Available online at www.sciencedirect.com
SCIENCE DIRECT
Information Processing Letters 97 (2006) 181–185
Information Processing Letters
www.elsevier.com/locate/ipl

Weighted random sampling with a reservoir

Pavlos S. Efraimidis [a,*], Paul G. Spirakis [b]

[a] *Department of Electrical and Computer Engineering, Democritus University of Thrace, 67100 Xanthi, Greece*
[b] *Computer Technology Institute, Riga Feraiou 61, 26221 Patras, Greece*

Received 24 June 2004; received in revised form 16 October 2005; accepted 2 November 2005

Available online 2 December 2005

Communicated by F. Meyer auf der Heide

Abstract

In this work, a new algorithm for drawing a weighted random sample of size m from a population of n weighted items, where $m \leqslant n$, is presented. The algorithm can generate a weighted random sample in one-pass over unknown populations.

Keywords: Weighted random sampling; Reservoir sampling; Randomized algorithms; Data streams; Parallel algorithms

1. Introduction

The problem of random sampling without replacement (RS) calls for the selection of m distinct random items out of a population of size n. If all items have the same probability to be selected, the problem is known as uniform RS. In weighted random sampling (WRS) the items are weighted and the probability of each item to be selected is determined by its relative weight. WRS can be defined with the following Algorithm D:

WRS Definition (*Algorithm D*).

Input: *A population V of n weighted items*
Output: A set S with a WRS of size m
1: Repeat Steps 2 and 3 for $k = 1, 2, \ldots, m$
2: The probability of v_i to be selected is:
$p_i(k) = \frac{w_i}{\sum_{s_j \in V-S} w_j}$
3: Randomly select an item $v_k \in V - S$ and insert it into S

Definitions. *One-pass WRS* is the problem of generating a weighted random sample in one-pass over a population. If additionally the population size is initially unknown (dynamic populations, data streams, etc.), the random sample can be generated with *reservoir sampling* algorithms. These algorithms keep an auxiliary storage, the reservoir, with all items that are candidates for the final sample. *Weighted random permutation* (WRP) is the problem of generating a random permutation of all items, where the relative weight of each item determines the probability that it appears early in the permutation.

Related work. Algorithms for one-pass RS and reservoir RS are given, for example, in [6,12,13,7]. *These results concern uniform RS.* A fast parallel algorithm for uniform RS is presented in [11]. Also this result does not seem to be appropriate for WRS. The most important algorithms for WRS are the Alias Method,

* Corresponding author.
E-mail addresses: pefraimi@ee.duth.gr (P.S. Efraimidis), spirakis@cti.gr (P.G. Spirakis).

0020-0190/$ – see front matter
doi:10.1016/j.ipl.2005.11.003

Partial Sum Trees and the Acceptance/Rejection method (see [10] for a summary). *None of these algorithms is appropriate for one-pass WRS. To the authors knowledge, there are no algorithms for one-pass WRS, reservoir WRS, and distributed or parallel algorithms for any variation of WRS.*[1]

Our results. A new algorithm for WRS is presented. The algorithm, called Algorithm A, is simple, very flexible, and solves several interesting variations of the weighted sampling problem, like one-pass and reservoir WRS, and weighted random permutations. Furthermore, the basic algorithm reveals, in a natural way, efficient adaptations for sequential, parallel and distributed settings. Finally, an exponential jumps version of Algorithm A, is presented.

Notation and assumptions. All sampling problems are without replacement. The item weights are initially unknown, strictly positive reals. The population size is n, the size of the random sample is m and the weight of item v_i is w_i. The function $\mathit{random}(L, H)$ generates a uniform random number in (L, H). Given a random variable X, $F_X(x)$ and $f_X(x)$ are its distribution and density functions, respectively. Infinite precision arithmetic is assumed.

2. Algorithm A

The basic principle underlying Algorithm A, is the following remark that a uniform random variable can be arbitrarily "amplified" by raising it to an appropriate power. The correctness of Remark 1 is shown as part of the proof of Proposition 3.

Remark 1. Let U_1 and U_2 be independent random variables with uniform distributions in $[0, 1]$. If $X_1 = (U_1)^{1/w_1}$ and $X_2 = (U_2)^{1/w_2}$, for $w_1, w_2 > 0$, then

$$P[X_1 \leqslant X_2] = \frac{w_2}{w_1 + w_2}.$$

In Algorithm A, each item v_i of the population independently generates a random number $u_i = \mathit{random}(0, 1)$ and calculates a key $k_i = {u_i}^{1/w_i}$. The items that correspond to the m largest keys form a weighted random sample.

