Algorithms for Molecular Biology

RESEARCH **Open Access**

A linear-time algorithm that avoids inverses and computes Jackknife (leave-one-out) products like convolutions or other operators in commutative semigroups

John L. Spouge[1*], Joseph M. Ziegelbauer[2] and Mileidy Gonzalez[3]

Abstract

Background: Data about herpesvirus microRNA motifs on human circular RNAs suggested the following statistical question. Consider independent random counts, not necessarily identically distributed. Conditioned on the sum, decide whether one of the counts is unusually large. Exact computation of the p-value leads to a specific algorithmic problem. Given n elements $g_0, g_1, \ldots, g_{n-1}$ in a set G with the closure and associative properties and a commutative product without inverses, compute the jackknife (leave-one-out) products $\bar{g}_j = g_0 g_1 \cdots g_{j-1} g_{j+1} \cdots g_{n-1}$ $(0 \leq j < n)$.

Results: This article gives a linear-time Jackknife Product algorithm. Its upward phase constructs a standard segment tree for computing segment products like $g_{[i,j)} = g_i g_{i+1} \cdots g_{j-1}$; its novel downward phase mirrors the upward phase while exploiting the symmetry of g_j and its complement $\bar{g}_j$. The algorithm requires storage for $2n$ elements of G and only about $3n$ products. In contrast, the standard segment tree algorithms require about n products for construction and $\log_2 n$ products for calculating each $\bar{g}_j$, i.e., about $n \log_2 n$ products in total; and a naïve quadratic algorithm using $n - 2$ element-by-element products to compute each $\bar{g}_j$ requires $n(n-2)$ products.

Conclusions: In the herpesvirus application, the Jackknife Product algorithm required 15 min; standard segment tree algorithms would have taken an estimated 3 h; and the quadratic algorithm, an estimated 1 month. The Jackknife Product algorithm has many possible uses in bioinformatics and statistics.

Keywords: Commutative semigroup, Leave-one-out, Jackknife products, Segment tree, Data structure

Background

A biological question

Circular RNAs (circRNAs) are single-stranded noncoding RNAs that can inhibit another RNA molecule by binding to it, mopping it up like a sponge. During herpesvirus infection, human hosts produce circRNAs with target sites that may bind herpesvirus microRNA (miRNA) [1] (see Fig. 1). Given a sequence motif, e.g., a target site for a miRNA, researchers counted how many times the motif occurs in each circRNA sequence. They then posed a question: is the motif unusually enriched in any of the circRNAs, i.e., does any circRNA have too many occurrences of the motif to be explained by chance alone? If "yes", the researchers could then focus their further experimental efforts on those circRNAs.

A statistical answer

Figure 1 illustrates a set of circRNAs with varying length, with a single miRNA motif occurring as

*Correspondence: spouge@nih.gov
[1] National Center for Biotechnology Information, National Library of Medicine, National Institutes of Health, Room 6N603, Building 38A, Bethesda, MD 20894, USA
Full list of author information is available at the end of the article

© The Author(s) 2020. This article is licensed under a Creative Commons Attribution 4.0 International License, which permits use, sharing, adaptation, distribution and reproduction in any medium or format, as long as you give appropriate credit to the original author(s) and the source, provide a link to the Creative Commons licence, and indicate if changes were made. The images or other third party material in this article are included in the article's Creative Commons licence, unless indicated otherwise in a credit line to the material. If material is not included in the article's Creative Commons licence and your intended use is not permitted by statutory regulation or exceeds the permitted use, you will need to obtain permission directly from the copyright holder. To view a copy of this licence, visit http://creativecommons.org/licenses/by/4.0/. The Creative Commons Public Domain Dedication waiver (http://creativecommons.org/publicdomain/zero/1.0/) applies to the data made available in this article, unless otherwise stated in a credit line to the data.

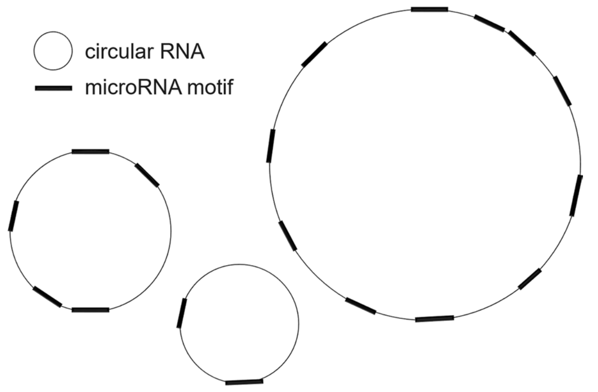

Fig. 1 A schematic diagram of herpesvirus miRNA motif occurring on a human circRNA. As indicated in the legend, each thin circle represents a circRNA; each thick line segment, the occurrence of a miRNA motif on the corresponding circRNA. Both circRNAs and the miRNA motif have nucleotide sequences represented by IUPAC codes (A, C, G, U). This figure illustrates occurrences of a single miRNA motif (e.g., UUACAGG) on the circRNAs. The biological question is: "does any circRNA have too many occurrences of the motif to be explained by chance alone?" In the actual application, the $n = 3086$ circRNAs ranged in length from 69 nt to 158565 nt

indicated on each circRNA. Let $i = 0, 1 \ldots, n-1$ index the circRNAs; the random variate X_i count the motif occurrences in the i-th circRNA; $k(i)$ equal the observed count for X_i; and the sum $S = \sum_{i=0}^{n-1} X_i$ count the total motif occurrences among the circRNAs, with observed total $K = \sum_{i=0}^{n-1} k(i)$.

The following set-up provides a general statistical test for deciding the biological question. Let $\{X_i : i = 0, 1, \ldots, n-1\}$ represent independent random counts (i.e., non-negative integer random variates), not necessarily identically distributed, with sum $S = \sum_{i=0}^{n-1} X_i$. Given observed values $\{X_i = k(i) : i = 0, 1, \ldots, n-1\}$ with observed sum $K = \sum_{i=0}^{n-1} k(i)$, consider the computation of the conditional p-values $\mathbb{P}\{X_i \geq k(i) | S = K\}$ $(i = 0, 1, \ldots, n-1)$. The conditional p-values can decide the question: "Is any term in the sum unusually large relative to the others?"

The abstract question in the previous paragraph generalizes some common tests. For example, the standard 2×2 Fisher exact test [2, p. 96] answers the question in the special case of $n = 2$ categories: each X_i has a binomial distribution with common success probability p, conditional on known numbers of trials N_i $(i = 0, 1)$. Although the Fisher exact test generalizes directly to a single exact p-value for a $2 \times n$ table [3], the generalization can require prohibitive amounts of computation. The abstract question corresponds to a computationally cheaper alternative that also decides which columns in the $2 \times n$ table are unusual [4].

