bkshuf: A Shuf-Like Utility with Pre-Image Resistance and Relative Order Preservation for Random Sampling of Long Lists

by Steven Baltakatei Sandoval

2023-02-14

1 Summary

bkshuf is a shuf-like utility designed to output randomly sized groups of lines with a group size distribution centered around some characteristic value.

2 Objective

The author desires to create a shuf-like utility named bkshuf to mix line lists in order to produce output line lists with the following somewhat conflicting properties:

Pre-image resistance (PIR). An output line's position should not contain information about its input line position.

Relative order preservation (ROP). Two neighboring lines in the input stream should have a high probability of remaining neighbors in the output stream.

The objective is to improve the value of a short random scan of a small fraction of a potentially large input list; output that demonstrates ROP as well as some degree of PIR may achieve this objective. In contrast, the shuf utility provides PIR but no ROP: a line's neighbor in the output of shuf is equally likely to be any other random line from the input.

In other words, output produced by bkshuf should group together sequential segments of the input lines in order to partially preserve relationships that may exist between sequential files. For example, this could be done by jumping to a random position in the input lines, consuming (i.e. reading, outputting, and marking a line not to be read again) some amount of sequential lines, then repeating the process until every line is consumed. The amount of sequential lines to read between jumps affects how well the above desired properties are satisfied.

The objective of bkshuf is not to completely prevent the possibility of reassembling the input given the output. Additionally, a valuable property desired of bkshuf is output which demonstrates sufficiently high PIR compared to ROP such that only a short (compared to the logarithm of the input list size) sequential scan of the output list from a random starting position is required before a jump to a new group is is encountered; this would permit the overal contents of very large input line lists to be sampled.

3 Design

3.1 Definitions

$$l \ : \ \text{number of lines}$$

l_{in}	:	input line count
l_{out}	:	output line count
c	:	target group count
s	:	target group size
p_{seq}	:	probability to include next sequential line
$s(l_{\text{in},0})$	:	target group size parameter
$l_{\text{in},0}$	:	input line count parameter

3.2 Process

1. Acquire and count input lines (via `/dev/stdin` or positional arguments).

2. Calculate line count l_{in} .

3. Calculate target group size s.

4. Select random unconsumed input line and consume it to output.

5. Consume the next sequential line with probability p_{seq}. Otherwise if some input lines remain unconsumed, go to step 4. Otherwise, exit.

3.3 Parameter analysis

3.3.1 Target group size calculation

The simultaneous presence of ROP and PIR properties in the output depends upon the amount of sequential lines that are read before `bkshuf` jumps to a new random position in the input list. This amount is the *target group size*, s; it is the “target” since s represents the average of a distribution of group sizes that may be selected, not a single group size. In this analysis, the total number of lines in the input list is l_{in}. For small input line counts, (e.g. $l_{\text{in}} \cong 10$) the target group size should be nearly one (e.g. $s \cong 1$) since group sizes any larger than this would have almost no PIR (e.g. a group size of $s = 8$ for $l_{\text{in}} = 10$ would be 80% identical to the input). For modest line input counts (e.g. $l_{\text{in}} \cong 100$), the target group size may be allowed to be larger, such as a tenth of the input line count (e.g. $s \cong 10$); this would provide some PIR (approximately 10! permutations between the approximately $\frac{l_{\text{in}}}{s} \cong \frac{100}{10} \cong 10$ groups) while each line in groups around size 10 would have a low probability of not being next to its neighbor (8 of the 10 lines would retain the same two neighbors while the two ends would retain one each). For very large input line counts (e.g. $l_{\text{in}} \cong 1\,000\,000$), however, breaking up and randomizing the input into ten groups of 100 000 offers very little PIR; the benefit of the very high ROP is also lost since sequential scanning of tens of thousands of lines is required before a random jump to a new group may be encountered; therefore, the target group size should be a much smaller fraction of l_{in}, (e.g. $s \cong 20$) while still increasing. The relationship between a desireable target group size s and the input line count l_{in} is non-linear. The author believes a reasonable approach is to scale the group size to the logarithm of input line count.

Figure 1 shows an example plot of $s(l_{\text{in}})$ that is tuned to achieve a target group size of $s(l_{\text{in}} = 10^6) = 25$ for an input list length of $l_{\text{in}} = 10^6$ lines.