Algorithm A (*High level description*).

Input: A population V of n weighted items
Output: A WRS of size m
1: For each $v_i \in V$, $u_i = \mathit{random}(0, 1)$ and $k_i = u_i^{1/w_i}$
2: Select the m items with the largest keys k_i as a WRS

Let V be a problem instance of WRS with n weighted items. Note that sampling with $m = n$ reveals a permutation of the items. For Algorithm D, assigning each selected item to position i, where i is the round in which the item was selected, gives a permutation of all items. Similarly applying Algorithm A and sorting the items according to their key value (in non-increasing order) gives also a permutation of all items. Let Π be a permutation of the n items. For simplicity and without loss of generality we assume that $\Pi = v_n, v_{n-1}, \ldots, v_2, v_1$. Let $P_A(\Pi)$ and $P_D(\Pi)$ be the probabilities that Π is the outcome of Algorithms A and D, respectively.

Remark 2. For Algorithm D, the probability that w_n is selected first is $w_n/(w_1 + w_2 + \cdots + w_n)$. Given that w_n is the first item, the probability that w_{n-1} is the second item is $w_{n-1}/(w_1 + w_2 + \cdots + w_{n-1})$, etc. Hence

$$P_D(\Pi) = \prod_{i=1}^{n} \frac{w_i}{w_1 + w_2 + \cdots + w_i}.$$

For Algorithm A:

Proposition 3. *Let U_i, for $i = 1, 2, \ldots, n$, be independent random variables with uniform distribution in $(0, 1)$. For $i = 1, 2, \ldots, n$ and positive reals w_i, let X_i be the random variables $X_i = (U_i)^{1/w_i}$. Then for any real α in $[0, 1]$:*

$$P[X_1 \leqslant \cdots \leqslant X_n \leqslant \alpha] = \alpha^{w_1 + \cdots + w_n} \cdot \prod_{i=1}^{n} \frac{w_i}{w_1 + \cdots + w_i}.$$

Proof (*By induction*)**.** For $\alpha \in [0, 1]$ and $w_i > 0$, the distribution function $F_{X_i}(\alpha)$ is

$$F_{X_i}(\alpha) = P[X_i \leqslant \alpha] = P\left[(U_i)^{1/w_i} \leqslant \alpha\right] = P\left[U_i \leqslant \alpha^{w_i}\right] = \alpha^{w_i}$$

and the density function is $f_{X_i}(\alpha) = w_i \cdot \alpha^{w_i - 1}$.

$n = 1$: $P[U_1 \leqslant \alpha^{w_1}] = F_{X_1}(a^{w_1}) = \alpha^{w_1}$.

[1] Knuth [6] describes reservoir sampling algorithms for uniform RS. In one of the exercises (problem 16 of Section 3.4.2) Knuth refers to the problem of weighted sampling. The proposed solutions of the exercise (Problem 16 in Answers to Exercises, p. 555) indicate that the problem concerns conventional weighted sampling (not with a reservoir). More precisely, the proposed solutions (Walter's alias method and the algorithm of [14]) are not appropriate for one-pass WRS.

$n = 2$:

$$P[X_1 \leqslant X_2 \leqslant \alpha] = \int_0^\alpha F_{X_1}(t) f_{x_2}(t)\,\mathrm{d}t = \int_0^\alpha t^{w_1} w_2 t^{w_2-1}\,\mathrm{d}t = \frac{w_2}{w_1+w_2}\cdot\alpha^{w_1+w_2}.$$

Setting $\alpha = 1$ gives Remark 1: $P[X_1 \leqslant X_2] = w_2/(w_1+w_2)$.

$n = k$: Assume

$$P[X_1 \leqslant \cdots \leqslant X_k \leqslant \alpha] = \alpha^{w_1+\cdots+w_k}\prod_{i=1}^{k}\frac{w_i}{w_1+\cdots+w_i}.$$

$n = k+1$:

$$\begin{aligned}
&P[X_1 \leqslant \cdots \leqslant X_{k+1} < \alpha] \\
&= \int_0^\alpha P[X_1 \leqslant \cdots \leqslant X_k \leqslant t]\cdot f_{X_{k+1}}(t)\,\mathrm{d}t \\
&= \int_0^\alpha \left(\prod_{i=1}^{k}\frac{w_i}{w_1+\cdots+w_i}\right) t^{w_1+\cdots+w_k} w_{k+1} t^{w_{k+1}-1}\,\mathrm{d}t \\
&= \left(\prod_{i=1}^{k}\frac{w_i}{w_1+\cdots+w_i}\right) w_{k+1}\int_0^\alpha t^{w_1+\cdots+w_{k+1}-1}\,\mathrm{d}t \\
&= \alpha^{w_1+\cdots+w_{k+1}}\prod_{i=1}^{k+1}\frac{w_i}{w_1+\cdots+w_i}. \quad \square
\end{aligned}$$