To derive an expression for the conditional p-value, therefore, let $g_i[k] = \mathbb{P}\{X_i = k\}$ be given, so the array $g_i = (g_i[0], g_i[1], \ldots, g_i[K])$ gives the distribution of X_i, truncated at the observed total $K = \sum_{i=0}^{n-1} k(i)$. Because g_i is a truncated probability distribution, $g_i \in G$, the set of all real $(K+1)$-tuples $(g[0], g[1], \ldots, g[K])$ satisfies $g[k] \geq 0$ $(k = 0, 1, \ldots, K)$ and $\sum_{k=0}^{K} g[k] \leq 1$. The truncation still permits exact calculation of the probabilities below. To calculate the distribution of the sum $S = \sum_{i=0}^{n-1} X_i$ for $S \leq K$, define the truncated convolution operation $g = g' \circ g''$, for which $g[k] = \sum_{j=0}^{k} g'[j] g''[k-j]$ $(k = 0, 1, \ldots, K)$. Hereafter, the operation "$\circ$" is often left implicit: $g' \circ g'' = g'g''$.

Let $\bar{g} = g_0 g_1 \cdots g_{n-1}$, so $\bar{g}[k] = \mathbb{P}\{S = k\}$ $(k = 0, 1, \ldots, K)$. Define the "jackknife products" $\bar{g}_j = g_0 g_1 \cdots g_{j-1} g_{j+1} \cdots g_{n-1}$ $(0 \leq j < n)$ (implicitly including the products $\bar{g}_0 = g_1 g_2 \cdots g_{n-1}$ and $\bar{g}_{n-1} = g_0 g_1 \cdots g_{n-2}$). The jackknife products contain the same products as $\bar{g}$, except that in turn each skips over g_j $(0 \leq j < n)$. Like the jackknife procedure in statistics, therefore, jackknife products successively omit each datum in a dataset [5].

With the jackknife products in hand, the conditional p-values are a straightforward computation:

$$\mathbb{P}\{X_i \geq k(i) | S = K\} = \frac{\sum_{k=k(i)}^{K} g_i[k] \bar{g}_i[K-k]}{\bar{g}[K]}. \tag{1}$$

With respect to Eq. (1) and the biological question in Fig. 1, Appendix B gives the count $\bar{g}[K]$ of the ways that n circRNAs of known but varying length may contain K miRNA motifs of equal length, the count $g_i[k]$ of the ways that the i th circRNA may contain k motifs, and the count $\bar{g}_i[K-k]$ of the ways that all circRNAs but the i th may contain $K-k$ motifs. Appendix B derives the count $g_i[k]$ for circRNAs from the easier count for placing motifs on a linear RNA molecule. For combinatorial probabilities like $\mathbb{P}\{X_i \geq k(i) | S = K\}$, Eq. (1) remains relevant, even if $\{g_i[k]\}$ are counts instead of probabilities. The biological question therefore exemplifies a commonplace computational need in applied combinatorial probability.

The Discussion indicates that in our application, transform methods can encounter substantial obstacles when computing Eq. (1) (e.g., see [6]), because the quantities in Eq. (1) can range over many orders of magnitude. This article therefore pursues direct exact calculation of $\mathbb{P}\{X_i \geq k(i) | S = K\}$. The product forms of $\bar{g}$ and $\{\bar{g}_j\}$ suggest that any efficient algorithm may be abstracted to broaden its applications, as follows.

Semigroups, groups, and commutative groups

Let $(G, \circ)$ denote a set G with a binary product $g \circ g'$ on its elements. Let "$g \in G$" denote "g is an element of G", and consider the following properties [7].

(1) *Closure* $g \circ g' \in G$ for every $g, g' \in G$
(2) *Associative* $\left(g \circ g'\right) \circ g'' = g \circ \left(g' \circ g''\right)$ for every $g, g', g'' \in G$
(3) *Identity* There exists an identity element $e \in G$, such that $e \circ g = g \circ e = g$ for every $g \in G$
(4) *Commutative* $g \circ g' = g' \circ g$ for every $g, g' \in G$

If the Closure and Associative properties hold, $(G, \circ)$ is a semigroup. Without loss of generality, we assume below that the Identity property holds. If not, adjoin an element $e \in G$, such that $e \circ g = g \circ e = g$ for every $g \in G$. In addition, if the Commutative property holds for every $g, g' \in G$, the semigroup $(G, \circ)$ is commutative. Unless stated otherwise hereafter, $(G, \circ)$ denotes a commutative semigroup. The Jackknife Product algorithm central to this article is correct in a commutative semigroup.

(5) *Inverse* For every $g \in G$, there exists an inverse $g^{-1} \in G$, such that $g \circ g^{-1} = g^{-1} \circ g = e$

As shown later, the Jackknife Product algorithm does not require the Inverse property. In passing, note that the convolution semigroup relevant to the circRNA–miRNA application lacks the Inverse property, as does any convolution semigroup for calculating p-values, e.g., the ones relevant to sequence motif matching [6]. To demonstrate, let $X, Y \geq 0$ be independent integer random variates. The identity e for convolution corresponds to the variate $Z = 0$, because $0 + X = X + 0 = X$ for every variate X. If $X + Y = 0$, however, the independence of X and Y implies that both are constant and therefore $X = Y = 0$. In the relevant convolution semigroup, therefore, all elements except the identity e lack an inverse.

The non-zero real numbers under ordinary multiplication form a commutative semigroup $(G, \circ)$ with the Inverse property. They provide a familiar setting for discussing some algorithmic issues when computing $\left\{\bar{g}_j\right\}$. Let $\bar{g} = g_0 g_1 \cdots g_{n-1}$ be the usual product of n real numbers, and consider the toy problem of computing all jackknife products $\left\{\bar{g}_j\right\}$ that omit a single factor g_j $(0 \leq j < n)$ from $\bar{g}$. Inverses $\left\{g_j^{-1}\right\}$ are available, so an obvious algorithm computes $\bar{g}$ and then $\left\{\bar{g}_j = \bar{g} g_j^{-1}\right\}$ with n inverses and $2n - 1 = (n-1) + n$ products. If the inverses were unavailable, however, the naïve algorithm using $n - 2$ element-by-element products to compute each $\left\{\bar{g}_j\right\}$ would require $n(n-2)$ products. The quadratic time renders the naïve algorithm impractical for many applications.

Figure 1 illustrates a standard data structure called a segment tree, omitting the root at the top of the segment tree. Algorithms based solely on a segment tree can calculate the jackknife products $\left\{\bar{g}_j\right\}$ in time $O\left(n \log n\right)$, fast enough for many applications. The segment tree computes segment products like $g_{[i,j)} = g_i g_{i+1} \cdots g_{j-1}$ without using the commutative property, so it can similarly compute jackknife products like $\left\{\bar{g}_j = g_{[0,j)} g_{[j+1,n)}\right\}$. If the semigroup $(G, \circ)$ is commutative, however, a Jackknife Product algorithm can avoid inverses and reduce the computational time further, from $O\left(n \log n\right)$ to $O(n)$. With in-place computations requiring only the space for the segment tree, the Jackknife Product algorithm avoids inverses yet still requires only about $3n$ products and storage for $2n$ numbers. It is therefore surprisingly economical, even when compared to the obvious algorithm using inverses. Indeed, our application to circular RNA required some economy, with its convolution of $n = 3086$ distributions, some truncated only after $K = 997$ terms. In a general statistical setting, convolutions form a commutative semigroup $(G, \circ)$ without inverses, so our application already indicates that the Jackknife Product algorithm has broad applicability.

Theory

Appendix A proves the correctness of the Jackknife Product algorithm given below.