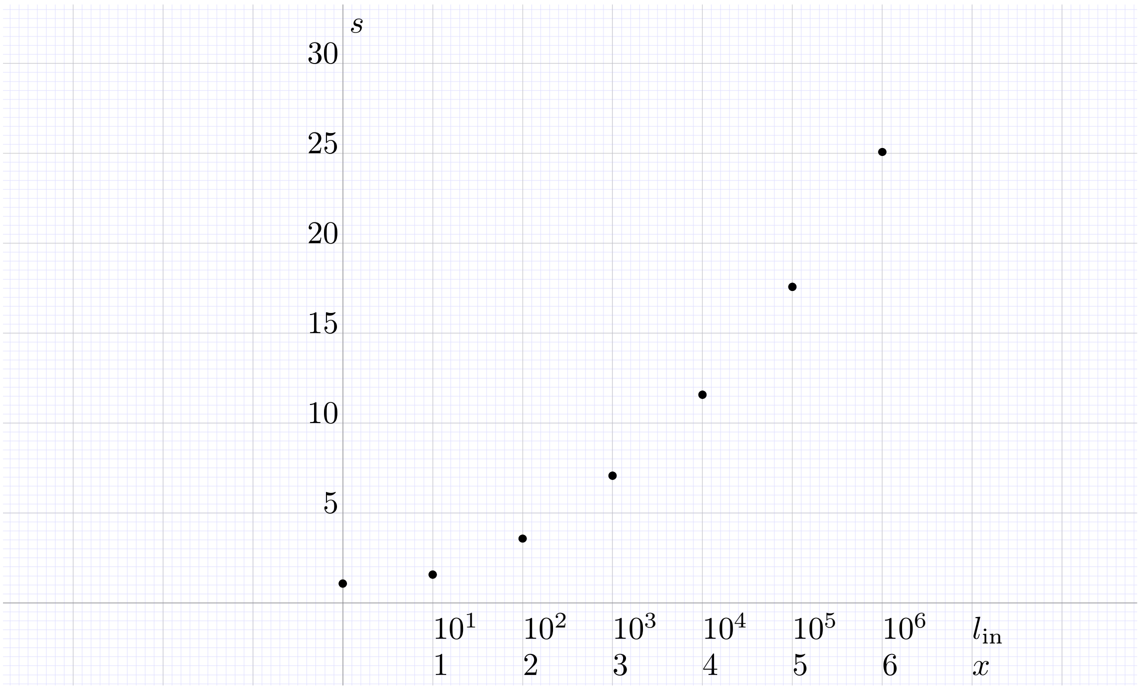

Figure 1. A plot of a possible function that relates target group size s and input lines l_{in} that provide some ROP and PIR. The function is tuned to achieve $s(l_{\mathrm{in}} = 10^6) = 25$.

The following is a set of equations that are used to derive a definition for $s(l_{\mathrm{in}})$ that satisfies the plot in Figure 1.

$$
\begin{align}
x &= \log\left(l_{\mathrm{in}}\right) = \frac{\ln\left(l_{\mathrm{in}}\right)}{\ln\left(10\right)} \tag{1}\\
10^x &= l_{\mathrm{in}} \nonumber\\
x_0 &= \log\left(l_{\mathrm{in},0}\right) = \frac{\ln\left(l_{\mathrm{in},0}\right)}{\ln\left(10\right)} \tag{2}\\
10^{x_0} &= l_{\mathrm{in},0} \nonumber
\end{align}
$$

$$
\begin{align}
s(x) &= (k\,x)^2 + 1 \tag{3}\\
s(x=6) = 25 &= k^2 \cdot (6)^2 + 1 \nonumber\\
25 &= k^2 \cdot (36) + 1 \nonumber\\
s(x_0) &= (k\,x_0)^2 + 1 \tag{4}\\
\frac{s(x_0) - 1}{x_0^2} &= k^2 \nonumber\\
k &= \sqrt{\frac{s(x_0) - 1}{x_0^2}} \nonumber\\
k &= \frac{\sqrt{s(x_0) - 1}}{x_0} \nonumber\\
k^2 &= \frac{s(x_0) - 1}{x_0^2} \tag{5}\\
s(l_{\mathrm{in}}) &= \left(\frac{s(x_0) - 1}{x_0^2}\right) \cdot \left(\frac{\ln(l_{\mathrm{in}})}{\ln\left(10\right)}\right)^2 + 1 \nonumber\\
&= \left(\frac{\ln\left(10\right)}{\ln\left(l_{\mathrm{in},0}\right)}\right)^2 \cdot (s(x_0) - 1) \cdot \left(\frac{\ln(l_{\mathrm{in}})}{\ln\left(10\right)}\right)^2 + 1 \nonumber
\end{align}
$$

$$
\begin{aligned}
s(l_{\text{in}}) &= (s(l_{\text{in},0})-1)\cdot\left(\frac{\ln(l_{\text{in}})}{\ln\left(l_{\text{in},0}\right)}\right)^2+1 \\
s(l_{\text{in}}) &= \left(\frac{s(l_{\text{in},0})-1}{[\ln\left(l_{\text{in},0}\right)]^2}\right)\cdot[\ln(l_{\text{in}})]^2+1 \qquad (6)
\end{aligned}
$$

Equation 3 defines a quadratic equation for the linear range s and the logarithmic domain x. x is defined in terms of l_{in} via a domain transformation defined by Equation 1. The result is Equation 6 which defines $s(l_{\text{in}})$ as a function of l_{in} and parameters $s(l_{\text{in},0})$ and $l_{\text{in},0}$. The parameters define how quickly or slowly the quadratic equation grows. In other words, if a user wishes for a 1 000 000 line input to be split into groups each containing, on average, 25 lines, then they should plug in $l_{\text{in},0}=1\,000\,000$ and $s(l_{\text{in},0})=25$ into Equation 6 as is done in Equation 7. This equation can then be used to calculate target group sizes s as a function of other input line counts l_{in} besides $l_{\text{in}}=1\,000\,000$. For example, plugging $l_{\text{in}}=500$ into Equation 7 yields Equation 8 which specifies a target group size of $5.85629\cong 6$.

