A Scalable Parallel Algorithm for Balanced Sampling (Supplement)

Alexander Lee,*[1] Stefan Walzer-Goldfeld,*[1] Shukry Zablah,[2] Matteo Riondato[1]

[1] Box 2232, Dept. of Computer Science, Amherst College, Amherst, MA 01002, USA
[2] Pallet Labs Inc.
{awlee22, swalzergoldfeld23, mriondato}@amherst.edu, shukry@pallet.xyz

Related Work

Deville and Tillé (2004) present the cube method, an efficient balanced sampling method. They use a balancing martingale to perform a random walk on a hypercube to select approximately balanced samples with specific inclusion probabilities (see Algorithms below for a complete description). However, the runtime of the algorithm depends on the square of the population size N, which results in very slow runs for large population sizes.

Chauvet and Tillé (2006) provide an improved algorithm for the cube method that reduces memory usage significantly and has time complexity linear in the population size N. This algorithm only performs matrix operations on a small submatrix instead of the entire population, providing significant improvement. However, this algorithm is still too slow for sampling from very large populations, which are common in modern data analytics.

Chauvet (2009) proposes an extension of the cube method for stratified populations. It provides a sampling design such that the sample is balanced across all auxiliary variables while maintaining a fixed allocation within each stratum. This is achieved by performing the flight phase of the cube method on each individual stratum and pooling the results (see Alg. 4 below). This stratified method has a slightly worse balancing quality than the cube methods presented by Deville and Tillé (2004) and Chauvet and Tillé (2006), though it still generates an approximately balanced sample.

There also exist multiple implementations of the cube method. Tillé and Matei (2016) implemented the algorithm by Chauvet and Tillé (2006) in the R `cubesampling` package. Grafström and Lisic (2016) implemented the same method in R with C++ subroutines in the `BalancedSampling` package. Both of these methods are implemented sequentially, and as a result, perform poorly on datasets with large population sizes.

Description of Algorithms

The Cube Method

We now describe the cube method by Deville and Tillé (2004) in detail, as its two phases, the *flight* and the *landing*, are used also in our algorithm. For ease of presentation, we discuss the flight phase as in the original cube method rather than the more efficient one by Chauvet and Tillé (2006), but our implementation uses the latter.

The flight phase The objective of the flight phase is to randomly select a vertex of the hypercube C that intersects with $K = C \cap Q$ by following a *balancing martingale*. A *balancing martingale* is a discrete time stochastic process $\boldsymbol{\phi}(t) \doteq \langle \phi_1(t), \ldots, \phi_N(t) \rangle$ in $\mathbb{R}^N$ for an inclusion probability vector $\boldsymbol{\pi}$ and auxiliary variables $\mathbf{A} = \langle \mathbf{a}_1, \ldots, \mathbf{a}_N \rangle$ if

1. $\boldsymbol{\phi}(0) = \boldsymbol{\pi}$;
2. $\mathbb{E}[\boldsymbol{\phi}(t) \mid \boldsymbol{\phi}(t-1), \ldots, \boldsymbol{\phi}(0)] = \boldsymbol{\phi}(t-1), t = 1, 2, \ldots$;
3. $\boldsymbol{\phi}(t) \in K = C \cap Q = [0,1]^N \cap (\boldsymbol{\pi} + \ker(\mathbf{W}))$, $t = 1, 2, \ldots$, where $\mathbf{W} \doteq \langle \mathbf{a}_1/\pi_1, \ldots, \mathbf{a}_N/\pi_N \rangle$ with dimensions $n \times N$.