The Jackknife Product algorithm

Let $(G, \circ)$ be a commutative semigroup. The Jackknife Product algorithm has three phases: upward, downward, and transposition. Its upward phase simply constructs a segment tree (see Fig. 1); its downward phase exploits the symmetry of g_j and its complement $\bar{g}_j$ to mirror the upward phase while computing $\left\{\bar{g}_j\right\}$ (see Fig. 2); and its final transposition phase then swaps successive pairs in an array (not pictured). As Figs. 1 and 2 suggest, the three phases yield a simpler algorithm if $n = n^* = 2^m$ is a binary power. To recover the $n^* = 2^m$ algorithm from them, pad $\left\{g_j\right\}$ on the right with copies of the identity e up to n^* elements, where $n^* = 2^m$ is the smallest binary power greater than or equal to n, i.e., replace $\left\{g_j\right\}$ with $\left\{g_0, g_1 \ldots, g_{n-1}, e, \ldots, e\right\}$, with $n^* - n$ copies of e. The $n^* = 2^m$ algorithm can therefore pad any input of n elements up to $n^* = 2^m$ elements without loss of generality. The algorithm given below is therefore slightly more intricate than the $n^* = 2^m$ algorithm, but it may save almost a factor of 2 in storage and time by omitting the padded copies of e. In any case, the simpler algorithm can always be recovered from the phases for general n given here, if desired.

We start with notational preliminaries. Define the floor function $\lfloor x \rfloor = \max\left\{j : j \leq x\right\}$ and the ceiling

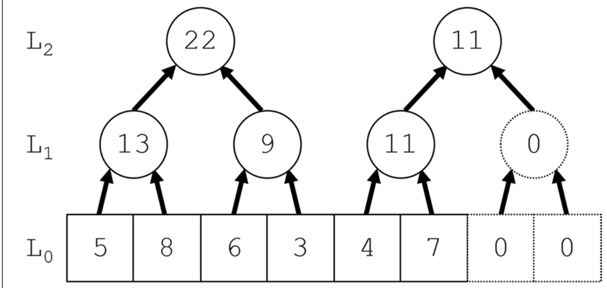

Fig. 2 A (rootless) segment tree. This figure illustrates the rootless segment tree constructed in the upward phase of the Jackknife Product algorithm. The commutative semigroup $(G, \circ)$ illustrated is the set of nonnegative integers under addition. The bottom row of $n^* = 2^m$ squares ($m = 3$) contains $L_0[j] = g_j$ $(0 \le j < n^*)$. In the next row up, as indicated by the arrow pairs leading into each circle, the array L_1 contains consecutive sums of consecutive disjoint pairs in L_0, e.g., $L_1[0] = 13 = 5 + 8$. The rest of the segment tree is constructed recursively upward to L_{m-1}, just as L_1 was constructed from L_0. Here, 2 copies of the additive identity $e = 0$ pad out L_0 on the right. Padded on the right, the copies contribute literally nothing to the segment tree above them. Their non-contributions have dotted outlines

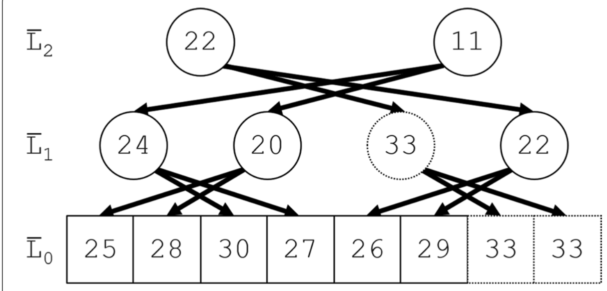

Fig. 3 A (rootless) complementary segment tree. This figure illustrates the rootless complementary segment tree constructed in the downward phase of the Jackknife Product algorithm from the rootless segment tree in Fig. 2. The downward phase starts by initializing the topmost row $\bar{L}_{m-1}$ ($m = 3$) with the topmost row L_{m-1} of the rootless segment tree. The row L_2 in Fig. 2 and the row $\bar{L}_2$ in Fig. 3, e.g., contain 22 and 11. For each $\bar{L}_{k-1}[j]$ in Fig. 3, downward arrows run from $\bar{L}_k[\alpha_k(j)]$ to $\bar{L}_{k-1}[j]$. As they indicate, each node in $\bar{L}_k$ contributes to its 2 "nieces" in Fig. 2 to produce the next row down in Fig. 3, e.g., $\bar{L}_2[1] = 11$ contributes to its nieces $L_1[0] = 13$ and $L_1[1] = 9$ in the segment, to produce $\bar{L}_1[0] = 13 + 11 = 24$ and $\bar{L}_1[1] = 9 + 11 = 20$ in the complementary segment tree. The rest of the complementary segment tree is constructed recursively downward to $\bar{L}_0$, just as $\bar{L}_1$ was constructed from $\bar{L}_2$. In Fig. 2, the elements of L_0 (in squares) total 33. To demonstrate the effect of the Jackknife Product algorithm, subtract in turn in Fig. 3 each element (25, 28, 30, 27, 26, 29, 33, 33) in the bottom row $\bar{L}_0$ from the total 33. The result (8, 5, 3, 6, 7, 4, 0, 0) is the bottom row L_0 in Fig. 2 with successive pairs transposed, so $\bar{L}_0[j] = \bar{g}_{\tau(j)}$, or equivalently $\bar{g}_j = \bar{L}_0[\tau(j)]$

function $\lceil x \rceil = \min\{j : x \le j\}$ (both standard); and the binary right-shift function $\rho(j) = \lfloor j/2 \rfloor$. Other quantities also smooth our presentation. Given a product $\bar{g} = g_0 g_1 \cdots g_{n-1}$ of interest, define $m = \lceil \log_2 n \rceil$ and $n_k = \lceil n 2^{-k} \rceil$ for $0 \le k < m$. Below, the symbol "□" connotes the end of a proof.

The upward phase

The upward phase starts with the initial array $L_0[j] = g_j$ $(0 \le j < n)$ and simply computes a standard (but rootless) segment tree consisting of segment products $L_k[j]$ for $j = 0, 1, \ldots, n_k - 1$ and $k = 0, 1, \ldots, m - 1$.

```
The upward phase

    for ($j=0$; $j<n$; $j$++) do
        $L_0[j]=g_j$;
    for ($k=1$; $n_k>2$; $k$++) do
        for ($j=0$; $j<\rho(n_{k-1})$; $j$++) do
            $L_k[j]=L_{k-1}[2j]\circ L_{k-1}[2j+1]$;
        if ($n_{k-1}$ is odd) $L_k[\rho(n_{k-1})]=L_{k-1}[n_{k-1}-1]$;
```

Comments

(1) If $n = n^* = 2^m$ is a binary power, $\rho(n_{k-1}) = n_k = 2^{m-k}$ and the final line in the upward phase can be omitted. (2) Of some peripheral interest, Laaksonen [8] gives the algorithm in a different context, embedding a binary tree in a single array of length $O(n)$. If any $L_0[j] = g_j$ changes, he also shows how to update the single array with $O(\log n)$ multiplications. If the downward phase (next) does not overwrite the segment tree $\{L_k\}$ by using in-place computation, it permits a similar update.