$$
\begin{aligned}
s(l_{\text{in}}) &= \left(\frac{s(l_{\text{in},0})-1}{[\ln\left(l_{\text{in},0}\right)]^2}\right)\cdot[\ln(l_{\text{in}})]^2+1 \\
s(l_{\text{in}}) &= \left(\frac{25-1}{[\ln\ (1\,000\,000)]^2}\right)\cdot[\ln(l_{\text{in}})]^2+1 \qquad (7)
\end{aligned}
$$

$$
\begin{aligned}
s(l_{\text{in}}=500) &= \left(\frac{25-1}{[\ln\ (1\,000\,000)]^2}\right)\cdot[\ln(500)]^2+1 \qquad (8)\\
s(l_{\text{in}}=500) &= 5.85629
\end{aligned}
$$

3.3.2 Jump from expected value

A method `bkshuf` may employ to decide when read the next sequential unconsumed input line is to simply do so with probability p_{seq} such that the expected value of the average group size trends towards s.

$$
\begin{aligned}
p_{\text{seq}} &= (1-p_{\text{jump}}) \\
p_{\text{jump}} &= 1-p_{\text{seq}} \\
s &= \frac{1}{p_{\text{jump}}}=\frac{1}{1-p_{\text{seq}}} \qquad (9)\\
s &= \frac{1}{1-p_{\text{seq}}} \\
1-p_{\text{seq}} &= \frac{1}{s} \\
p_{\text{seq}}-1 &= \frac{-1}{s} \\
p_{\text{seq}} &= 1-\frac{1}{s(l_{\text{in}})} \qquad (10)\langle\text{label}|\text{eq pseq-from-s-lin}\rangle \\
p_{\text{jump}} &= \frac{1}{s(l_{\text{in}})} \qquad (11)\langle\text{label}|\text{eq pjump-from-s-lin}\rangle
\end{aligned}
$$

$$
p_{\text{seq}}(l_{\text{in}}) = 1-\left[\left(\frac{s(l_{\text{in},0})-1}{[\ln\left(l_{\text{in},0}\right)]^2}\right)\cdot[\ln(l_{\text{in}})]^2+1\right]^{-1} \qquad (12)\langle\text{label}|\text{eq pseq-from-s-lin-exp}\rangle
$$

3.3.3 Jump from random variate of inverse gaussian distribution

Another method `bkshuf` may employ to decide when to read the next sequential unconsumed input line is to use an inverse gaussian distribution. This may be done by generating from the distribution a float sampled from the inverse gaussian with range 0 to infinity with mean μ whenever a new random position in the input list is selected; the float is rounded to the nearest integer.[1] Then, after consuming an input line, this integer is decremented by one and another sequential line is consumed provided the integer does not become less than or equal to zero. The inverse gaussian distribution requires specifying the mean μ and the shape parameter λ; a higher λ results in a greater spread. An upper bound may also be specified since the distribution has none except for that imposed by its programming implementation.

The result of using an inverse gaussian distribution is an output with potentially much more regular group sizes than using the previously mentioned expected value method. However, the implementation of the inverse gaussian sampling operation described by (Michael, 1976) requires several exponent calculations and a square root calculation in addition to various multiplication and division operations. If sufficient processing power is available, this may not necessarily be an issue.

3.3.4 Output structure

Regardless of whether group sizes are determined by the expected value method or using variates of an inverse gaussian distribution, mimicking the `shuf` property of all input lines being present in the output, albeit rearranged, results in a side effect: the first output lines are more likely to contain groups with uninterrupted sequence runs (high ROP) while groups in the last output lines are almost certain to contain sequence jumps within a group (less ROP). The reason for this is that `bkshuf`, when it encounters an input line that has already been consumed, will skip to the next available input line. The decision could be made to skip to a new random line, but, it is simpler to simply read the next available input line. The author's original intention of sampling only a short initial portion of the output is compatible with the behavior that ROP is preserved mostly at the beginning of the output.

4 Version History

Version No.	Date	Path	Description
`0.0.1`	2023-02-14	`unitproc/bkshuf`	Initial draft implemented in Bash.

Table 1. A table listing versions of `bkshuf`.

v0.0.1. Initial implementation in `bash 5.1.16` with `bc 1.07.1` and GNU Coreutils `8.32` and tested on Pop!_OS `22.04 LTS`. Saved to the author's BK-2020-03 repository[2].

[1] See Michael, John R. "Generating Random Variates Using Transformations with Multiple Roots". 1976. https://doi.org/10.2307/2683801 .

[2] See commit `22772fe` at https://gitlab.com/baltakatei/baltakatei-exdev .