Deville and Tillé (2004) propose an implementation of a balancing martingale for the flight phase. Pseudocode for it is in Algorithm 1. The algorithm initializes the balancing martingale with $\boldsymbol{\phi} = \boldsymbol{\pi}$, as required by the first property above. Then it chooses an arbitrary non-zero vector $\mathbf{u} \in \ker(\mathbf{W})$ such that $u_k(t) = 0$ if $\phi_k(t) \in \{0, 1\}$. The balancing martingale will move along the line corresponding to this vector at the next time steps, i.e., $\boldsymbol{\phi}(t+1) = \boldsymbol{\phi}(t) + \lambda \mathbf{u}$, for some *step size* λ. Line 5 then computes two *candidate* step sizes λ_1^* and λ_2^* by taking the maximum values of step sizes along $\mathbf{u}$ that still ensure $\boldsymbol{\phi}(t+1)$ is within $[0,1]^N$. We then compute (line 6) the probability q that $\boldsymbol{\phi}(t+1) = \boldsymbol{\phi}(t) + \lambda_1^* \mathbf{u}$, and we use it to update $\boldsymbol{\phi}$ on line 7. The probability q is constructed to ensure that, at each step t, point 2 of the balancing martingale definition holds. With respect to its previous value, $\boldsymbol{\phi}$ has at least one additional entry with value in $\{0, 1\}$. A new vector $\mathbf{u}$ is chosen, and, provided it exists, the algorithm iterates. The loop terminates when it is impossible to update $\mathbf{u}$, at which point $\boldsymbol{\phi}$ is returned.

The landing phase If the vector $\boldsymbol{\phi}$ returned by the flight phase does not belong to $\{0,1\}^N$ (i.e., it is not a vertex of the hypercube C), then it does not actually represent a valid sample, despite satisfying the balancing equations. Thus we have to solve the *rounding problem* and find a nearby vertex of the hypercube that *approximately* satisfies the balancing

*These authors contributed equally.

Algorithm 1: Flight Phase

Input : Auxiliary variables $\mathbf{A} = \langle \mathbf{a}_1, \ldots, \mathbf{a}_N \rangle$; inclusion probabilities $\boldsymbol{\pi}$.
Output: Vector $\boldsymbol{\phi} \in [0,1]^N$

$\mathbf{W} \leftarrow \mathbf{A}/\boldsymbol{\pi}$ `// Entry-wise division`
$\boldsymbol{\phi} \leftarrow \boldsymbol{\pi}$
$\mathbf{u} \leftarrow$ arbitrary non-zero vector in $\ker(\mathbf{W})$ s.t. $u_k = 0$ if $\phi_k \in \{0,1\}$
while $\mathbf{u}$ *exists* **do**
  $\lambda_1^*, \lambda_2^* \leftarrow$ max. λ_1, λ_2 s.t. $0 \leq \boldsymbol{\phi} + \lambda_1 \mathbf{u} \leq 1$, and $0 \leq \boldsymbol{\phi} - \lambda_2 \mathbf{u} \leq 1$
  $q \leftarrow \lambda_2^*/(\lambda_1^* + \lambda_2^*)$
  $\boldsymbol{\phi} \leftarrow \begin{cases} \boldsymbol{\phi} + \lambda_1^* \mathbf{u} & \text{with prob. } q \\ \boldsymbol{\phi} - \lambda_2^* \mathbf{u} & \text{with prob. } 1-q \end{cases}$
  $\mathbf{u} \leftarrow$ arbitrary non-zero vector in $\ker(\mathbf{W})$ s.t. $u_k = 0$ if $\phi_k \in \{0,1\}$
return $\boldsymbol{\phi}$

Algorithm 2: Landing Phase (by Suppression of Variables)

Input : Auxiliary variables $\mathbf{A} = \langle \mathbf{a}_1, \ldots, \mathbf{a}_N \rangle$; vector $\boldsymbol{\phi}$.
Output: Sample $\boldsymbol{\phi}$.

$n \leftarrow$ `RowCount`$(\mathbf{A})$
while $\boldsymbol{\phi} \notin \{0,1\}^N$ *and* $n > 0$ **do**
  $\mathsf{actIdxs} \leftarrow$ indices in $\boldsymbol{\phi}$ such that $0 < \phi_k < 1$
  $\boldsymbol{\phi}[\mathsf{actIdxs}] \leftarrow$ `FlightPhase(`$\mathbf{A}[:n, \mathsf{actIdxs}], \boldsymbol{\phi}[\mathsf{actIdxs}]$`)`
  $n \leftarrow n - 1$
return $\boldsymbol{\phi}$

Algorithm 3: Cube Method

Input : Auxiliary variables $\mathbf{A} = \langle \mathbf{a}_1, \ldots, \mathbf{a}_N \rangle$; inclusion probabilities $\boldsymbol{\pi}$.
Output: Sample $\boldsymbol{\phi}$.

