A Dyadic Simulation Approach to Efficient Range-Summability

Jingfan Meng
School of Computer Science, Georgia Institute of Technology, Atlanta, GA, USA

Huayi Wang
School of Computer Science, Georgia Institute of Technology, Atlanta, GA, USA

Jun Xu
School of Computer Science, Georgia Institute of Technology, Atlanta, GA, USA

Mitsunori Ogihara
Department of Computer Science, University of Miami, Coral Gables, FL, USA

Abstract

Efficient range-summability (ERS) of a long list of random variables is a fundamental algorithmic problem that has applications to three important database applications, namely, data stream processing, space-efficient histogram maintenance (SEHM), and approximate nearest neighbor searches (ANNS). In this work, we propose a novel dyadic simulation framework and develop three novel ERS solutions, namely Gaussian-dyadic simulation tree (DST), Cauchy-DST and Random Walk-DST, using it. We also propose novel rejection sampling techniques to make these solutions computationally efficient. Furthermore, we develop a novel k-wise independence theory that allows our ERS solutions to have both high computational efficiencies and strong provable independence guarantees.

2012 ACM Subject Classification Theory of computation → Streaming, sublinear and near linear time algorithms; Mathematics of computing → Random number generation

Keywords and phrases fast range-summation, locality-sensitive hashing, rejection sampling

Digital Object Identifier 10.4230/LIPIcs.ICDT.2022.17

Related Version *Previous Version*: https://arxiv.org/abs/2109.06366

Funding This material is based upon work supported by the National Science Foundation under Grant No. CNS-1909048, CNS-2007006, CNS-2051800, and by Keysight Technologies under Grant No. BG005054.

1 Introduction

In this work, we propose *dyadic simulation*, a novel solution framework to a fundamental algorithmic problem that has applications to three important database applications: data stream processing [8], space-efficient histogram maintenance (SEHM) [7, 11] and approximate nearest neighbor searches (ANNS) [24]. This algorithmic problem, called *efficient range-summability* (ERS) of random variables (RVs) [5, 17], can be stated as follows. Let $X_0, X_1, \cdots, X_{U-1}$ be a list of i.i.d. RVs, where the (index) universe size U is typically a large number (say $U = 2^{64}$). Given a range $[a, b) \triangleq \{a, a+1, \cdots, b-1\}$ that lies in $[0, U)$, we need to compute $S[a, b) \triangleq \sum_{i=a}^{b-1} X_i$, the sum of the RVs $X_a, X_{a+1}, \cdots, X_{b-1}$ in the range. A straightforward but naive solution to this problem, which follows an intuitive "bottom-up" approach, is to generate RVs $X_a, X_{a+1}, \cdots, X_{b-1}$ individually and then add them up. This solution, however, has a time complexity of $O(b - a)$, which is inefficient computationally when the range length $b - a$ is large. In contrast, an efficient solution should be able to do so with only $O(\text{polylog}(b - a))$ time complexity. Indeed, all existing ERS solutions [2, 5, 17, 7] have $O(\log(b - a))$ time complexity.

25th International Conference on Database Theory (ICDT 2022).
Editors: Dan Olteanu and Nils Vortmeier; Article No. 17; pp. 17:1–17:18
Leibniz International Proceedings in Informatics
LIPICS Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

Each such ERS solution generates a range-sum $S[a,b)$ or an underlying RV X_i in a very different way than the naive solution. This difference however does not matter since an ERS solution is considered correct as long as it satisfies two requirements: consistency and correct distribution. The consistency requirement is that, given any *outcome* ω in the *sample space* Ω (in probability theory terms), the realization of $S[a,b)$ associated with the outcome ω must be equal to $\sum_{i=a}^{b-1} X_i(\omega)$, where $X_i(\omega)$ is the realization of X_i associated with the same outcome ω. The correct distribution requirement is that the underlying RVs thus generated are ideally i.i.d. with distribution X.

1.1 Our Dyadic Simulation Approach

For ease of presentation, we make two harmless simplifying assumptions. The first assumption is a typical "computer science" one: The universe size U is a power of 2. This assumption can always be fulfilled by increasing U to at most $2U$. The second assumption is that $[a,b)$ is a dyadic range in the sense there exist integers $j \geq 0$ and $i \geq 0$ such that $a = j \cdot 2^i$ and $b = (j+1) \cdot 2^i$. It suffices for our solution to work for any dyadic range since any non-dyadic range can be split into at most $2\log_2 U$ dyadic ranges, as we will elaborate in Subsection 2.1.

Unlike the naive solution, our dyadic simulation approach computes $S[a,b)$ in a counter-intuitive "top-down" manner as follows. Its first step is to generate the RV $S[0,U)$, the range-sum of the entire universe. If we denote the distribution of each underlying RV X_i as X, then $S[0,U)$ has distribution X^{*U}, where, for any $n > 1$, X^{*n} denotes the n^{th} convolution power of X. When X is one of a few nice distributions, the distribution X^{*U} can be analytically derived and also takes a nice form; in this case, it is straightforward to generate $S[0,U)$. For example, when X is standard Gaussian distribution $\mathcal{N}(0,1)$, then X^{*U} is $\mathcal{N}(0,U)$.

The rest of dyadic simulation proceeds as follows. If $[a,b)$ is the same as $[0,U)$, then the ERS problem is solved. Otherwise, we split $S[0,U)$ into two half-range-sums $S[0,U/2) + S[U/2,U)$, such that RVs $S[0,U/2)$ and $S[U/2,U)$ are (mutually) independent and each has distribution $X^{*(U/2)}$. While this may sound wishful thinking, we will show in Section 2 that it is always mathematically possible and can be done in a computationally efficient manner in some cases.

After the split, we have either $[a,b) \subseteq [0,U/2)$ or $[a,b) \subseteq [U/2,U)$ by Proposition 2.4 in [17], since $[a,b)$, $[0,U/2)$, and $[U/2,U)$ are all dyadic intervals. We then recursively "binary-search" for $[a,b)$ either in the left-half $[0,U/2)$ if $[a,b) \subseteq [0,U/2)$ or in the right-half $[U/2,U)$ if $[a,b) \subseteq [U/2,U)$. It is not hard to verify that after at most $\log_2 U$ such splits we can "find" $[a,b)$ and as a result compute $S[a,b)$. Hence, the time complexity of a dyadic simulation algorithm is $O(\log U)$ splits for generating any dyadic range-sum. Perhaps surprisingly, even for generating any range-sum that is not necessarily dyadic, the time complexity remains $O(\log U)$ splits instead of becoming $O(\log^2 U)$, as we will show in Subsection 2.1.

We can generate any underlying RV X_i via $\log_2 U$ such binary splits, because $X_i \equiv S[i,i+1)$, and $[i,i+1)$ is a dyadic range. We say dyadic simulation takes a "top-down" approach because when all the underlying RVs $X_0, X_1, \cdots, X_{U-1}$ are generated this way, they become the "leaves" (at the "bottom") of the full binary tree of the binary splits involved in generating them. This tree, called *dyadic simulation tree* (DST), will be officially introduced in Subsection 2.1. In this work, we propose novel DST-based solutions to three ERS problems whose underlying RVs have Gaussian, Cauchy, and single-step random walk (RW) (*aka.* Rademacher) distributions, respectively. We also propose novel rejection sampling techniques that make these three solutions, called Gaussian-DST, Cauchy-DST, and RW-DST respectively, computationally efficient. Each binary split operation takes only nanoseconds for Gaussian and 20+ nanoseconds for Cauchy and random walk, as we will show in Subsection 2.6.

All existing ERS solutions were proposed for the single-step random walk distribution $\Pr[X = 1] = \Pr[X = -1] = 0.5$. Here we highlight a key difference between our dyadic simulation approach and these ERS solutions. This difference is a major contribution of this work. The underlying RVs X_0, X_1, $\cdots$, X_{U-1} generated by our dyadic simulation approach are at least empirically independent for all practical purposes. In contrast, those generated by all existing ERS solutions are strongly correlated. For example, in the EH3 scheme proposed in [5, 17], the underlying RVs are approximately 4-wise independent, but all independence beyond 4-wise is completely destroyed. However, in nearly all applications of dyadic simulation that we will describe next, we need these RVs to be at least empirically independent.

A very sketchy idea of dyadic simulation was proposed, in a few sentences, in a theory paper [7] that mainly focused on the aforementioned SEHM problem. Although it was stated in [7] that dyadic simulation can possibly be used for the ERS of Gaussian and Cauchy RVs, no computationally efficient technique was specified in it for binary-splitting a Gaussian or Cauchy RV. We will elaborate on such techniques in Subsections 2.3 and 2.4.