From Remark 2 and Proposition 3:

Lemma 4. *For any permutation Π: $P_A(\Pi) = P_D(\Pi)$.*

For every item v_i of the population, the probability that v_i is selected in a sample of size m is equal to the sum of the probabilities of all permutations (of all items) where v_i is in one of the first m positions. This sum of probabilities is the same for Algorithms D and A (by Lemma 4). Hence:

Proposition 5. *Algorithm A generates a WRS.*

3. Efficient adaptations of Algorithm A

The two main operations required by Algorithm A, are the generation of an independent random key for each item and the selection of the m largest keys. Both operations can be implemented efficiently in sequential, distributed and parallel settings. With the use of a reservoir of size m to store the candidates for the final sample, all operations can be accomplished in a single scan over the entire (possibly unknown) population. A reservoir-type instantiation of Algorithm A is the following Algorithm A-Res:

Algorithm A *with a Reservoir* (*A-Res*).

Input: A population V of n weighted items
Output: A reservoir R with a WRS of size m
1: The first m items of V are inserted into R
2: For each item $v_i \in R$: Calculate a key $k_i = u_i^{1/w_i}$, where $u_i = \mathit{random}(0,1)$
3: Repeat Steps 4–7 for $i = m+1, m+2, \ldots, n$
4: The smallest key in R is the current threshold T
5: For item v_i: Calculate a key $k_i = u_i^{1/w_i}$, where $u_i = \mathit{random}(0,1)$
6: If the key k_i is larger than T, then:
7: The item with the minimum key in R is replaced by item v_i

3.1. The number of insertions

In uniform reservoir RS the expected number of insertions/deletions into the reservoir is $O(m\cdot\log(\frac{n}{m}))$ [3, p. 639]. Note that in the weighted problem the number of insertions can vary depending on the item weights. If the item weights are in increasing or decreasing order, then the number of insertions can be very large or very small, respectively. However we will show that in the general case, when the item weights are assumed to be independent random variables with a common distribution, the number of insertions/deletions is of the same order as in the uniform case.

Remark 6. Let $X_1, X_2, \ldots, X_k$ be independent random variables in the range $[0,1]$ with the same continuous distribution function $F_X(x)$ such that $F_X(0) = 0$ and $F_X(1) = 1$. Then $P[X_k$ is in the largest m out of k items$] = m/k$.

Proof.

$$\begin{aligned}
&P\big[X_{n+1} = \max\{X_i \mid i = 1,2,\ldots,n+1\}\big] \\
&= \int_0^1 P[X_1 \leqslant t]\cdots P[X_n \leqslant t]\cdot P[X_{n+1} = t]\,\mathrm{d}t
\end{aligned}$$

$$= \int_0^1 \big(F_X(t)\big)^n \cdot F'_X(t)\,\mathrm{d}t = \frac{1}{n+1}\int_0^1 \big(\big(F_X(t)\big)^{n+1}\big)'\,\mathrm{d}t$$
$$= \frac{1}{n+1}\big(F_X(1) - F_X(0)\big) = \frac{1}{n+1}.$$

Let $R(X_k, n)$ be the order of X_k in a sorted non-increasing list of n random variables. For example, $R(X_k, n) = 2$ represents the event that X_k has the second largest value among n random variables. Then:

$$\begin{aligned}
&P\big[R(X_k,k) \leqslant m\big] \\
&\quad = P\big[R(X_k,k) = 1\big] \\
&\qquad + P\big[R(X_k,k) \neq 1\big] \cdot \big(P\big[R(X_k,k-1) = 1\big] \\
&\qquad + P\big[R(X_k,k-1) \neq 1\big] \cdot \big(P\big[R(X_k,k-2) = 1\big] \\
&\qquad + P\big[R(X_k,k-2) \neq 1\big]\cdots\big)\big) \\
&\quad = \frac{1}{k} + \frac{k-1}{k} \cdot \left(\frac{1}{k-1} + \frac{k-2}{k-1}\right. \\
&\qquad \left.\cdot \left(\frac{1}{k-2} + \frac{k-3}{k-2} \cdot \left(\frac{1}{k-3} + \cdots\right)\right)\right) \\
&\quad = \frac{1}{k} + \sum_{i=2}^{m}\left(\frac{1}{k+1-i}\prod_{j=2}^{i}\left(\frac{k+1-j}{k+2-j}\right)\right) \\
&\quad = \sum_{i=1}^{m}\left(\frac{1}{k}\right) = \frac{m}{k}. \quad \square
\end{aligned}$$