$\boldsymbol{\phi} \leftarrow \boldsymbol{\pi}$
$\mathsf{actIdxs} \leftarrow$ indices in $\boldsymbol{\pi}$ such that $0 < \pi_k < 1$
$\boldsymbol{\phi}' \leftarrow$ `FlightPhase(`$\mathbf{A}[:, \mathsf{actIdxs}], \boldsymbol{\pi}[\mathsf{actIdxs}]$`)`
$\boldsymbol{\phi}[\mathsf{actIdxs}] \leftarrow$ `LandingPhase(`$\mathbf{A}[:, \mathsf{actIdxs}], \boldsymbol{\phi}'$`)`
return $\boldsymbol{\phi}$

equations. The goal of the *landing phase* is to find such vertex.

Deville and Tillé (2004) propose two possible algorithms for the landing phase: landing by suppression of variables and landing by linear programming. Here, we only describe the former because it scales better. Pseudocode for the landing phase by suppression of variables is in Alg. 2. Let $\boldsymbol{\phi}$ be the vector returned by the flight phase. We refer to the indices of $\boldsymbol{\phi}$ corresponding to nonintegral components (i.e., the set of indices k, $1 \leq k \leq N$, such that $0 < \phi_k < 1$) as the *active indices* of $\boldsymbol{\phi}$. The elements in $\boldsymbol{\phi}$ that correspond to the active indices of $\boldsymbol{\phi}$ are the *active units* of $\boldsymbol{\phi}$. Similarly, we can also refer to the elements in $\mathbf{A}$ that correspond to the active indices of $\boldsymbol{\phi}$ as the active units of $\mathbf{A}$. The landing phase by suppression of variables repeatedly executes the flight phase with the active units of $\mathbf{A}$, but dropping one auxiliary variable per repetition, and the active units of $\boldsymbol{\phi}$. The repeated flight phases accumulate the sample in $\boldsymbol{\phi}$ until $\boldsymbol{\phi} \in \{0,1\}^N$. The resulting $\boldsymbol{\phi}$ is used as the sample, which is not perfectly balanced because it only satisfies a subset of the balancing equations. We only use the active units of $\boldsymbol{\phi}$ for each execution of the flight phase to prevent a division by zero error when computing $\mathbf{W} \leftarrow \mathbf{A}/\boldsymbol{\phi}$.

Algorithm 3 shows how the flight and landing phases come together to create the whole cube method.

Cube Method for Stratified Populations

We now describe Chauvet (2009)'s variant of the cube method for stratified populations, whose pseudocode is in Alg. 4.

1. (Lines 1–5) Initialize $\boldsymbol{\phi}$ with $\boldsymbol{\pi}$. Let U_{ha} denote the active units in stratum U_h, $h = 1, \ldots, H$. For each stratum U_h, $h = 1, \ldots, H$, compute a flight phase with $\mathbf{A}[:, U_{ha}]$ and $\boldsymbol{\pi}[U_{ha}]$, where $\mathbf{A}[:, U_{ha}]$ and $\boldsymbol{\pi}[U_{ha}]$ are respectively the active units of $\mathbf{A}$ and $\boldsymbol{\pi}$ for U_h. The results are accumulated into $\boldsymbol{\phi}$.
2. (Lines 6–8) Perform a flight phase with the active units of $\boldsymbol{\phi}$ and $\mathbf{A}' = \langle \mathbf{a}'_1, \ldots, \mathbf{a}'_N \rangle$, where, for $k = 1, \ldots, N$,
$$\mathbf{a}'_k \doteq \langle \phi_k \mathbb{1}[u_k \in U_1], \ldots, \phi_k \mathbb{1}[u_k \in U_H], \mathbf{a}_k^\intercal \phi_k / \pi_k \rangle^\intercal,$$
where $\mathbb{1}[\cdot]$ is the indicator function taking value 1 if the argument is true, and 0 otherwise. The H *new constraints* $\phi_k \mathbb{1}[u_k \in U_1], \ldots, \phi_k \mathbb{1}[u_k \in U_H]$ are used to restrict the support $\mathcal{S}$ to fixed-size samples in each stratum. This step produces modified inclusion probabilities $\boldsymbol{\phi}'$.
3. (Lines 9–10) Perform a landing phase with $\boldsymbol{\phi}'$ and the active units of $\mathbf{A}'$. The sample produced by the landing phase is then used to replace the active units of $\boldsymbol{\phi}$, which is then returned.