1.2 Independence Guarantees

As we have just explained, each non-leaf node in a DST corresponds to a dyadic range $[a, b)$, whose two children correspond to the two dyadic half ranges $[a, (a+b)/2)$ and $[(a+b)/2, b)$. We will show in Subsection 2.2 that each such non-leaf node, now identified by its corresponding dyadic range say $[a, b)$, is associated with a uniformly random binary string $C_{[a,b)}$ that determines the values of half-range-sums $S[a, (a + b)/2)$ and $S[(a + b)/2, b)$ that the range-sum $S[a, b)$ is split into. Depending on how each $C_{[a,b)}$ is generated, we can obtain various theoretical guarantees concerning how independent the underlying RVs $X_0, X_1, \cdots, X_{U-1}$ are.

Ideally, each such $C_{[a,b)}$ should be a freshly generated RV in the sense that it is independent of all other RVs. If this is the case, then we can prove that, starting with $S[0, U)$ that is distributed as X^{*U}, the U underlying RVs generated through these binary splits are i.i.d. with distribution X. However, this idealized case is impractical when the universe size U is massive, since the value of each freshly generated RV would all have to be remembered (stored in memory) and there can be a massive number of them. In practice, we typically generate each such $C_{[a,b)}$ value (on demand) by applying a hash function $h(\cdot)$ to the dyadic range $[a, b)$. There are two standard choices of such hash functions in the literature. The practical type is "off-the-shelf" random hash functions that can produce a hash value in nanoseconds, such as `wyhash` [25]. Although they provide no theoretical guarantees, they are demonstrated to ensure a level of empirical independence that is good enough for all practical applications [23]. The theoretical type, called k-wise independent hash functions [3, 21, 15], generates $(C_{[a,b)})$'s that are k-wise independent. In this work, we establish a novel k-wise independence theory for DST which shows, among other things, that k-wise independence among $C_{[a,b)}$ values implies k-wise independence among the underlying RVs. Although the latter theoretical guarantee is weaker than the ideal all-wise mutual independence, it leads to rigorous theoretical guarantees that are strong enough for most ERS applications, as we will show in Subsection 1.3.

We note all our DST solutions can use Nisan's pseudorandom generator (PRG) [12], which delivers strong independence guarantees for memory- (state-space-) constrained algorithms. However, Nisan's PRG is quite computationally intensive, and hence has never been implemented and used in practice. Indeed, a key contribution of our k-wise independence theory lies in its ability to satisfy the "theoretical needs" of most ERS applications using k-wise independent hash functions that are much less computationally intensive.

1.3 Applications

In this section, we describe the three aforementioned applications that motivate our DST-based ERS solutions. Since we claim none of them as a contribution of this work, each description here is only detailed enough to explain how an ERS problem arises in it. Furthermore, we will not elaborate on any application in the rest of this paper.

The *first* application is data stream processing, where two of our ERS solutions extend an existing data streaming algorithm suite for efficiently handling *range updates*. We start our introduction with an oversimplified characterization of the data stream model. In this model, the precise system state is comprised of a large number (say U) of counters σ_0, σ_1, $\cdots$, σ_{U-1} whose values are initialized to 0. A data stream is comprised of a large number of data items that can take one of the following two forms: *standard (point-update)* and *range-update*. In a standard data stream, each item, say the t^{th}, in the data stream is in the form (i_t, δ_t). This data item should cause the following update to the precise system state: Counter σ_{i_t} is to be incremented by δ_t, which we call a *point update*. In a range-update data stream, which is more general (than standard data streams), each data item is in the form $([a_t, b_t), \delta_t)$. In this case, for each index i in the range $[a_t, b_t)$, the corresponding counter σ_i needs to be incremented by δ_t, which we call a *range update*. A typical data streaming query is to estimate a certain function of the counter values after the updates caused by all the data items in the data stream are committed to the system state. For example, the L_2-norm and the L_1-norm estimation problems are to estimate the values of $d_2 \triangleq (\sum_{i=0}^{U-1} |\sigma_i|^2)^{1/2}$ (the L_2-norm of the system state) and $d_1 \triangleq \sum_{i=0}^{U-1} |\sigma_i|$ (the L_1-norm), respectively. Since U is usually too huge for the precise system state to fit in fast memory, a data streaming algorithm has to summarize it into a synopsis data structure called a *sketch*, whose size is much smaller than $O(U)$.

A data streaming algorithm suite, proposed in [8], solves the L_2- and the L_1-norm estimation problems for standard data streams. It employs a Gaussian-sum or Cauchy-sum sketch comprised of $r > 0$ i.i.d. accumulators (viewed as RVs) $A_1, A_2, \cdots, A_r$. Since these accumulators are independent and functionally equivalent, it suffices to describe the point-update procedure for one such accumulator, which we denote as A. A is initialized to 0 at the beginning. Given a point update (i_t, δ_t), A is incremented by $\delta_t X_{i_t}$, where X_{i_t} is a standard Gaussian (for L_2-norm) or Cauchy (for L_1-norm) RV associated with the counter σ_{i_t}. The value of X_{i_t} is fixed after it is generated on-demand for the first time. After the entire data stream has passed, it was shown in [8] that $A = \sum_{i=0}^{U-1} \sigma_i X_i$ has distribution $\mathcal{N}(0, d_2^2)$ or Cauchy$(0, d_1)$ respectively, wherein the parameters d_2^2 and d_1 can be estimated using standard estimators.

This algorithm can handle a range update $([a_t, b_t), \delta_t)$ as follows:

$$\boxed{\text{For } i = a_t \text{ to } b_t - 1, \text{ do } A \leftarrow A + \delta_t X_i.}$$

However, the time complexity of this update procedure is $O(b_t - a_t)$, which is very high when $b_t - a_t$ is gigantic. In comparison, our Gaussian-DST and Cauchy-DST solutions can process this range update in $O(\log(b_t - a_t))$ time, since the net effect of this range update is to increment A by $\delta_t \cdot (\sum_{i=a_t}^{b_t-1} X_i)$, which is precisely δ_t times the (Gaussian or Cauchy) range-sum $S[a_t, b_t)$.

In [8], the median estimator is used in L_1-norm estimation, in which case all-wise independence of the underlying RVs are needed for a theoretical guarantee. However, for L_2-norm estimation, a mean-estimator is used [8], which is a standard quadratic polynomial of the accumulators $\hat{d_2^2} = (A_1^2 + A_2^2 + \cdots + A_r^2)/r$. In this case, it can be shown (e.g., using arguments similar to those in Theorem 2.2 in [1]) that the L_2-norm estimator achieves the same statistical efficiency whether the underlying RVs are 4-wise independent or all-wise independent. According to Theorem 9 (in Section 3), our Gaussian-DST solution guarantees that the underlying RVs are 4-wise independent when it is implemented using $\log_2 U$ 4-wise independent hash functions.

The *second* application is the space-efficient histogram maintenance (SEHM) problem in the data streaming setting, which as mentioned earlier was the focus of [7]. The precise system state to be approximately maintained by a proposed SEHM solution is a scaled probability mass function (pmf) $f(\cdot)$ whose domain is the set of integers $\{0, 1, 2, \cdots, U-1\}$, where the universe U is typically a large (positive) integer; we denote this domain simply as $[0, U)$. This $f(\cdot)$ starts as a zero function, and at any moment τ, $f(\cdot)$ is defined by a stream of point updates before or at τ in the sense each point update (i_τ, δ_τ) causes the value of $f(i_\tau)$ to be incremented by δ_τ. Hence $f(\cdot)$ is a “pmf in motion”.

A part of the SEHM problem is to answer the following query. At any given moment τ, the proposed SEHM solution needs to approximately represent the snapshot of $f(\cdot)$ at τ using a good and simple *histogram* function whose domain is also $[0, U)$. Here, a histogram $\mathbf{H}$ is a piecewise-constant function defined by B non-overlapping intervals (buckets) $I_1, I_2, \cdots, I_B$ that comprise $[0, U)$ and B *spline parameters* $\chi_1, \chi_2, \cdots, \chi_B$ that define the height of each bucket, as follows: $\mathbf{H}(i) = \chi_j$ when $i \in I_j$, for $i = 0, 1, \cdots, U-1$. The approximation error of $\mathbf{H}$ (relative to $f(\cdot)$) is defined as the L_2-error $(\sum_{i=0}^{U-1} |\mathbf{H}(i) - f(i)|^2)^{1/2}$ or the L_1-error $\sum_{i=0}^{U-1} |\mathbf{H}(i) - f(i)|$. A histogram $\mathbf{H}$ is called simple when B is small and called good when its approximate error is small.