The downward phase

The transposition function $\tau(j) = j + (-1)^j$ transposes adjacent indices, e.g., $(L_{\tau(0)}, L_{\tau(1)}, L_{\tau(2)}, L_{\tau(3)}) = (L_1, L_0, L_3, L_2)$. We also require $\alpha_k(j) = \min\{\tau\rho(j), n_k - 1\}$ for $0 \le j < n_{k-1}$ and $1 \le k < m$, the index of "aunts", as illustrated by Fig. 2. Just as Fig. 1 illustrates a rootless segment tree in the upward phase, Fig. 2 illustrates the corresponding rootless complementary segment tree in the downward phase.

The downward phase computes complementary segment products $\bar{L}_k[j]$ for $j = 0, 1 \ldots, n_k - 1$ and $k = m - 1, m - 2, \ldots, 0$.

```
The downward phase

$\bar{L}_{m-1}[0] = L_{m-1}[0]$;

$\bar{L}_{m-1}[1] = L_{m-1}[1]$;

for ($k = m-1$; $k > 0$; $k--$) do

    for ($j = 0$; $j < 2\rho(n_{k-1})$; $j++$) do

        $\bar{L}_{k-1}[j] = L_{k-1}[j] \circ \bar{L}_k[\alpha_k(j)]$;

    if ($n_{k-1}$ is odd) $\bar{L}_{k-1}[n_{k-1}-1] = \bar{L}_k[\alpha_k(n_{k-1}-1)]$;
```

Comments

(1) If $n = n^* = 2^m$ is a binary power, $\rho(n_{k-1}) = n_k = 2^{m-k}$, $\alpha_k(j) = \tau\rho(j)$, and the final line in the downward phase can be omitted. (2) The downward phase can be modified in the obvious fashion to permit in-place calculation of $\bar{L}_{k-1}[j]$ from $L_{k-1}[j]$, reducing total memory allocation by about 2.

As Appendix A proves, the final array $\bar{L}_0$ has elements $\bar{L}_0[\tau(j)] = \bar{g}_j$ $(0 \le j < 2\rho(n))$, with an additional final element $\bar{L}_0[n-1] = \bar{g}_{n-1}$ if $n_0 = n$ is odd, so the Jackknife Product algorithm ends with a straightforward transposition phase.

```
Transposition phase

for ($j = 0$; $j < 2\rho(n)$; $j++$) do

    $\bar{g}_j = \bar{L}_0[\tau(j)]$;

if ($n$ is odd) $\bar{g}_{n-1} = \bar{L}_0[n-1]$;
```

Comments

The transposition phase can permit an in-place calculation of $\{\bar{g}_j\}$ to overwrite $\bar{L}_0$.

Computational time and storage

Note $n_j = \lceil n2^{-j} \rceil$, so $0 \le n_j - n2^{-j} < 1$. To compute L_k from L_{k-1} or to compute $\bar{L}_k$ from L_k and $\bar{L}_{k+1}$, the Jackknife Product algorithm requires n_k products. For large n, therefore, the upward phase computing the segment tree requires about $\sum_{j=1}^{m-1} n_j \approx \sum_{j=1}^{\infty} n2^{-j} = n$ products; the downward phase, about $\sum_{j=0}^{m-2} n_j \approx 2n$ products. Likewise, if the downward and transposition phases compute in place by replacing L_k with $\bar{L}_k$ and $\bar{L}_0$ with $\{\bar{g}_j\}$, the algorithm storage is $\sum_{j=0}^{m-1} n_j \approx 2n$ semigroup elements. Each of the three estimates just given for products and storage have an error bounded by $m = \lceil \log_2 n \rceil$. Although the case of general n could be handled by the algorithm for binary powers $n^* = 2^m$ by setting $m = \lceil \log_2 n \rceil$ and $g_n = g_{n+1} = \ldots = g_{n^*-1} = e$, the truncated arrays in the Jackknife Product algorithm for general n save about a factor of $1 \le n^*/n < 2$ in both products and storage.

As written, the conditional copy statements at the end of the upward and downward phases replicate elements already in storage. If the downward phase of the Jackknife Product algorithm is implemented with in-place computation of $\bar{L}_{k-1}[j]$ from $L_{k-1}[j]$, the copy statements ensure that the algorithm never overwrites any array element it needs later. Some statements may copy some elements more than once (and therefore unnecessarily), but a negligible $m = \lceil \log_2 n \rceil$ copies at most are unnecessary.

The complementary segment tree in Fig. 2 implicitly indicates the nodes in the segment tree required to compute $L_0[\tau(j)] = \bar{g}_j$ for each $\bar{g}_j$, i.e., exactly one node in each row L_k $(k = 0, 1, \ldots, m-1)$. Alone, the segment tree therefore requires at least $n \log_2 n$ multiplications to compute $\{\bar{g}_j\}$.

Results

Appendix B gives the combinatorics relevant to the circRNA-miRNA application described in "Background" section. As is typical in combinatorial probability, the quantities $\{g_i[k]\}$ were counts of configurations, here, the ways of placing miRNA motifs on circRNAs. The length of each motif was $m = 7$; the largest circRNA (hsa-circ-0003473) contained $I = 158{,}565$ nt, and the most abundant motif (CCCAGCU, for the m12-9star miRNA family) appeared $K = 997$ times, so the $\{g_i[k]\}$ spanned thousands of orders of magnitude in Eq. (13) of Appendix B, from $g_i[0] = 1$ to $g_I[K] \approx 10^{2608}$. In Eq. (1), the dimension K controls the number of terms in the convolutions. In the application, over each miRNA motif examined, the maximum number of motif occurrences on the circRNAs was $K = 997$. An Intel Core i7-3770 CPU computed the p-value relevant to the biological application on June 17, 2015. To compare later with estimated times for competing algorithms, the Jackknife Product algorithm with $n = 3086$ computed the relevant p-values in about 45 min, requiring about $3n$ products. In the application, therefore, n products required about 15 min.

The application of this article to circRNA–miRNA data appears elsewhere [1].

Discussion

This article has presented a Jackknife Product algorithm, which applies to any commutative semi-group $(G, \circ)$. The biological application to a circRNA–miRNA system exemplifies a general statistical method in combinatorial probability. In turn, the application in combinatorial probability exemplifies an even more general statistical test for whether a term in a sum of independent counting variates (not necessarily identically distributed) is unusually large.

Many biological contexts lead naturally to sums of independent counting variates. Domain alignments of proteins from cancer patients, e.g., display point mutations in their columns. For a given domain, a column with an excess of mutations might be inferred to cause cancer [9]. The Background section gives the pattern: let X_i represent the mutation count in column $i = 0, 1, \ldots, n-1$, with total mutations $S = \sum_{i=0}^{n-1} X_i$. Given observed mutation counts $\{X_i = k(i) : i = 0, 1, \ldots, n-1\}$ with observed sum $K = \sum_{i=0}^{n-1} k(i)$, the conditional p-values $\mathbb{P}\{X_i \geq k(i) | S = K\}$ $(i = 0, 1, \ldots, n-1)$ can decide the question: "Does any column have an excess of mutations?" The actual application used other, very different statistical methods [9]. Unlike those methods, however, our methods can incorporate information from control (non-cancer) protein sequences to set column-specific background distributions for $\{X_i\}$.