Parallel Algorithm for Balanced Sampling

We now describe more in depth our novel parallel algorithm for balanced sampling, which follows closely the stratified cube method. The pseudocode is reported in Alg. 5.

1. (Lines 1–6) Execute in parallel step 1 of the stratified cube method with fake strata $U'_1, \ldots, U'_p$, where p is the number of available processors.
2. (Lines 7–9) Execute step 2 of the stratified cube method with auxiliary variables $\mathbf{a}'_k = \langle \mathbf{a}_k^\intercal \phi_k / \pi_k \rangle^\intercal$ (instead of $\mathbf{a}'_k = \langle \phi_k \mathbb{1}[u_k \in U_1], \ldots, \phi_k \mathbb{1}[u_k \in U_H], \mathbf{a}_k^\intercal \phi_k / \pi_k \rangle^\intercal$), $k = 1, \ldots, N$. This step is performed using only the original auxiliary variables instead of prepending the fixed size sampling constraints because fixed sized sampling is only necessary for stratification: since the strata are fake, we do not care about how many units are sampled per stratum.

Algorithm 4: Stratified Cube Method

Input : Auxiliary variables $\mathbf{A} = \langle \mathbf{a}_1, \ldots, \mathbf{a}_N \rangle$;
inclusion probabilities $\boldsymbol{\pi}$;
strata $U_1, \ldots, U_H$.
Output: Sample $\boldsymbol{\phi}$.

$\boldsymbol{\phi} \leftarrow \boldsymbol{\pi}$
actIdxs $\leftarrow$ indices in $\boldsymbol{\pi}$ such that $0 < \pi_k < 1$
for $h \leftarrow 1$ **to** H **do**
  $U_{ha} \leftarrow U_h$[actIdxs]
  $\boldsymbol{\phi}[U_{ha}] \leftarrow$ `FlightPhase(`$\mathbf{A}[:, U_{ha}]$, $\boldsymbol{\pi}[U_{ha}]$`)`
actIdxs $\leftarrow$ indices in $\boldsymbol{\phi}$ such that $0 < \phi_k < 1$
$\mathbf{A}' \leftarrow \langle \mathbf{a}'_1, \ldots, \mathbf{a}'_N \rangle$ where, for $k = 1, \ldots, N$, $\mathbf{a}'_k \doteq \langle \phi_k \mathbb{1}[u_k \in U_1], \ldots, \phi_k \mathbb{1}[u_k \in U_H], \mathbf{a}_k^\intercal \phi_k / \pi_k \rangle^\intercal$
$\boldsymbol{\phi}' \leftarrow$ `FlightPhase(`$\mathbf{A}'$[:, actIdxs], $\boldsymbol{\phi}$[actIdxs]`)`
$\boldsymbol{\phi}$[actIdxs] $\leftarrow$ `LandingPhase(`$\mathbf{A}'$[:, actIdxs], $\boldsymbol{\phi}'$`)`
return $\boldsymbol{\phi}$

3. (Lines 10–11) Execute step 3 of the stratified cube method as is.

Algorithm 5: Parallel Cube Method

Input : Number of processors p;
auxiliary variables $\mathbf{A} = \langle \mathbf{a}_1, \ldots, \mathbf{a}_N \rangle$;
inclusion probabilities $\boldsymbol{\pi}$.
Output: Sample $\boldsymbol{\phi}$.