A subproblem of this query problem is, given a (simple) candidate histogram $\mathbf{H}$, to determine whether it is good in terms of L_2- or L_1-error. It was shown in [7] that the SEHM problem can be solved by maintaining a Gaussian-sum (for the L_2 case) or a Cauchy-sum (for the L_1 case) sketch of $f(\cdot)$. In addition, for solving this subproblem given a candidate histogram $\mathbf{H}$, a Gaussian-sum or Cauchy-sum sketch of $\mathbf{H}$ needs to be computed. Suppose $I_j = [a_j, a_{j+1})$ for $j = 1, 2, \cdots, B$. Then the value of an accumulator A in the sketch of $\mathbf{H}$ takes value $A = \sum_{j=1}^{B} \chi_j S[a_j, a_{j+1})$ (as explained above), where each $S[a_j, a_{j+1})$ is a Gaussian or Cauchy range-sum that needs to be *efficiently computed*. It was shown in [7] that the L_2- or L_1-error of approximating $f(\cdot)$ by $\mathbf{H}$ can be estimated from the difference between the sketches of $f(\cdot)$ and $\mathbf{H}$.

We now shift our attention to the *third* application of ERS: Locality-Sensitive Hashing (LSH) schemes for approximate nearest neighbors searches (ANNS). An ERS problem arises in efficiently implementing a state-of-the-art LSH solution, called multi-probe random-walk LSH (MP-RW-LSH) [24], for ANNS in Manhattan (L_1) distance. As explained in [24], to compute the value of a random-walk LSH (RW-LSH) function acting on a query vector (as its argument), we need to map an (arbitrarily) given nonnegative even integer ϕ (which can be a very large number) to a ϕ-step random walk. This computation is precisely an ERS problem with X being a single-step random walk. The aforementioned EH3 scheme [5] does not work for this ERS problem for the following reason. It was shown in [24] that, for MP-RW-LSH to work properly, the probability distribution of any computed range-sum $S[a, b)$ must be either identical or close to that of a $(b-a)$-step random walk. This requirement, however, is not generally satisfied by EH3, which destroys all independence beyond 4-wise. In contrast, according to Theorem 9 (in Section 3), our Random Walk (RW)-DST solution strictly satisfies this requirement when it is implemented using 2-wise independent hash functions.

In this work, we make two major and nontrivial contributions. First, we propose a dyadic simulation framework and develop three novel and computationally efficient ERS solutions, namely Gaussian-DST, Cauchy-DST and RW-DST, based on it. Second, we establish a novel

k-wise independence theory that allows our ERS solutions to have both strong provable independence guarantees and low computational complexities.

2 Dyadic Simulation Theory

In this section, we first describe how to generate an arbitrary dyadic range-sum using a *dyadic simulation tree* (DST) of binary splits. After that, we describe three aforementioned DST-based *efficient range-summability* (ERS) solutions for three different target distributions. These three solutions, called Gaussian-DST, Cauchy-DST, and RW-DST (RW for random walk) respectively, follow a common framework and differ only in the binary split procedure. In the rest of the paper, whenever possible, we focus on the design and the efficient implementation of only a single instance of DST. A real-world application usually needs to use many DST instances [8, 7, 24]. These DST instances are independent in the sense that the full vectors of underlying RVs $X_0, X_1, \cdots, X_{U-1}$ generated by them are independent.

Before we describe the dyadic simulation approach, we state the precise problem statement of ERS, which consists of three requirements. First, the underlying RVs $X_0, X_1, \cdots, X_{U-1}$ are i.i.d. with distribution X. Second, every range-sum $S[a, b)$ is equal to $X_a + X_{a+1} + \cdots + X_{b-1}$. Third, given any range $[a, b)$, its range-sum $S[a, b)$ can be computed in $O(\text{polylog}(b-a))$ time. Whereas the second and the third requirements are straightforward to satisfy, to *provably* satisfy the strict independence part of the first requirement, we have to make an idealized assumption that we will elaborate on in Subsection 2.1.

As mentioned earlier, each range in $[0, U)$ can be partitioned into disjoint dyadic ranges. Such a partitioning can usually be done in multiple ways, but only one such way results in the minimum number of partitions. This minimum partitioning is called the *dyadic cover*, which contains at most $O(\log U)$ dyadic ranges [17]. For example, the dyadic cover of $[4, 11)$ contains three dyadic ranges: $[4, 8)$, $[8, 10)$, and $[10, 11)$. In the rest of the paper, we only show how to compute the sum of any dyadic range, since the sum of any general (not necessarily dyadic) range $[a, b)$ can be computed by summing up the dyadic range-sums in the dyadic cover of $[a, b)$. Also as explained earlier, for notational convenience and ease of presentation, we assume that the universe range U is a power of 2.

2.1 Dyadic Simulation Framework

In this section, we describe the dyadic simulation framework, and prove that a DST-based ERS solution satisfies all three requirements specified earlier. We illustrate a DST using a "small universe" example (with $U = 16$) shown in Figure 1. Sitting at the root of the tree is the $S[0, 16)$, which has distribution X^{*16} by initialization. Its two children are the two half-range-sums $S[0, 8)$ and $S[8, 16)$ resulting from splitting $S[0, 16)$, its four grandchildren are the four quarter-range-sums $S[0, 4)$, $S[4, 8)$, $S[8, 12)$ and $S[12, 16)$ resulting from splitting $S[0, 8)$ and $S[8, 16)$ respectively, and so on. At the bottom of the tree are the 16 underlying RVs X_0, X_1, $\cdots$, X_{15}.

Under this model, every dyadic range-sum, including every underlying RV, corresponds to a node in this tree and its value is generated by binary-splitting all its ancestors. The computational complexity of generating a dyadic range-sum is clearly $O(\log U)$ splits. Corollary 1 states the aforementioned surprising result that the computational complexity of generating the sum of any general range is also $O(\log U)$ splits. Hence a DST-based solution satisfies the third requirement above. The remark after the proof of Lemma 10 gives an informal proof of Corollary 1. In addition, under this dyadic simulation framework, the dyadic range-sum at each non-leaf tree node is the sum of two dyadic range-sums at its two

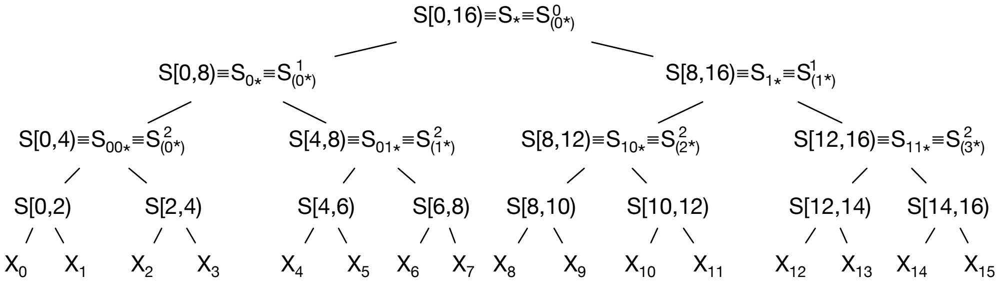

Figure 1 An illustration of the DST.

children. As a result, every dyadic range-sum $S[a, b)$ computed this way is indeed equal to $X_a + X_{a+1} + \cdots + X_{b-1}$. Hence the second requirement above is satisfied.

▶ **Corollary 1.** *For any range $[a, b) \subseteq [0, U)$, the range-sum $S[a, b)$ can be computed in no more than $2 \log_2 U$ splits.*

We now introduce the concept of *prefix* that will simplify our presentation next. Viewing the DST as a binary trie, we can index each tree node as a prefix. For example, in Figure 1, the range $[4, 8)$ is equivalent to the prefix $01*$ since it contains four binary numbers $4 = (0100)_2$, $5 = (0101)_2$, $6 = (0110)_2$, $7 = (0111)_2$ that share the common prefix $01*$.

Next, we will prove that our DST-based approach satisfies the first requirement (underlying RVs being i.i.d.) above if the split procedure possesses two properties that we call (I) and (II). Suppose a dyadic range-sum $S_{\alpha*}$ that has distribution X^{*2n} is split into $S_{\alpha 0*} + S_{\alpha 1*}$. *Property (I)* is that $S_{\alpha 0*}$ and $S_{\alpha 1*}$ are i.i.d. with distribution X^{*n}. *Property (II)* is that the random vector $\langle S_{\alpha 0*}, S_{\alpha 1*}\rangle$ is a (vector) function of only $S_{\alpha*}$ as far as independence analysis is concerned.