The Benjamini–Hochberg procedure for controlling the false discovery rate in multiple tests requires either independent p-values [10] or dependent p-values with a positive regression dependency property [11]. Loosely, the positive regression dependency property means that the p-values tend to be small together, i.e., under the null hypothesis, given that one p-value is small, then the other p-values tend to be smaller also. Inconveniently, our null hypothesis posits a fixed sum of independent counting variates, so if one variate is large and has a small p-value, it tends to reduce the other variates and increase their p-values. The circRNA-miRNA application therefore violates the statistical hypotheses of the Benjamini–Hochberg procedure. Fortunately, in the circRNA-miRNA application, a Bonferroni multiple test correction [12] sufficed because empirically, any p-value was either close to 1 or extremely small.

The Results state that for $n = 3086$, the Jackknifed Product algorithm computed the relevant p-values in about 45 min, with n products requiring about 15 min of computation. In contrast, the naïve algorithm avoiding inverses and requiring $n(n-2)$ products would have taken about $3086 * 15$ min, i.e., about 1 month. As explained under the "Computational Time and Storage" heading in the Theory section, without exploiting the special form of the jackknife products $\{\bar{g}_j\}$, a segment tree requires about n products for its construction and at least $n \log_2 n$ products for the computation of the products $\{\bar{g}_j\}$. Alone, segment tree algorithms would therefore have taken a minimum of about $(1 + \log_2 3086) * 15$ min, i.e., about 3 h.

The convolutions in Eq. (1) might suggest that jackknife products are susceptible to computation with Fourier or Laplace transforms, which convert convolutions into products. "Results" section notes that in the biological application, however, $\{g_i[k]\}$ in Eq. (1) spanned thousands of orders of magnitude, at least from $g_i[0] = 1$ to $g_I[K] \approx 10^{2608}$, obstructing the direct use of transforms (e.g., see [6]). On one hand, the widely varying magnitudes necessitated an internal logarithmic representation of $\{g_i[k]\}$ in the computer, a minor inconvenience for direct computation with the Jackknife Product algorithm. On the other hand, they might have presented a substantial obstacle for transforms. The famous Feynman anecdote about Paul Olum's tan (10^{100}) problem indicates the reason [13]:

So Paul is walking past the lunch place and these guys are all excited. "Hey, Paul!" they call out. "Feynman's terrific! We give him a problem that can be stated in ten seconds, and in a minute he gets the answer to 10 percent. Why don't you give him one?" Without hardly stopping, he says, "The tangent of 10 to the 100th." I was sunk: you have to divide by pi to 100 decimal places! It was hopeless.

The Jackknife Product algorithm also abstracts to any commutative semigroup $(G, \circ)$, broadening its applicability enormously. As usual, abstraction eases debugging. Consider, e.g., the commutative semigroup consisting of all bit strings of length n under the bitwise "or" operation. If the bit string g_j has 1 in the j-th position and 0 s elsewhere, then the segment product $g_{[i,j)}$ equals the bit string with 1 s in positions $[i, j) = \{i, i+1, \ldots, j-1\}$ and 0 s elsewhere. Similarly, the complementary segment product $g_{\overline{[i,j)}} = g_{[0,i)} g_{[j,n)}$ equals the bit string with 0 s in positions $[i, j) = \{i, i+1, \ldots, j-1\}$ and 1 s elsewhere. The Jackknife Product algorithm is easily debugged with output consisting of the segment and complementary segment trees for the bit strings.

As a final note, even if a semigroup $(G, \circ)$ lacks the Commutative property, the general product algorithm for a segment tree can still compute $\{\bar{g}_j = g_{[0,j)} g_{[j+1,n)}\}$ in time $O(n \log n)$. In a commutative semigroup $(G, \circ)$, however, the downward phase of the Jackknife Product algorithm exploits the special form of the products $\{\bar{g}_j\}$ to decrease the time to $O(n)$.

Conclusions

This article has presented a Jackknife Product algorithm, which applies to any commutative semi-group $(G, \circ)$. The biological application to a circRNA–miRNA system uses a commutative semigroup of truncated convolutions to exemplify a specific application to combinatorial probabilities. In turn, the specific application in combinatorial probability exemplifies an even more general statistical test for whether a term in a sum of independent counting variates (not necessarily identically distributed) is unusually large. The general statistical test can evaluate the results of searching for a sequence or structure motif, or several motifs simultaneously. As "Discussion" section explains, the test violates the hypotheses of the Benjamini–Hochberg procedure for estimating false discovery rates, but fortunately the Bonferroni and other multiple-test corrections remain available to control familywise errors. Abstraction from convolutions to commutative semi-groups broadens the algorithm's applicability even further. If an application only requires jackknife products $\{\bar{g}_j\}$ and their number n is large enough, "Results" and "Theory" sections show that the linear time of the Jackknife Product algorithm can make it well worth the programming effort.

Acknowledgements
JLS would like to acknowledge useful conversations with Dr. Amir Manzour and Dr. DoHwan Park.

Authors' contributions
JLS developed the algorithms, performed computational and statistical analysis, and drafted the manuscript. JMZ provided the data stimulating and exemplifying the analysis. MG performed bioinformatics and sequence analysis. All authors read and approved the final manuscript.

Funding
This research was supported by the Intramural Research Program of the NIH, National Library of Medicine, and by the Center for Cancer Research, National Cancer Institute.

Availability of data and materials
A self-testing, annotated file "jls_jackknifeproduct.py" implementing an in-place Jackknife Product algorithm in Python is available without any restrictions at https://github.com/johnlspouge/jackknife-product. Data were previously published elsewhere [1].

Ethics approval and consent to participate
Not applicable.

Consent for publication
Not applicable.

Competing interests
The authors declare that they have no competing interests.

Author details
[1] National Center for Biotechnology Information, National Library of Medicine, National Institutes of Health, Room 6N603, Building 38A, Bethesda, MD 20894, USA. [2] HIV and AIDS Malignancy Branch, Center for Cancer Research, National Cancer Institute, National Institutes of Health, Bethesda, MD 20892, USA. [3] Genomics Research and Development, Lenovo HPC and AI, 1009 Think Pl, Morrisville, NC 27560, USA.

Appendix A

This appendix proves the correctness of the Jackknife Product algorithm in "Theory" section.

Let L_k have length l_k; $\bar{L}_k$, length $\bar{l}_k$. Some observations about l_k, $\bar{l}_k$, and n_k facilitate later analysis. Because $\lceil \lceil x \rceil /2 \rceil = \lceil x/2 \rceil$, $\{n_k\}$ satisfies the recursion $n_k = \lceil n2^{-k} \rceil = \lceil n2^{-(k-1)}/2 \rceil = \lceil n_{k-1}/2 \rceil$, with initial value $n_0 = n$ and final value $n_{m-1} = 2$.

Proposition 1 $l_k = \bar{l}_k = n_k$ for $0 \le k < m$.

