear algebra library for fast prototyping and computationally intensive experiments. Technical report, Nicta, St Lucia (Australia).
14. OpenMP Architecture Review Board (2008) OpenMP application program interface version 3.0. Available: http://www.openmp.org/mp-documents/spec30.pdf.
15. Schellenberger J, Park JO, Conrad TM, Palsson BØ (2010) Bigg: a biochemical genetic and genomic knowledgebase of large scale metabolic reconstructions. BMC bioinformatics 11: 213.
16. Cowles MK, Carlin BP (1996) Markov chain monte carlo convergence diagnostics: A comparative review. Journal of the American Statistical Association 91: 883–904.
17. Gelman A, Rubin DB (1992) Inference from iterative simulation using multiple sequences. Statistical Science 7: 457–472.
18. Geweke J (1992) Evaluating the accuracy of sampling-based approaches to the calculation of posterior moments. In: IN BAYESIAN STATISTICS. University Press, pp. 169–193.
19. Heidelberger P, Welch PD (1983) Simulation run length control in the presence of an initial transient. Operations Research 31: 1109–1144.
20. Plummer M, Best N, Cowles K, Vines K (2006) CODA: Convergence diagnosis and output analysis for MCMC. R News 6: 7–11.
21. Gorard S (2005) Revisiting a 90-year-old debate: The advantages of the mean deviation. British Journal of Educational Studies: 417–439.