Now, we describe the binary split procedure. To split any $S_{\alpha*}$, we simply generate an RV $L_{\alpha*}$ using a conditional distribution that we will specify next, and then let $S_{\alpha 0*} \triangleq L_{\alpha*}$ and $S_{\alpha 1*} \triangleq S_{\alpha*} - L_{\alpha*}$. Since the split procedure is the same for any $\alpha*$, we drop the subscript $\alpha*$ from $S_{\alpha*}$ and $L_{\alpha*}$ in describing it whenever possible. In the following derivations and proofs, we assume that S is a continuous RV, so its probability density function (pdf) is used; if S is instead a discrete RV, we can use its probability mass function (pmf) instead. To split S *for the first time, a fresh RV* L is generated according to the following conditional pdf:

$$f_{L|S}(L = x|S = z) \triangleq \rho_n(x)\rho_n(z - x)/\rho_{2n}(z), \tag{1}$$

where $\rho_n(\cdot)$ and $\rho_{2n}(\cdot)$ are the pdfs of X^{*n} and X^{*2n} respectively. For notational simplicity, we drop the subscript $L|S$ from $f_{L|S}$ in the sequel. The following theorem states that this split procedure satisfies the aforementioned property (I).

▶ **Theorem 2.** *If S has distribution X^{*2n}, then the conditional distribution of $L|S$ in Equation 1 implies that L and $S - L$ are i.i.d. RVs having distribution X^{*n}.*

Proof. We first calculate the joint pdf of L and $S - L$ as follows

$$f(L = x, S - L = v) = f(L = x|S = x + v)f(S = x + v) = \rho_n(x)\rho_n(v), \tag{2}$$

where Equation 2 can be derived from Equation 1 by replacing z with $x + v$.

Hence we have $f(L = x) = \int_{-\infty}^{\infty} \rho_n(x)\rho_n(v)\mathrm{d}v = \rho_n(x)$. Similarly, $f(S - L = v) = \rho_n(v)$. Hence we have $f(L = x, S - L = v) = f(L = x)f(S - L = v)$, which proves the independence. ◀

We now put the index subscript $\alpha *$ back into S and L, since we need to state results concerning a set of S- and L-terms with different indices. We pause to clarify the mathematical meanings of two emphasized phrases used in stating the split procedure. The first phrase is "for the first time". It means that, in case $S_{\alpha *}$ is to be split again, the same $L_{\alpha *}$, that was generated and used for the first time, must be used again. This is a basic requirement for generating RVs, because the values of RVs should be fixed upon generation. The second phrase is "a fresh RV". It means that each $L_{\alpha *}$ is generated based on only the value of $S_{\alpha *}$ using fresh randomness. As a result, the random vector $\langle S_{\alpha 0 *}, S_{\alpha 1 *}\rangle$ is a vector function of only $S_{\alpha *}$ as far as independence analysis is concerned, which is precisely property (II). The language of property (II), such as "fresh randomness", is a bit vague right now. It will be further simplified and clarified in Subsection 2.2.

The aforementioned idealized assumption is simply that we can somehow remember the fresh randomness involved in generating each $L_{\alpha *}$ (for the first time), so that property (II) can be ensured. However, since the number of non-leaf prefixes in each DST is $O(U)$, it is typically prohibitively expensive in terms of storage cost to remember such fresh randomness for every $L_{\alpha *}$ generated, and this idealized assumption is impractical. Since property (II) depends on this assumption, it is also impractical. In Section 3, we will introduce a slightly weakened property (II*) that does not require this assumption, yet can still lead to strong provable statistical guarantees.

Before we state and prove the following theorem, we introduce a third notation $S^l_{(i*)}$ for a dyadic range-sum (besides $S[a,b)$ and $S_{\alpha *}$). $S^l_{(i*)}$ represents the same dyadic range-sum as $S_{\alpha *}$, if the number i, written as an l-bit binary number, is (the binary prefix) α. For example, in the example shown in Figure 1, $S^2_{(1*)}$ is equivalent to S_{01*} and $S[4,8)$. Similarly, we denote the L-term involved in splitting $S^l_{(i*)}$ as $L^l_{(i*)}$. Note that if $S_{\alpha *}$ is the same as $S^l_{(i*)}$, then $S_{\alpha 0 *}$ and $S_{\alpha 1 *}$, the two children of $S_{\alpha *}$, are the same as $S^{l+1}_{((2i)*)}$ and $S^{l+1}_{((2i+1)*)}$ respectively.

Since the DST is a full binary tree, there are 2^l nodes at the l^{th} level down the root. Under this $S^l_{(i*)}$ notation, these 2^l nodes are $S^l_{(0*)}$, $S^l_{(1*)}$, $\cdots$, $S^l_{(\lambda_l *)}$, where $\lambda_l = 2^l - 1$ (defined for any l). The following theorem states that for any $1 \le l \le \log_2 U$, these 2^l dyadic range-sums are i.i.d. RVs.

▶ **Theorem 3.** *Suppose that the split procedure satisfies properties (I) and (II). Then, for any l such that $1 \le l \le \log_2 U$, the 2^l dyadic range-sums $S^l_{(0*)}$, $S^l_{(1*)}$, $\cdots$, $S^l_{(\lambda_l *)}$ at level l have i.i.d. distribution $X^{*(U/2^l)}$.*

Proof. We prove by induction on l. For the base case when $l = 1$, there are two dyadic range-sums at the 1^{st} level: $S^1_{(0*)}$ and $S^1_{(1*)}$. Since they result from splitting S_*, which has distribution X^{*U} (by initialization), $S^1_{(0*)}$ and $S^1_{(1*)}$ are i.i.d. RVs with distribution $X^{*(U/2)}$ according to property (I).

Now, we prove the case of $l+1$ from that of l. By the induction assumption, for any i, the parent $S^l_{(i*)}$ follows $X^{*(U/2^l)}$, so by property (I), its two children $S^{l+1}_{((2i)*)}$ and $S^{l+1}_{((2i+1)*)}$ are independent and each has the marginal distribution $X^{*(U/2^{l+1})}$. We denote this as *fact (*)*. It remains to show $S^{l+1}_{(0*)}$, $S^{l+1}_{(1*)}$, $\cdots$, $S^{l+1}_{(\lambda_{l+1} *)}$, the generated range-sums on level $l+1$, are independent. By induction assumption, $S^l_{(0*)}$, $S^l_{(1*)}$, $\cdots$, $S^l_{(\lambda_l *)}$ are independent. Each $\langle S^{l+1}_{((2i)*)}, S^{l+1}_{((2i+1)*)}\rangle$ is a (vector) function of only $S^l_{(i*)}$, which we called property (II) earlier. Hence the random vectors $\langle S^{l+1}_{((2i)*)}, S^{l+1}_{((2i+1)*)}\rangle$ are independent for different i, which we denote as *fact (**)*.

Therefore, the independence of all values on level $l+1$ follows from the following factorization of the joint cdf for any sequence of values x_0, x_1, $\cdots$, $x_{\lambda_{l+1}} \in \mathbb{R}$,
$\Pr\left(S^{l+1}_{(0*)} \le x_0, S^{l+1}_{(1*)} \le x_1, \cdots, S^{l+1}_{(\lambda_{l+1}*)} \le x_{\lambda_{l+1}}\right)$
$= \prod_{i=0}^{\lambda_l} \Pr\left(S^{l+1}_{((2i)*)} \le x_{2i}, S^{l+1}_{((2i+1)*)} \le x_{2i+1}\right)$
$= \prod_{i=0}^{\lambda_l} \Pr\left(S^{l+1}_{((2i)*)} \le x_{2i}\right) \Pr\left(S^{l+1}_{((2i+1)*)} \le x_{2i+1}\right) = \prod_{i=0}^{\lambda_{l+1}} \Pr\left(S^{l+1}_{(i*)} \le x_i\right)$, where the first equation is due to fact (**) above and the second is due to fact (*) above. ◀

▶ **Corollary 4.** *The underlying RVs $X_0, X_1, \cdots, X_{U-1}$, which are $S^l_{(0*)}$, $S^l_{(1*)}$, $\cdots$, $S^l_{(\lambda_l*)}$ at level $l = \log_2 U$, have i.i.d. distribution X.*

▶ Remark. The following observation, which is a part of fact (*) in the proof of Theorem 3 above, continues to hold when property (II) is taken away, since the proof of this part only needs property (I).