Proof (by induction)

$l_0 = n = n_0$. If $l_{k-1} = n_{k-1}$ for any $1 \le k < m$, the upward phase of the Jackknife Product algorithm shows: (1) if n_{k-1} is even, $l_k = \rho(n_{k-1})$; and (2) if n_{k-1} is odd, $l_k = \rho(n_{k-1}) + 1$. In either case, $l_k = \lceil n_{k-1}/2 \rceil = n_k$. Thus, $l_k = n_k$ for $0 \le k < m$.

Similarly, the downward phase shows: (1) if n_{k-1} is even, $\bar{l}_{k-1} = 2\rho(n_{k-1}) = n_{k-1}$; if is odd, $\bar{l}_{k-1} = 2\rho(n_{k-1}) + 1 = n_{k-1}$. It therefore initializes $\bar{L}_{m-1}$ with $\bar{l}_{m-1} = n_{m-1} = 2$ elements and assigns $\bar{l}_{k-1} = n_{k-1}$ $(1 \le k < m)$ elements to $\bar{L}_{k-1}$. $\square$

Proposition 1 and its proof ensure that with the possible exception of $\alpha_k(j)$ in the downward phase, all array indices in the Jackknife Product algorithm lie within the array bounds of L_k and $\bar{L}_k$. Moreover, case-by-case analysis of the definition of α_k shows that $\alpha_k(j)$ $(0 \le j < n_{k-1})$ always falls within the array bounds $0 \le \alpha_k(j) < n_k$ of $\bar{L}_k$. Inspection of the upward and downward phases shows that they define every array element before using it. With array bounds and definitions in hand, to verify the Jackknife Product algorithm, it therefore suffices to check conditions satisfied by individual elements of L_k and $\bar{L}_k$. We examine first the case of binary powers $n = n^* = 2^m$, and afterwards the case of general n.

Proof of Correctness for Binary Powers $n^* = 2^m \ge 2$

In this subsection, some entities pertaining to binary powers n^* receive stars (e.g., n^*, n_k^*, L_k^*, $\bar{L}_k^*$), to distinguish them later from the corresponding entities for general n.

For convenience in Appendix A only, drop "g" in the notation $g_{[i,j)}$, and abbreviate the segment product $g_{[i,j)} = g_i g_{i+1} \cdots g_{j-1}$ by the corresponding half-open interval $[i,j)$ and the complementary segment product $g_0 g_1 \cdots g_{i-1} g_j \cdots g_{n^*-1}$ by $\overline{[i,j)}$. The notation provides a mnemonic aid, used without comment below: for $i < j < k$, $[i,j) \circ [j,k) = [i,k)$, $[i,j) \circ \overline{[i,k)} = \overline{[j,k)}$ and $[j,k) \circ \overline{[i,k)} = \overline{[i,j)}$, i.e., the product $\circ$ on sub-products behaves like a set-theoretic union of the corresponding intervals, and complementary sub-products behave like the corresponding set-theoretic complements of intervals.

The Commutative property is required to justify the correspondence between set-theoretic operations and products, e.g., the equality $[i,j) \circ \overline{[i,k)} = \overline{[j,k)}$ commutes the segment products: examine, e.g., the second equality in the equation

$$\begin{aligned} g_{[i,j)} \circ \left(g_{[0,i)} \circ g_{[k,n*)}\right) &= \left(g_{[i,j)} \circ g_{[0,i)}\right) \circ g_{[k,n*)} \\ &= \left(g_{[0,i)} \circ g_{[i,j)}\right) \circ g_{[k,n*)} = g_{[0,j)} \circ g_{[k,n*)}. \end{aligned} \tag{2}$$

Proposition 2 For $0 \le k < m$, $L_k^*[j] = [j \times 2^k, (j+1) \times 2^k)$ $(0 \le j < n_k^*)$.

Proof See Fig. 1. The array L_0^* at generation $k = 0$ initializes the upward phase, where

$$L_0^*[j] = g_j = \left[j \times 2^0, (j+1) \times 2^0\right) \quad \text{for} \quad 0 \le j < n^*. \tag{3}$$

Thus, Proposition 2 is true for $k = 0$. Given the array L_{k-1}^* at generation $k-1$ in the upward phase, the array L_k^* at level k contains products of successive adjacent pairs of elements in L_{k-1}^*:

$$\begin{aligned} L_k^*[j] &= L_{k-1}^*[2j] \circ L_{k-1}^*[2j+1] \\ &= \left[2j \times 2^{k-1}, (2j+1) \times 2^{k-1}\right) \circ \\ &\quad \left[(2j+1) \times 2^{k-1}, (2j+2) \times 2^{k-1}\right) \\ &= \left[j \times 2^k, (j+1) \times 2^k\right) \end{aligned} \tag{4}$$

for $0 \le j < n_k^* = n_{k-1}^*/2$. The upward phase terminates with $L_{m-1}^* = \left([0, 2^{m-1}), [2^{m-1}, 2^m)\right)$, so Proposition 2 is true for $0 \le k < m$. □

Proposition 3 For $0 \le k < m$, $\bar{L}_k^*[j] = \overline{[\tau(j) \times 2^k, (\tau(j)+1) \times 2^k)}$ $(0 \le j < n_k^*)$.

Comment

Propositions 3 and 2 formalize the previously mentioned complementary symmetry between the upward and downward phases. Because $\tau(\tau(j)) = j$ (i.e., transposition is idempotent), $\bar{L}_0^*[\tau(j)] = \overline{[j, j+1)} = \bar{g}_j$ for $0 \le j < n_0^* = n^*$. Thus, $\bar{L}_0^*$ contains all jackknife products.

Proof See Fig. 2. For $0 \le j < n_{m-1}^* = 2$, the first two lines of pseudo-code in the downward phase and Proposition 2 for $k = m-1$ show that for $j \in \{0, 1\}$,

$$\begin{aligned} \bar{L}_{m-1}^*[j] = L_{m-1}^*[j] &= \left[j \times 2^{m-1}, (j+1) \times 2^{m-1}\right) \\ &= \overline{\left[\tau(j) \times 2^{m-1}, (\tau(j)+1) \times 2^{m-1}\right)}, \end{aligned} \tag{5}$$

so Proposition 3 holds for $k = m-1$. We proceed by descending induction on k.

For even j on one hand, $2\rho(j) = j < j+1 = \tau(j) < \tau(j)+1 = 2(\rho(j)+1)$. For $0 \le j < n_{k-1}^*$, therefore,

$$\begin{aligned} &\left[j \times 2^{k-1}, (j+1) \times 2^{k-1}\right) \circ \left[\tau(j) \times 2^{k-1}, (\tau(j)+1) \times 2^{k-1}\right) \\ &\quad = \left[\rho(j) \times 2^k, (\rho(j)+1) \times 2^k\right). \end{aligned} \tag{6}$$

For odd j on the other hand, $2\rho(j) = \tau(j) < \tau(j)+1 = j < j+1 = 2(\rho(j)+1)$, so Eq. (6) holds with the factors on the left reversed, an irrelevant difference in a commutative semigroup $(G, \circ)$.