$U'_1, \ldots, U'_p \leftarrow$ `CreateFakeStrata(`$\mathbf{A}$`)`
$\boldsymbol{\phi} \leftarrow \boldsymbol{\pi}$
actIdxs $\leftarrow$ indices in $\boldsymbol{\pi}$ such that $0 < \pi_k < 1$
for $f \leftarrow 1$ **to** p **do** in parallel
  $U'_{fa} \leftarrow U'_f$[actIdxs]
  $\boldsymbol{\phi}[U'_{fa}] \leftarrow$ `FlightPhase(`$\mathbf{A}[:, U'_{fa}]$, $\boldsymbol{\pi}[U'_{fa}]$`)`
actIdxs $\leftarrow$ indices in $\boldsymbol{\phi}$ such that $0 < \phi_k < 1$
$\mathbf{A}' \leftarrow \langle \mathbf{a}'_1, \ldots \mathbf{a}'_N \rangle$ where, for $k = 1, \ldots, N$, $\mathbf{a}'_k = \langle \mathbf{a}_k^\intercal \phi_k / \pi_k \rangle^\intercal$
$\boldsymbol{\phi}' \leftarrow$ `FlightPhase(`$\mathbf{A}'$[:, actIdxs], $\boldsymbol{\phi}$[actIdxs]`)`
$\boldsymbol{\phi}$[actIdxs] $\leftarrow$ `LandingPhase(`$\mathbf{A}'$[:, actIdxs], $\boldsymbol{\phi}'$`)`
return $\boldsymbol{\phi}$

Additional Experiments

Figure 1 shows the runtimes for additional population sizes that were not included in the main document.

Fig. 2 describes the absolute relative deviation of the Horvitz-Thompson estimate of the population total for each auxiliary variable. The vector of absolute relative deviations is defined as

$$100 \frac{|\widetilde{\mathbf{T}} - \mathbf{T}|}{\mathbf{T}}, \tag{1}$$

where the division is entry-wise.

Each group of box-and-whiskers plots represents the absolute relative deviation for a specific auxiliary variable. The middle of each box-and-whiskers plot is the median absolute relative deviation. The top and bottom of the box represent the first and third quartiles, respectively. The whiskers are 1.5 times the interquartile range ($Q_1 - Q_3$).

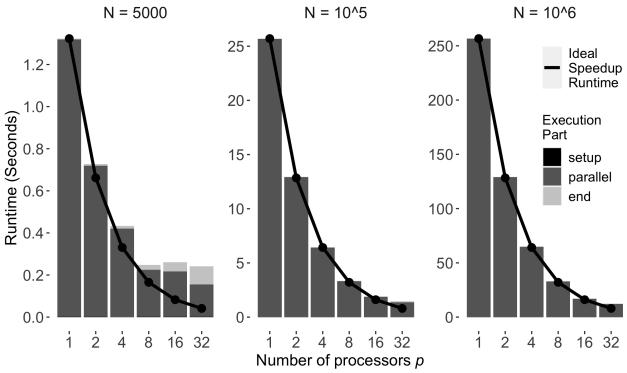

Figure 1: Parallel algorithm runtimes for different N and p.

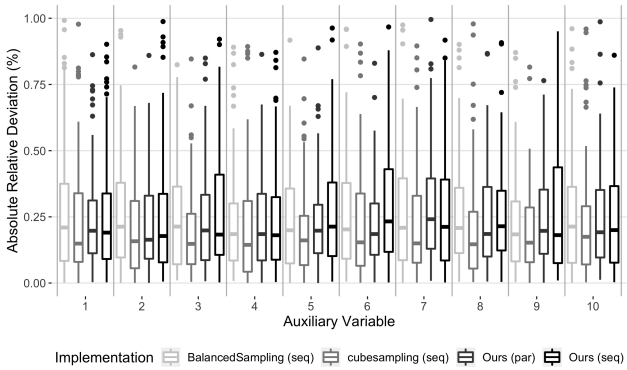

Figure 2: Absolute relative deviation by auxiliary variable.

Fig. 2 shows that our parallel algorithm creates samples that have very small absolute deviations for the population totals and similar to that of existing sequential methods for balanced sampling, thus confirming the correctness of our algorithm.

Acknowledgements

This work is funded, in part, by NSF award IIS-2006765.

References

Chauvet, G. 2009. Stratified balanced sampling. *Survey Methodology*, 35(1): 115–119.

Chauvet, G.; and Tillé, Y. 2006. A fast algorithm for balanced sampling. *Computational Statistics*, 21(1): 53–62.

Deville, J.-C.; and Tillé, Y. 2004. Efficient balanced sampling: the cube method. *Biometrika*, 91(4): 893–912.

Grafström, A.; and Lisic, J. 2016. *BalancedSampling: Balanced and spatially balanced sampling*.

Tillé, Y.; and Matei, A. 2016. *sampling: Survey Sampling*.