▶ **Observation 5.** *Even if the split procedure satisfies only property (I), each $S^l_{(i*)}$ still has marginal distribution $X^{*(U/2^l)}$.*

The logic of the induction step in the proof of Theorem 3 can be stated as the following lemma, which will be used in the proofs in Section 3.

▶ **Lemma 6.** *If a set of $k > 1$ distinct dyadic range-sums $S^l_{(i_1*)}$, $S^l_{(i_2*)}, \cdots$, $S^l_{(i_k*)}$ at level l are independent and they are* split conditionally independently*, then their $2k$ children $S^{l+1}_{((2i_1)*)}$, $S^{l+1}_{((2i_1+1)*)}$, $S^{l+1}_{((2i_2)*)}$, $S^{l+1}_{((2i_2+1)*)}$, $\cdots$, $S^{l+1}_{((2i_k)*)}$, $S^{l+1}_{((2i_k+1)*)}$ at level $l+1$ are also independent.*

▶ Remark. Here, "split conditionally independently" means the following two conditions that together lead to fact (**). First, the L-terms involved in these splits, namely $L^l_{(i_1*)}$, $L^l_{(i_2*)}, \cdots$, $L^l_{(i_k*)}$ are conditionally independent given $S^l_{(i_1*)}$, $S^l_{(i_2*)}, \cdots$, $S^l_{(i_k*)}$. Second, each such $L^l_{(i*)}$ involved is a (random) function of $S^l_{(i*)}$ only.

2.2 Efficient Range-Summable (ERS) Solutions

As explained earlier, every DST-based solution boils down to generating $L_{\alpha*}$ according to the conditional distribution $f(L_{\alpha*}|S_{\alpha*})$ specified in Equation 1. Although Equation 1 applies to any distribution X in principle, for such a solution to work, two hurdles have to be overcome. The first hurdle is a mathematical one: Nice closed-form formulae for $\rho_n(x)$ (pdf of X^{*n}) and $\rho_{2n}(x)$ (pdf of X^{*2n}), and hence for $f(L_{\alpha*}|S_{\alpha*})$, appear to exist for only a few such X's. For other target distributions, designing DST-based ERS solutions appears to be challenging.

Even when the distribution X is nice so that we have a closed-form formula, we are still facing the second hurdle, which is to generate $L_{\alpha*}$ in a computationally efficient manner. A computational procedure for generating $L_{\alpha*}$ is typically a two-step process as follows. First, we generate a *fresh* (i.e., independent of all other RVs including $S_{\alpha*}$) uniform random μ-bit-long binary string $C_{\alpha*}$ that, if viewed as a nonnegative integer, is uniformly distributed in the set $\{0, 1, 2, \cdots, 2^\mu - 1\}$. Usually $\mu = 32$ provides enough statistical precision. Second, $L_{\alpha*}$ is set to $\theta(C_{\alpha*}, S_{\alpha*})$, where $\theta(x, z)$ is a *deterministic* function designed in such a way that the resulting $L_{\alpha*}$ has the right conditional distribution as specified in Equation 1.

Now we are ready to simplify the language of property (II) as promised earlier. The simplified property (II) is that each $C_{\alpha*}$ is a fresh RV (that is independent of any other RV). As a result, each $L_{\alpha*} \triangleq \theta(C_{\alpha*}, S_{\alpha*})$ is a *fresh* RV that is a function only of $S_{\alpha*}$, which is precisely property (II). With this simplified property (II), the idealized assumption becomes that each such $C_{\alpha*}$ (not $L_{\alpha*}$) needs to be remembered after it is first generated.

In probability theory, the standard textbook technique, called *inverse transform method* [16], is to let $\theta(x,z) = F^{-1}(x|z)$ where $F(x|z) \triangleq \int_{-\infty}^{x} f(v|z)\mathrm{d}v$ is the conditional cdf of $L_{\alpha*}|S_{\alpha*}$. However, inverse transform is usually not computationally efficient, since the inverse function of the conditional cdf $F^{-1}(x|z)$ usually does not have a closed form, as we will elaborate in Subsection 2.4. We will show that, for all three ERS solutions, we propose alternative designs of $\theta(x,z)$ that are much more efficient, in terms of computational and/or space complexity, than the respective inverse transforms. Finally, when X is a discrete RV (e.g., when X is a single-step random walk), it is possible to precompute $F^{-1}(x|z)$ for all possible values of x and z, and store the values in a table. This technique, called the tabular inverse transform [9], can only be used when the memory cost of storing the table is manageable.

2.3 Gaussian-DST

For notational simplicity, we again drop the subscript $\alpha*$ from $S_{\alpha*}$, $L_{\alpha*}$, and $C_{\alpha*}$ in describing the binary split procedures in the sequel. When X is standard Gaussian $\mathcal{N}(0,1)$, X^{*n} is $\mathcal{N}(0,n)$ with pdf $\rho_n(x) = 1/\sqrt{2\pi n}\cdot\exp(-x^2/(2n))$, and X^{*2n} is $\mathcal{N}(0,2n)$ with pdf $\rho_{2n}(x) = 1/\sqrt{4\pi n}\cdot\exp(-x^2/(4n))$. According to Equation 1, we have $f(L=x|S=z) = \rho_n(x)\rho_n(z-x)/\rho_{2n}(z) = 1/\sqrt{\pi n}\cdot\exp(-(x-z/2)^2/n)$, which can be written as the pdf of $\mathcal{N}(z/2,n/2)$. We generate L from the distribution $\mathcal{N}(z/2,n/2)$ according to (the value of) the random string C as follows. L is set to $z/2+Y$, where Y is a fresh Gaussian RV with distribution $\mathcal{N}(0,n/2)$ generated from C using efficient techniques such as Box-Muller transform [16]. In [7], no specific technique was suggested for generating this L.

2.4 Cauchy-DST

Now we describe how to generate L from C when the target distribution X is standard Cauchy (Cauchy$(0,1)$). By the stability property of Cauchy distribution [8], the n^{th} convolution power X^{*n} is Cauchy$(0,n)$, which has pdf $\rho_n(x) = \left(\pi n\left[1+(x/n)^2\right]\right)^{-1}$. The pdf of X^{*2n} is $\rho_{2n}(x) = \left(2\pi n\left[1+(x/2n)^2\right]\right)^{-1}$. Therefore, by Equation 1, the conditional pdf is

$$f(L=x|S=z) = \frac{\rho_n(x)\rho_n(z-x)}{\rho_{2n}(z)} = \frac{n}{2\pi}\cdot\frac{z^2+4n^2}{(n^2+x^2)(n^2+(z-x)^2)}. \tag{3}$$

In [7], it was suggested that the inverse transform method described above be used to generate L. The rationale offered in [7] was that since the conditional pdf $f(x|z)$ in Equation 3 is a rational fraction, the conditional cdf $F(x|z)$ has a closed-form expression [19], which makes its inverse $F^{-1}(x|z)$ numerically calculable. However, the procedure for calculating $F^{-1}(x|z)$ has a high computational complexity in practice, since the (closed-form) formula of $F(x|z)$ is very complicated.

We propose a much more efficient way of sampling L from $f(x|z)$ based on a Monte Carlo simulation technique called *rejection sampling* [4]. The idea of rejection sampling is that, we instead sample another RV Y from another pdf $\psi(x|z)$ that is computationally easier to sample from than $f(x|z)$. Supposing the value of this sample is x. Then this sample is accepted with probability $\gamma = f(x|z)/(Q\psi(x|z))$ and rejected with probability $1-\gamma$. The rejection sampling step is repeated until a sample of Y is accepted, and the finally accepted sample is (the realized value of) L. Here, this constant Q should be set such that $\gamma \le 1$ for all values of x and z, or in other words $Q \ge \max_{x,z} f(x|z)/\psi(x|z)$. In statistics, a key objective as well as challenge in designing a rejection sampling procedure is to select $\psi(x|z)$ so that $\max_{x,z} f(x|z)/\psi(x|z)$ and hence this Q can be made as small as possible. Hence the *probability of acceptance*, defined as probability that any sample thus generated is accepted, (which is equal to $1/Q$ as shown in [4] pp. 51) is made as large as possible.