For $0 \le j < n_{k-1}^*$, then, if $1 \le k < m$, Proposition 2 for $k-1$ and Proposition 3 for k yield

$$\begin{aligned} \bar{L}_{k-1}^*[j] &= L_{k-1}^*[j] \circ \bar{L}_k^*[\tau\rho(j)] \\ &= \left[j \times 2^{k-1}, (j+1) \times 2^{k-1}\right) \circ \\ &\quad \overline{\left[\rho(j) \times 2^k, (\rho(j)+1) \times 2^k\right)} \\ &= \overline{\left[\tau(j) \times 2^{k-1}, (\tau(j)+1) \times 2^{k-1}\right)}. \end{aligned} \tag{7}$$

Thus, Proposition 3 for k implies Proposition 3 for $k-1$ $(1 \le k < m)$. □

The Jackknife Product algorithm therefore computes $\bar{L}_0^*[\tau(j)] = \overline{[j, j+1)} = \bar{g}_j$, as desired.

Proof of correctness for general $n \ge 2$: For general $n \ge 2$, initialize the upward phase with $L_0 = (g_0, g_1, \ldots, g_{n-1})$. To apply the results of the previous subsection, let $n^* = 2^m$ be the smallest binary power greater than or equal to n, i.e., let $m = \lceil \log_2 n \rceil$. If $n < n^*$, set $g_n = g_{n+1} = \cdots = g_{n^*-1} = e$, with the arrays L_k^* and $\bar{L}_k^*$ of length n_k^* $(0 \le k < m)$ as above. For general $n \ge 2$, consider the computation of the arrays L_k $(0 < k < m)$ in the upward phase of the pseudocode above.

We prove Proposition $P(k)$ (next) by induction on the level $0 \le k < m$.

Proposition $P(k)$ $L_k[j] = L_k^*[j]$ for $0 \le j < n_k$, and $L_k^*[j] = e$ for $n_k \le j < n_k^*$.

Proof See Fig. 1. By construction, $P(0)$ holds. With $P(k-1)$ in hand, the upward phase and Eq. (4) show that $L_k[j] = L_k^*[j]$ for $0 \le j < \rho(n_{k-1})$. On one hand, if n_{k-1} is even, $\rho(n_{k-1}) = n_k$, yielding $P(k-1)$

immediately. On the other hand, if n_{k-1} is odd, $\rho(n_{k-1}) = \lceil n_{k-1}/2 \rceil - 1 = n_k - 1$, so

$$\begin{aligned} L_k[n_k - 1] &= L_k\left[\rho\left(n_{k-1}\right)\right] = L_{k-1}[n_{k-1} - 1] \\ &= L^*_{k-1}[n_{k-1} - 1] \\ &= L^*_{k-1}[n_{k-1} - 1] \circ e = L^*_{k-1}[n_{k-1} - 1] \circ \\ &\quad L^*_{k-1}[n_{k-1}] = L^*_k[n_k - 1], \end{aligned} \tag{8}$$

the second equality reflecting the copy of the final element of L_{k-1} in the pseudocode; the third and fifth, $P(k-1)$; and the sixth, Eq. (4). Equation (8) completes the proof that $L_k[j] = L^*_k[j]$ for $0 \le j < n_k$. For the remaining indices $n_k \le j < n^*_k$ of L^*_k, note that $n_{k-1} \le 2\lceil n_{k-1}/2 \rceil = 2n_k \le 2j < 2j+1 < 2n^*_k = n^*_{k-1}$. Then, $P(k-1)$ and Eq. (4) show that $L^*_k[j] = L^*_{k-1}[2j] \circ L^*_{k-1}[2j+1] = e \circ e = e$ for $n_k \le j < n^*_k$. □

Note: $n_{m-1} = 2 = n^*_{m-1}$, so $P(m-1)$ shows that $L_{m-1} = L^*_{m-1}$.

In the downward phase, the transposition function τ in Eq. (7) facilitates in-place computation for $\bar{L}^*_{k-1}[j]$ in Eq. (7). Similarly, the minimization in the accessory index $\alpha_k(j) = \min\{\tau\rho(j), n_k - 1\}$ within $\bar{L}_k$ avoids storing a superfluous element $\bar{g}$ of $\bar{L}^*_k$ within the penultimate element of any truncated complementary array $\bar{L}_k$ (see $\bar{L}^*_1[3]$, the dotted circle in Fig. 2).

We prove Proposition $\bar{P}(k)$ (next) by descending induction on the level $0 \le k < m$.

Proposition $\bar{P}(k)$ $\bar{L}_k[j] = \bar{L}^*_k[j]$ for $0 \le j < n_k$, unless n_k is odd and $j = n_k - 1$, in which case $\bar{L}_k[n_k - 1] = \bar{L}^*_k[n_k]$.

Proof The proposition $\bar{P}(m-1)$ is true, because $\bar{L}_{m-1} = L_{m-1} = L^*_{m-1} = \bar{L}^*_{m-1}$, and $\bar{L}_{m-1}$ has even length $n_{m-1} = 2$. We now show that $\bar{P}(k)$ implies $\bar{P}(k-1)$ for $0 < k < m$.

If $0 \le j < n_{k-1}$, then $0 \le \rho(j) < \lceil n_{k-1}/2 \rceil = n_k$. For every j, either: (1) n_k is odd and $\rho(j) = n_k - 1$; (2) n_k is odd and $\rho(j) < n_k - 1$; or (3) n_k is even. In Case 1, $\tau\rho(j) = n_k$ but $\alpha_k(j) = n_k - 1$. Because n_k is odd, $\bar{P}(k)$ implies $\bar{L}_k\left[\alpha_k(j)\right] = \bar{L}_k[n_k - 1] = \bar{L}^*_k[n_k] = \bar{L}^*_k\left[\tau\rho(j)\right]$. In Case 2, $0 \le \alpha_k(j) = \tau\rho(j) < n_k - 1$, or in Case 3, $0 \le \alpha_k(j) = \tau\rho(j) < n_k$ and n_k is even. In either case, $\bar{P}(k)$ implies $\bar{L}_k\left[\alpha_k(j)\right] = \bar{L}_k\left[\tau\rho(j)\right] = \bar{L}^*_k\left[\tau\rho(j)\right]$. Thus, regardless of whether Case 1, 2, or 3 pertains, $\bar{L}_k\left[\alpha_k(j)\right] = \bar{L}^*_k\left[\tau\rho(j)\right]$ for every $0 \le j < n_{k-1}$.