Let $Y_1, Y_2, \ldots, Y_n, Y_{n+1}$ be independent random variables with the same continuous distribution function $F_Y(y)$, such that $F_Y(0) = 0$. Let Z_i be the independent random variables representing the random item keys: $Z_i = (X_i)^{Y_i}$, $i = 1, 2, \ldots, n, n+1$. Since $0 \leqslant X_i \leqslant 1$ and $Y_i > 0$, it follows that $F_Z(0) = 0$ and $F_Z(1) = 1$. Hence Remark 6 applies to the random variables $Z_i = (X_i)^{Y_i}$:

Proposition 7. *If A-Res is applied on n weighted items, where the weights $w_i > 0$ are independent random variables with a common continuous distribution, then the expected number of reservoir insertions* (*without the initial m insertions*) *is*:

$$\sum_{i=m+1}^{n} P[\textit{item } i \textit{ is inserted into } S]$$
$$= \sum_{i=m+1}^{n} \frac{m}{i} = \mathrm{O}\left(m \cdot \log\left(\frac{n}{m}\right)\right).$$

4. Sampling with exponential jumps

Let S_w be the sum of the weights of the items that will be skipped by A-Res until a new item enters the reservoir. If T_w is the current threshold to enter the reservoir, then S_w is a continuous random variable that follows an exponential distribution. Instead of generating a key for every item, it is possible to generate random jumps that correspond to the sum S_w. Similar techniques have been applied for uniform random sampling (see, for example, [3]). The following random variable X_w can be used to generate the appropriate exponential "jumps":

$$X_w = \frac{\log(\mathit{random}(0,1))}{\log(T_w)}.$$

The exponential jumps method reduces the number of random variates that have to be generated from $\mathrm{O}(n)$ to $\mathrm{O}(m\log(n/m))$. Since generating high-quality random variates can be a costly operation this is a significant improvement for the complexity of the sampling algorithm.

Algorithm A-ExpJ (*Steps 1 and 2 are identical to A-Res*).

3: The threshold T_w is the minimum key of R
4: Repeat Steps 5–10 until the population is exhausted
5: Let $r = \mathit{random}(0,1)$ and $X_w = \log(r)/\log(T_w)$
6: From the current item v_c skip items until item v_i, such that:
7: $w_c + w_{c+1} + \cdots + w_{i-1} < X_w \leqslant w_c + w_{c+1} + \cdots + w_{i-1} + w_i$
8: The item in R with the minimum key is replaced by item v_i
9: Let $t_w = {T_w}^{w_i}$, $r_2 = \mathit{random}(t_w, 1)$ and v_i's key: $k_i = (r_2)^{1/w_i}$
10: The new threshold T_w is the new minimum key of R

Assume that Algorithms A-Res and A-ExpJ are applied to the same problem instance of WRS and that both algorithms have processed $c-1$ items (for $c > m$). Also assume that the current threshold to enter the reservoir is T_w for both algorithms. Let $W_{i,j} = w_i + w_{i+1} + \cdots + w_j$ for $i \leqslant j$. The next item to be examined is v_c. For each $i = c, c+1, \ldots, n$ the probability $P_{\mathrm{next}}(v_i)$ that v_i is the next item that enters the reservoir R is:

- **For Algorithm A-Res:** $P^{\mathrm{Res}}_{\mathrm{next}}(v_i) = P[v_i \text{ enters } R] \cdot P[v_c, v_{c+1}, \ldots, v_{i-1} \text{ do not enter } R] = P[v_i \text{ enters } R] \cdot \prod_{k=c}^{i-1} P[v_k \text{ does not enter } R] = P[(U_i)^{1/w_i}$

$> T_w] \cdot \prod_{k=c}^{i-1} P[(U_k)^{1/w_k} \leqslant T_w] = (1 - T_w^{w_i}) \cdot \prod_{k=c}^{i-1} T_w^{w_k} = (T_w)^{W_{c,i-1}} - (T_w)^{W_{c,i}}$.