We propose to sample RV Y (whose conditional pdf is $\psi(x|z)$) from the following mixture distribution: Y is equal to Y' or $Y'+z$ each with probability $1/2$ (depending on the value of C), where Y' is a fresh RV with distribution Cauchy$(0, n)$. This Y' can generated from C via the aforementioned inverse transform $Y' = F_{Y'}^{-1}(C) = n\tan(\pi(C-1/2))$; note that, unlike the conditional inverse cdf $F^{-1}(x|z)$ described above, the inverse function of the unconditional cdf $F_{Y'}^{-1}(C)$ here takes a much simpler form and hence can be computed efficiently. It can be shown that the conditional pdf of Y is

$$\psi(L=x|S=z) = \frac{\rho_n(x)+\rho_n(x-z)}{2} = \frac{n}{2\pi}\cdot\frac{2n^2+x^2+(z-x)^2}{(n^2+x^2)(n^2+(z-x)^2)}.$$

We set the parameter Q to 2 so that the probability of acceptance is $1/2$, since for any x and z, we have

$$\frac{f(L=x|S=z)}{\psi(L=x|S=z)} = \frac{4n^2+z^2}{2n^2+x^2+(z-x)^2} = \frac{4n^2+z^2}{2n^2+z^2/2+2(x-z/2)^2} \le 2. \tag{4}$$

.

2.5 Random Walk (RW)-DST

We now describe how to generate L from $S=z$ and C when the target distribution X is a single-step random walk. We first derive the conditional pmf $f(L=x|S=z)$. Since X^{*n} has pmf $\rho_n(x) = 2^{-n}\binom{n}{(n-x)/2}$, and X^{*2n} has pmf $\rho_{2n}(x) = 2^{-2n}\binom{2n}{(2n-x)/2}$, by Equation 1, the conditional pmf is

$$f(L=x|S=z) = \frac{\rho_n(x)\rho_n(z-x)}{\rho_{2n}(z)} = \binom{n}{(n-x)/2}\binom{n}{(n-z+x)/2}\Big/\binom{2n}{n-z/2}, \tag{5}$$

if z is an even integer such that $-2n \le z \le 2n$, x is an integer such that $-n \le x \le n$ and $-n+z \le x \le n+z$, and $n-x$ is even; otherwise $f(L=x|S=z)=0$.

We now introduce a concept that will become handy in the rest of this section. We say that y is a *probable value* of a discrete RV Y, if the probability $P(Y=y)$ is not 0 or vanishingly small. This concept is important here, because we will trade memory space for computation time by precomputing and storing some conditional probability values, and the memory cost could be greatly reduced if we store only those for probable values of S and L conditioned upon S. Now we analyze the asymptotic number of probable values of S and L when n is a large number. For S, only integers that are no larger than $O(\sqrt{n})$ are probable, since as will be shown in the proof of Proposition 7, its pmf $\rho_{2n}(x)$ (on integer values of x) is close to the pdf of $\mathcal{N}(0, 2n)$, which is $1/\sqrt{4\pi n}\cdot\exp(-x^2/(4n))$ as shown above. The above formula is not vanishingly small only when $x = O(\sqrt{n})$. Hence, by storing probability values only for the probable values of S, the space complexity reduces from $O(n)$ to $O(\sqrt{n})$. The same can be said about L for a similar reason.

We have tried the aforementioned tabular inverse transform method [9] on $f(L=x|S=z)$. However, even when the probable value trick is used, the memory cost is still very high for most applications. The total memory cost is $O(U)$, since for each of the $\log U$ values of n, we need to store the values of $f(L=x|S=z)$ for all combinations of $O(\sqrt{n})$ probable z values and $O(\sqrt{n})$ probable x values, and the largest n value is U. For example, when $U = 2^{20}$, the total size of the precomputed tables would still be several gigabytes.

We propose a rejection sampling technique that, in combination with the tabular inverse transform and the probable value trick, provides a fast, space-efficient, and accurate solution to this ERS problem. Like in Subsection 2.4, the rejection sampling method is specified by the RV Y whose conditional pdf (given $S = z$) is $\psi(x|z) = 2^{-n}\binom{n}{(n-x+2\lceil z/4\rceil)/2}$, and the constant Q (defined later). Y can be generated as $Y' + 2\lceil z/4\rceil$, where Y' is a fresh RV with distribution X^{*n} generated from C by tabular inverse transform [9]. Our next step is to determine Q, which is an upper bound on the ratio $f(x|z)/\psi(x|z)$ for each n value and for all probable x and z values (those that are $O(\sqrt{n})$ as explained earlier). For all $n \geq 256$, we know from calculations and from Proposition 7 that this ratio is at most 1.47. Hence, we set $Q = 1.47$ so that the probability of acceptance is $1/1.47 = 0.68$. The rejection sampling operation is computationally efficient, because both $f(x|z)$ and $\psi(x|z)$ can be computed in $O(1)$ time if the factorials $i!$ and $(n-i)!$ are precomputed for probable i values (that is $i = O(\sqrt{n})$). When $n \geq 256$, we use rejection sampling (with $Q = 1.47$). When $n \leq 128$, we use the tabular inverse transform (with the probable value trick) since the table size grows as $O(n)$ as explained earlier. When $U = 2^{20}$ like in the example above, the total size of the precomputed tables (for all 20 values of n) is only several megabytes.

▶ **Proposition 7.** *When n is large and $z = O(\sqrt{n})$ is a probable value, the maximum ratio $\max_{x=O(\sqrt{n})} f(x|z)/\psi(x|z)$ is close to $\sqrt{2} \approx 1.414$.*

Proof. By de Moivre-Laplace Theorem [13], when n is large and $x = O(\sqrt{n})$ is a probable value, $\rho_n(x)$ is close to the pdf of $\mathcal{N}(0, n)$ at x, which is $1/\sqrt{2\pi n} \cdot \exp(-x^2/(2n))$. Similarly, $\rho_n(z - x)$ is close to $1/\sqrt{2\pi n} \cdot \exp(-(z - x)^2/(2n))$, and $\rho_{2n}(x)$ is close to $1/\sqrt{4\pi n} \cdot \exp(-x^2/(4n))$. By straightforward computation, $f(x|z)$ in Equation 5 is close to $1/\sqrt{\pi n} \cdot \exp(-(x - z/2)^2/n)$. Similarly, $\psi(x|z)$, the conditional pdf of Y is close to $1/\sqrt{2\pi n} \cdot \exp(-(x - 2\lceil z/4\rceil)^2/(2n))$. If z is a multiple of 4, the maximum ratio is achieved at $x = z/2$, and the ratio is $\sqrt{2}$. Otherwise, z is an even number but not a multiple of 4, the maximum ratio is achieved at $x = 2\lfloor z/4\rfloor$, and the ratio is $\sqrt{2}\exp(1/n)$, which is close to $\sqrt{2}$ when n is large. ◀

2.6 Speed of Dyadic Simulation

Recall that our idealized and impractical assumption is that we can somehow remember the value of every $C_{\alpha*}$ after it was first generated, with which we can rigorously prove that $X_0, X_1, \cdots, X_{U-1}$ are i.i.d. As mentioned in Subsection 1.2, this assumption can be removed by instead computing each such $C_{\alpha*}$ as $h(\alpha)$, where $h(\cdot)$ is a hash function.

We have implemented the DST framework using an off-the-shelf hash function family called `wyhash` [25]. `wyhash` offers two attractive advantages. First, computationally `wyhash` is very efficient: It takes roughly two nanoseconds for `wyhash` to compute a hash value [25]. Second, it guarantees excellent empirical independence among the values of $(C_{\alpha*})$'s generated in the sense that it passes a number of quality tests in SMhasher, a well-established benchmark for hash functions [23]. To further improve this empirical independence, we use a different (independent) hash function at each level of the DST. The storage cost of a DST is tiny, since each hash function uses only a 32-bit random seed that needs to be remembered. Table 1 shows the average amount of time it takes for a DST to split a Gaussian, Cauchy, and random walk RV respectively, measured on a workstation running Ubuntu 18.04 with Intel(R) Core(TM) i9-10980XE 3.00 GHz CPU. It is a few times faster to split a Gaussian than to split the other two, largely because the other two involve rejection sampling, which is a relatively computationally intensive process.

As shown in Corollary 1, the generation of any range-sum involves at most $2\log_2 U$ splits, so for typical universe sizes (say $U = 2^{32}$), the total time for generating a range-sum is around or less than 1 µs for all three target distributions in Table 1.

Table 1 Average split time of an RV.