For $0 \le j < 2\rho(n_{k-1})$, the Jackknife Product algorithm in the downward phase and $P(k-1)$ from the upward phase yield

$$\bar{L}_{k-1}[j] = L_{k-1}[j] \circ \bar{L}_k\left[\alpha_k(j)\right] = \bar{L}^*_{k-1}[j] \circ \bar{L}^*_k\left[\tau\rho(j)\right] = \bar{L}^*_{k-1}[j] \tag{9}$$

if $j < n_{k-1}$. On one hand, if n_{k-1} is even, $n_{k-1} = 2\rho(n_{k-1})$, yielding $\bar{P}(k-1)$ immediately. On the other hand, if n_{k-1} is odd, $n_{k-1} = 2\rho(n_{k-1}) + 1$, so $\bar{L}_{k-1}$ has an additional, final element copied from $\bar{L}_k$:

$$\bar{L}_{k-1}[n_{k-1} - 1] = \bar{L}_k\left[\alpha_k\left(n_{k-1} - 1\right)\right] = \bar{L}^*_k\left[\tau\rho\left(n_{k-1} - 1\right)\right]. \tag{10}$$

$P(k-1)$ yields $L^*_{k-1}[n_{k-1}] = e$, so moreover,

$$\bar{L}^*_{k-1}[n_{k-1}] = L^*_{k-1}[n_{k-1}] \circ \bar{L}^*_k\left[\tau\rho\left(n_{k-1}\right)\right] = \bar{L}^*_k\left[\tau\rho\left(n_{k-1}\right)\right]. \tag{11}$$

Because $n_{k-1} - 1$ is even, $\rho\left(n_{k-1} - 1\right) = \rho\left(n_{k-1}\right)$. Equations (10) and (11) therefore yield $\bar{L}_{k-1}[n_{k-1} - 1] = \bar{L}^*_{k-1}[n_{k-1}]$, so $\bar{P}(k-1)$ holds. □

Appendix B

This appendix gives combinatoric calculations for the circRNA-miRNA application in "Background" section. It therefore has some peripheral interest to this article.

Motifs on a single circle

Consider r points equally spaced around a circle (a "ring"). Call a set of m consecutive points on the ring a "motif". The following fixes m, so the notation can leave it implicit. Let $C_{r,k}$ count the ways of choosing k non-overlapping motifs around the ring (i.e., the motifs have no point in common). Note: $C_{r,k} = 0$ if $r < mk$ or $k < 0$. Define the factorial function $n! = n(n-1)\cdots 1$ and the binomial (combinatorial or Pascal) coefficient

$$\binom{n}{k} = \frac{n!}{k!(n-k)!} \tag{12}$$

for $0 \le k \le n$ and 0 otherwise.

Theorem Clearly, $C_{r,k} = 1$ when $r = mk$. For $r > mk$,

$$\begin{aligned} C_{r,k} &= \binom{r - 1 - (m-1)k}{k} + m\binom{r - m - (m-1)(k-1)}{k-1} \\ &= \binom{r - (m-1)k - 1}{k} + m\binom{r - (m-1)k - 1}{k-1} \end{aligned} \tag{13}$$

counts the ways of placing k motifs around the ring. For convenience below and for consistency with Eq. (13), $C_{r,0} = 1$ for $r \ge 0$.

Proof Consider a line segment containing r equally spaced points, and let $L_{r,k}$ count the ways of choosing k non-overlapping motifs, each of m consecutive points, on it. First, $C_{r,k} = L_{r-1,k} + mL_{r-m,k-1}$, proved as follows.

Number the ring points arbitrarily as positions $1, 2, \ldots, r$, and place the k motifs as follows. Consider position 1, which might have no motif. If so, place the k motifs in a line consisting of $r-1$ positions $2, 3, \ldots, r$ ($L_{r-1,k}$ ways). Otherwise, place the first motif in one of the m positions in which it covers position 1, and then place the remaining $k-1$ motifs in a line consisting of $r-m$ positions ($mL_{r-m,k-1}$ ways). Each such configuration corresponds to placing k leftmost end-positions on the line. For each of the k motifs, delete $m-1$ positions on the ring, all but its leftmost end-position. Each of the configurations for k motifs therefore corresponds to choosing k positions from the $r-(m-1)k$ positions remaining, yielding the remaining factors in Eq. (13).

Motifs on several circles

Now, let $C_{r(1),r(2),\ldots,r(N);K}$ count the ways of distributing K non-overlapping motifs (all of m consecutive points) around several rings, the rings' points numbering $r(n)$ ($n = 1, 2 \ldots, N$). Without loss of generality, assume $r(n) \geq m$ ($n = 1, 2 \ldots, N$). (Otherwise, discard all rings with $r(n) < m$.) The following recursion holds: $C_{r(1),r(2),\ldots,r(N);K} = 0$ for $K < 0$ or $K > \sum_{n=1}^{N} \lfloor r(n)/m \rfloor$, $C_{r(1),r(2),\ldots,r(N);0} = 1$, and otherwise

$$C_{r(1),r(2),\ldots,r(N);K} = \sum C_{r(N),k} C_{r(1),r(2),\ldots,r(N-1);K-k}, \tag{14}$$

where on the right, the index of summation k runs from $\max\left\{0, K - \sum_{n=1}^{N-1} \lfloor r(n)/m \rfloor\right\}$ up to $\min\{K, \lfloor r(N)/m \rfloor\}$. Equation (13) initializes the convolution recursion in Eq. (14) with $C_{r(1),K}$.

Relation to the Jackknife Product algorithm

To apply the Jackknife Product algorithm in the circRNA–miRNA application, let $g_i = \left(C_{r(i),0}, C_{r(i),1}, \ldots, C_{r(i),K}\right)$ ($i = 1, \ldots, N$) in the (commutative) semigroup $(G, \circ)$ of non-negative integer vectors with indices $k = 0, 1, \ldots, K$, under the convolution operation.

Received: 20 March 2020 Accepted: 8 September 2020
Published online: 19 September 2020

References

1. Tagawa T, Gao SJ, Koparde VN, Gonzalez M, Spouge JL, Serquina AP, Lurain K, Ramaswami R, Uldrick TS, Yarchoan R, et al. Discovery of Kaposi's sarcoma herpesvirus-encoded circular RNAs and a human antiviral circular RNA. Proc Natl Acad Sci USA. 2018;115(50):12805–10.
2. Siegel S. Nonparametric statistics for the behavioral sciences. 1st ed. New York: MacGraw-Hill; 1956.
3. Freeman GH, Halton JH. Note on an exact treatment of contingency, goodness of fit and other problems of significance. Biometrika. 1951;38(1–2):141–9.
4. Fisher Exact Test Batch Processing. https://tinyurl.com/spouge-fisher.
5. Efron B, Stein C. The Jackknife estimate of variance. Ann Stat. 1981;9(3):586–96.
6. Nagarajan N, Keich U. FAST: Fourier transform based algorithms for significance testing of ungapped multiple alignments. Bioinformatics. 2008;24(4):577–8.
7. Artin M. Algebra. Eaglewood Cliffs: Prentice-Hall; 1991.
8. Laaksonen A. Competitive Programmer's Handbook; 2017.
9. Peterson TA, Gauran IIM, Park J, Park D, Kann MG. Oncodomains: a protein domain-centric framework for analyzing rare variants in tumor samples. PLoS Comput Biol. 2017;13(4):e1005428.
10. Benjamini Y, Hochberg Y. Controlling the false discovery rate—a practical and powerful approach to multiple testing. J R Stat Soc Ser B-Methodol. 1995;57(1):289–300.
11. Benjamini Y, Yekutieli D. The control of the false discovery rate in multiple testing under dependency. Ann Stat. 2001;29(4):1165–88.
12. Hochberg Y. A sharper Bonferroni procedure for multiple tests of significance. Biometrika. 1988;75(4):800–2.
13. Feynman R, Leighton R, Hutchings E. Surely your're joking, Mr Feynman!. New York: W.W. Norton & Company; 1985.

Publisher's Note

Springer Nature remains neutral with regard to jurisdictional claims in published maps and institutional affiliations.