- **For Algorithm A-ExpJ:** Let U be the uniform random variable $U = random(0, 1)$. If X_w is the random step of Algorithm A-ExpJ, then v_i is the next selected item if $W_{c,i-1} = w_c + \cdots + w_{i-1} < X_w \leqslant w_c + \cdots + w_i = W_{c,i}$. The probability that this is true is: $P_{\text{next}}^{\text{ExpJ}}(v_i) = P[W_{c,i-1} < X_w \leqslant W_{c,i}] = P[W_{c,i-1} < \frac{\log(U)}{\log(T_w)} \leqslant W_{c,i}] = P[(T_w)^{W_{c,i-1}} < U \leqslant (T_w)^{W_{c,i}}] = F_U((T_w)^{W_{c,i-1}}) - F_U((T_w)^{W_{c,i}}) = (T_w)^{W_{c,i-1}} - (T_w)^{W_{c,i}}$.

Since $P_{\text{next}}^{\text{Res}}(v_i) = P_{\text{next}}^{\text{ExpJ}}(v_i)$, Algorithm A-ExpJ simulates A-Res. Hence:

Proposition 8. *Algorithm A-ExpJ generates a WRS.*

Each exponential jump corresponds to an insertion into the reservoir and hence the number of exponential jumps of A-ExpJ is given by Proposition 7. Interestingly, the probability of an item to be *the next item* that will enter the reservoir depends on the total weight of all items preceding it but not on their number.

5. Discussion

Random sampling is a fundamental problem in computer science with applications in many fields including databases [10,2], data mining, and randomized algorithms [4]. Consequently, Algorithm A for WRS is a general tool that can find applications in the design of randomized algorithms. For example, Algorithm A can be used within approximation algorithms for the k-Median [8], for a maximum flow problem [5] and to generate weighted random permutations.

The reservoir based versions of Algorithms A, A-Res and A-ExpJ, have very small requirements for auxiliary storage space (m keys organized as a heap) and during the sampling process their reservoir continuously contains a weighted random sample that is valid for the already processed data. This makes the algorithms applicable to the emerging area of algorithms for processing data streams [1,9].

Preliminary implementations[2] of the algorithms of this work certify the performance improvement of the exponential jump Algorithm A-ExpJ over Algorithm A-Res, and indicate that the finite precision arithmetic has no evident impact on the quality of the random samples of all algorithms. However, the question if, and to what extend, the finite precision arithmetic affects the algorithms remains an open problem.

[2] In Java (version 1.4.2_03), with double precision arithmetic. The random numbers are generated with the random number generator class "java.util.Random".

Acknowledgement

We wish to thank the anonymous referees for helpful remarks that improved the presentation of this paper.

References

[1] B. Babcock, S. Babu, M. Datar, R. Motwani, J. Widom, Models and issues in data stream systems, in: ACM PODS, 2002, pp. 1–16.
[2] S. Chaudhuri, R. Motwani, V. Narasayya, On random sampling over joins, in: Proc. 1999 ACM SIGMOD Conf., ACM Press, New York, 1999, pp. 263–274.
[3] L. Devroye, Non-Uniform Random Variate Generation, Springer-Verlag, Berlin, 1986.
[4] D.R. Karger, Random sampling in cut, flow, and network design problems, in: 26th ACM STOC, ACM Press, New York, 1994, pp. 648–657.
[5] D.R. Karger, M.S. Levine, Random sampling in residual graphs, in: 34th ACM STOC, ACM Press, New York, 2002, pp. 63–66.
[6] D.E. Knuth, Seminumerical Algorithms, second ed., The Art of Computer Programming, vol. 2, Addison-Wesley, 1981.
[7] K.-H. Li, Reservoir-sampling algorithms of time complexity $o(n(1 + \log(n/n)))$, ACM Trans. Math. Software 20 (4) (1994) 481–493.
[8] J.-H. Lin, J.S. Vitter, ϵ-approximations with minimum packing constraint violation, in: 24th ACM STOC, 1992, pp. 771–782.
[9] S. Muthukrishnan, Data streams: Algorithms and applications, Foundations Trends Theoret. Comput. Sci. 1 (2) (2005).
[10] F. Olken, Random sampling from databases, PhD thesis, Department of Computer Science, University of California at Berkeley, 1993.
[11] V. Rajan, R.K. Ghosh, P. Gupta, An efficient parallel algorithm for random sampling, Inform. Process. Lett. 30 (1989) 265–268.
[12] J.S. Vitter, Faster methods for random sampling, Comm. ACM 27 (7) (1984) 703–718.
[13] J.S. Vitter, Random sampling with a reservoir, ACM Trans. Math. Software 11 (1) (1985) 37–57.
[14] C.K. Wong, M.C. Easton, An efficient method for weighted sampling without replacement, SIAM J. Comput. 9 (1) (1980) 111–113.