Distribution	Gaussian	Cauchy	Random Walk
Time per split (ns)	4.8	24.8	21.2

3 k-wise Independence Theory for DST

At the end of the previous section, we have shown that, by using a per-level `wyhash` function to hash a prefix $\alpha*$ into a uniformly random string $C_{\alpha*}$, our solutions have high performance and the underlying RVs are empirically independent. However, `wyhash` does not provide any theoretical guarantee concerning independence. In this section, we describe our novel k-wise independence theory that provides both high computational efficiency and strong provable independence guarantees.

Our k-wise independence theory guarantees that the U underlying RVs X_0, $X_1, \cdots$, X_{U-1} generated by the DST are k-wise independent in the sense that given an arbitrary set of k different indices $i_1, i_2, \cdots, i_k$ in the universe $[0, U)$, the RVs $X_{i_1}, X_{i_2}, \cdots, X_{i_k}$ are independent. To this end, our idea is to use $\log_2 U$ k-wise independent hash functions (instead of `wyhash`) to generate $(C_{\alpha*})$'s. A k-wise independent hash function $h(\cdot)$ has the following property: Given an arbitrary set of k different keys $i_1, i_2, \cdots, i_k$, their hash values $h(i_1), h(i_2), \cdots, h(i_k)$ are independent. Such hash functions are very computationally efficient when k is a small number such as $k = 2$ (roughly 2 nanoseconds per hash just like `wyhash`) and $k = 4$ (several nanoseconds per hash) [3, 21, 15].

Our scheme uses $\log_2 U$ independent k-wise independent hash functions that we denote as $h^l(\cdot)$, for $l = 0, 1, \cdots, \log_2 U - 1$. During the initialization phase, we seed these $\log_2 U$ hash functions each using a uniformly random binary string; once seeded, they are deterministic (hash) functions thereafter. Our scheme can be stated in *literally one sentence:* Each such (seeded and fixed) $h^l(\cdot)$ is solely responsible for hash-generating all random strings $C_{\alpha*}$ in which the binary prefix α is a l-bit number. For example, when a random string $C_{\alpha*}$ is needed for computing a range-sum, we rewrite $C_{\alpha*}$ into $C^l_{(i*)}$ in the same way as we did on $S_{\alpha*}$ in the second last paragraph before Theorem 3. Then, $C^l_{(i*)}$ is hash-generated as $h^l(i)$ just like using `wyhash`. Note that the hash function $h^l(\cdot)$ is a random function before it is seeded, so $h^l(i)$ for any i can be viewed as a RV where the randomness comes from the seed of $h^l(\cdot)$. We will use this view in the proof that follows.

The construction above weakens property (II) slightly. The weakened one, called *property (II*)*, is that, at any level l, any k distinct range-sums $S^l_{(i_1*)}, S^l_{(i_2*)}, \cdots, S^l_{(i_k*)}$ are split conditionally independently. The construction above can guarantee property (II*), because their "split seeds" $C^l_{(i_1*)}, C^l_{(i_2*)}, \cdots, C^l_{(i_k*)}$ are not only independent among themselves (thanks to $h^l(\cdot)$ being k-wise independent) but also independent of $S^l_{(i_1*)}, S^l_{(i_2*)}, \cdots, S^l_{(i_k*)}$ (since $h^l(\cdot)$ is a fresh hash function that has never been used in hash-generating any such $S^l_{(i*)}$). With this construction, the DST has the following nice k-wise independence property at every level.

▶ **Theorem 8.** *If every $h^l(\cdot)$, for $1 \le l < \log_2 U$, is k-wise independent, then for any l such that $1 \le l \le \log_2 U$, the 2^l range-sums $S^l_{(0*)}$, $S^l_{(1*)}$, $\cdots$, $S^l_{(\lambda_l *)}$ are k-wise independent.*

Proof. The proof is similar to that of Theorem 3 by induction. For the base case when $l = 1$, there are two dyadic range-sums at the 1^{st} level: $S^1_{(0*)}$ and $S^1_{(1*)}$. Since they result from splitting S_*, which follows X^{*U} (by initialization), $S^1_{(0*)}$ and $S^1_{(1*)}$ are i.i.d. RVs according to property (I).

Now, we prove the induction on level $l+1$ from level l. For any fixed set of k indices i_1, i_2, $\cdots$, i_k on level $l+1$, we need to prove $S^{l+1}_{(i_1*)}$, $S^{l+1}_{(i_2*)}$, $\cdots$, $S^{l+1}_{(i_k*)}$ are independent. This follows from Lemma 6, since these k elements are the children of no more than k parents after duplicates are removed. These parents, no more than k in number, are independent by the induction hypothesis and are split conditionally independently by property (II*). ◀

Theorem 8 implies that the underlying RVs $X_0, X_1, \cdots, X_{U-1}$, which are the U singleton range-sums at level $\log_2 U$, are also k-wise independent. This implication, however, is far from capturing the "full theoretical strength" of our k-wise independence theory. For example, under the assumption that every $h^l(\cdot)$, for $1 \le l < \log_2 U$, is 2-wise independent, Theorem 8 can only guarantee that the underlying RVs are 2-wise independent. In contrast, under this assumption, Theorem 9 guarantees a distributional property that is much stronger than the underlying RVs being 2-wise independent. Here we explain this point by an example. Suppose we take out this assumption, and instead make the alternative assumption that the underlying RVs are k-wise independent for a certain k. It is not hard to prove that, to guarantee the same distributional property, we would have to assume that k is as large as U, or in other words that the underlying RVs are all-wise independent.

This distributional property is in practice very useful. As mentioned earlier in Subsection 1.3, when X is a single-step random walk, this distributional property satisfies the requirement of RW-LSH. Hence any RW-LSH value computed using our RW-DST scheme is statistically indistinguishable from the original RW-LSH value (computed in the original inefficient manner), as long as every $h^l(\cdot)$, for $1 \le l < \log_2 U$, is 2-wise independent. Theorem 9 is an immediate corollary of Lemma 10.

▶ **Theorem 9.** *If every $h^l(\cdot)$, for $1 \le l < \log_2 U$, is 2-wise independent, then for any range $[a, b) \subseteq [0, U)$, the range-sum $S[a, b)$ has marginal distribution $X^{*(b-a)}$.*

▶ **Lemma 10.** *If every $h^l(\cdot)$ is 2-wise independent, then for any $1 \le l < \log_2 U$ and any integers a, b such that $0 \le a \le b < 2^l$, the following two properties hold.*

1. *The three RVs $\sum_{i=a+1}^{b-1} S^l_{(i*)}$, $S^l_{(a*)}$, and $S^l_{(b*)}$ are independent.*
2. *The range-sum $S[aU/2^l, (b+1)U/2^l) = \sum_{i=a}^{b} S^l_{(i*)}$ has distribution $X^{*((b-a+1)U/2^l)}$, where $(b-a+1)U/2^l$ is the number of underlying RVs contained in the range $[aU/2^l, (b+1)U/2^l)$.*

Before we start the proof, we note that Observation 5, which states that the marginal distribution of any range-sum $S^l_{(i*)}$ being $X^{*(U/2^l)}$ continues to hold despite the weakening of property (II) in this section.

Proof. The proof for Lemma 10 is by induction on l. For the base case when $l = 1$, the three RVs, after 0's and duplicates are removed, belong to the set of two range-sums on the first level, $S^1_{(0*)}$ and $S^1_{(1*)}$. These two range-sums have distribution i.i.d. $X^{*(U/2)}$ thanks to property (I). This leads to the two properties on the first level.

We now prove the case of level $l+1$ from that of level l. We first define the following notations: $a' \triangleq \lfloor a/2 \rfloor$ (so that $S^l_{(a'*)}$ is the parent of $S^{l+1}_{(a*)}$); $b' \triangleq \lfloor b/2 \rfloor$; $\tilde{i}$ is defined as $i+1$ if i is even (the younger of the siblings) and as $i-1$ otherwise so that $S^l_{(\tilde{i}*)}$ is always the other sibling of $S^l_{(i*)}$. Without loss of generality, we assume $a' < b'$, since otherwise ($a' = b'$) the two induction claims become trivial. The induction claim of the first property that $\sum_{i=a+1}^{b-1} S_{(i*)}^{l+1}$ and $\langle S_{(a*)}^{l+1}, S_{(b*)}^{l+1}\rangle$ are independent holds, due to the following two facts: (i) $\sum_{i=a'+1}^{b'-1} S_{(i*)}^{l}$, $S_{(a*)}^{l+1}$, $S_{(\tilde{a}*)}^{l+1}$, $S_{(b*)}^{l+1}$, and $S_{(\tilde{b}*)}^{l+1}$ are independent; (ii) $\sum_{i=a+1}^{b-1} S_{(i*)}^{l+1}$ is a deterministic function of $\langle \sum_{i=a'+1}^{b'-1} S_{(i*)}^{l}, S_{(\tilde{a}*)}^{l+1}, S_{(\tilde{b}*)}^{l+1}\rangle$ in the sense that $\sum_{i=a+1}^{b-1} S_{(i*)}^{l+1} = \sum_{i=a'+1}^{b'-1} S_{(i*)}^{l} + S_{(\tilde{a}*)}^{l+1}\mathbf{1}_{even}(a) + S_{(\tilde{b}*)}^{l+1}\mathbf{1}_{odd}(b)$, where $\mathbf{1}_{even}(a)$ is 1 if a is an even integer and is 0 otherwise, and similarly $\mathbf{1}_{odd}(b)$ is 1 if b is an odd integer and is 0 otherwise.

We now prove fact (i). By the induction assumption, the three RVs $\sum_{i=a'+1}^{b'-1} S_{(i*)}^{l}$, $S_{(a'*)}^{l}$, and $S_{(b'*)}^{l}$ are independent. Since $h^l(\cdot)$ is a fresh 2-wise independent hash function, property (II*) holds for $k = 2$, so the four children $S_{(a*)}^{l+1}$, $S_{(\tilde{a}*)}^{l+1}$, $S_{(b*)}^{l+1}$, and $S_{(\tilde{b}*)}^{l+1}$ are independent by Lemma 6. Furthermore, these four children and $\sum_{i=a'+1}^{b'-1} S_{(i*)}^{l}$ together are independent, because by property (II*), $\langle S_{(a*)}^{l+1}, S_{(\tilde{a}*)}^{l+1}\rangle$ is a function of only $S_{(a'*)}^{l}$ but not $\sum_{i=a'+1}^{b'-1} S_{(i*)}^{l}$, which is composed of the other range-sums, and similarly $\langle S_{(b*)}^{l+1}, S_{(\tilde{b}*)}^{l+1}\rangle$ is a function of only $S_{(b'*)}^{l}$. By the induction assumption, $\sum_{i=a'+1}^{b'-1} S_{(i*)}^{l}$ is independent of these four children.

We now prove the induction claim of the second property that $\sum_{i=a}^{b} S_{(i*)}^{l+1}$ has distribution X^{*n} with $n = (b - a + 1)U/2^{l+1}$. By the second property in the induction assumption, $\sum_{i=a'+1}^{b'-1} S_{(i*)}^{l}$ has distribution $X^{*n'}$, where $n' = (b' - a' - 1)U/2^l$ is the number of $(X_i)'s$ (underlying RVs) contained in its range $[(a' + 1)U/2^l, b'U/2^l)$. To see why the induction claim holds, we have to go through four possible cases on the parities of a and b. We show the most inclusive case where a is even and b is odd, and the other three cases are just similar. In this case, $b - a = 2(b' - a') + 1$, so $n = (2(b' - a') + 2)U/2^{l+1} = (2(b' - a') - 2 + 4)U/2^{l+1} = n' + 4U/2^{l+1}$. $\sum_{i=a}^{b} S_{(i*)}^{l+1}$ has distribution X^{*n}, because it is the sum of the following five independent RVs (*cf.* fact (i)): $\sum_{i=a'+1}^{b'-1} S_{(i*)}^{l}$, $S_{(a*)}^{l+1}$, $S_{(\tilde{a}*)}^{l+1}$, $S_{(b*)}^{l+1}$, and $S_{(\tilde{b}*)}^{l+1}$. The first RV has distribution $X^{*n'}$, and the other four RVs each has distribution $X^{*(U/2^{l+1})}$ by Observation 5. ◀

▶ Remark. In this proof, to compute the range-sum $S[a, b) = \sum_{i=a}^{b-1} S_{(i*)}^{\log_2 U}$, at most two splits need to performed at each level $0 \leq l < \log_2 U$, on namely $S_{(a_l*)}^{l}$ and $S_{(b_l*)}^{l}$ (they can be the same node), where $a_l = \lfloor a2^l/U \rfloor$ and $b_l = \lfloor b2^l/U \rfloor$. This implies Corollary 1.

4 Related Work

Since the contribution of this work is a new and practical solution approach to the ERS problem, we focus only on related works on ERS. The ERS problem was first formulated in [5]. In [5], the aforementioned EH3, which is the first ERS solution, was proposed to augment the AMS sketching [1] technique. The EH3-augmented AMS solves a wide range of new data streaming problems, such as estimating the size of spatial joins and the selectivity of histogram buckets, and outperforms previous ad-hoc solutions [17]. In the ERS literature, the one most related to this work is [7]. We have compared our work with [7] in several places throughout this paper.

All existing ERS solutions except [7] are proposed for the case in which the target distribution X is a single-step random walk. Among them, EH3 [5] is the best known and has been compared with our dyadic simulation approach in Subsection 1.3. The "3" in EH3 refers to the fact that underlying RVs generated by EH3 are provably 3-wise independent. BCH3 [17] is another ERS scheme that also guarantees 3-wise independence. Although BCH3 is faster to compute than EH3, the underlying RVs generated by BCH3 are even more strongly correlated (beyond 4-wise) [17] than those by EH3. RM7 [2], which guarantees 7-wise independence, is the only existing ERS scheme that goes beyond 3-wise, but it is too slow to be practical. Empirically, RM7 takes more than 26 ms to compute a single range-sum [17], whereas for dyadic simulation, the time is typically about 1 µs as shown in Subsection 2.6.

Besides performance, another issue for these schemes is that they destroy all empirical independence beyond 4-wise (in the cases of EH3 and BCH3) and 8-wise (in the case of RM7). For existing ERS solutions except dyadic simulation, this destruction (of empirical independence) is unavoidable due to the fact that they all solve an ERS problem by crafting a "magic" hash function that is based on error correction codes. For example, in RM7 this magic hash function is defined by an instantiation of the Reed-Muller (RM) code. In contrast, in our dyadic simulation approach, the ERS is achieved through a DST that does not require any such magic hash function: For all practical purposes, `wyhash` will do, as explained earlier. This difference allows our approach to generalize to more target distributions and more applications.

A marginally related range-efficient computing problem to ERS, called efficient range minimizability (ERM), has been studied in the contexts of data streaming and computational geometry. In ERM, we would like to efficiently compute the minimum value of the RVs (that each has distribution X) in a range. An example ERM problem is when X is a uniform distribution in the interval $(0, 1)$. We have come up with a new efficient solution to this problem, but cannot include it in this paper in the interest of space. Any efficient solution (including ours) to this problem can be used, in combination with the MinHash sketch [6], to solve the range-efficient F_0 (estimation) problem [14, 20]: to efficiently estimate the number of distinct elements (F_0) in a data stream with range-updates. Existing solutions to this problem, such as range-efficient sampling [14, 20], are sampling-based in the sense they maintain a select subset of sampled data items instead of a sketch (e.g., accumulators like in [8]). The range-efficient F_0 problem has been generalized to high-dimensional spaces, where it is called the Klee's measure problem in computational geometry [22, 18, 10]. Existing solutions to Klee's measure problem are also sampling based.

5 Conclusion

In this work, we propose *dyadic simulation*, a novel solution framework to ERS that extends and improves existing frameworks in a fundamental and systematic way. We develop three novel ERS solutions for Gaussian, Cauchy, and single-step random walk distributions. We also propose novel rejection sampling techniques to make these solutions computationally efficient. Finally, we develop a novel k-wise independence theory of DSTs that provide both high computational efficiency and strong provable independence guarantees.

References

1 Noga Alon, Yossi Matias, and Mario Szegedy. The space complexity of approximating the frequency moments. In *Proceedings of the Twenty-Eighth Annual ACM Symposium on Theory of Computing*, STOC '96, pages 20–29, New York, NY, USA, 1996. Association for Computing Machinery. doi:10.1145/237814.237823.

2 A. Robert Calderbank, Anna C. Gilbert, Kirill Levchenko, Shan Muthukrishnan, and Martin Strauss. Improved range-summable random variable construction algorithms. In *Proceedings of the Sixteenth Annual ACM-SIAM Symposium on Discrete Algorithms*, SODA '05, pages 840–849, USA, 2005. Society for Industrial and Applied Mathematics. URL: https://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.86.6849.

3 J. Lawrence Carter and Mark N. Wegman. Universal classes of hash functions. *Journal of Computer and System Sciences*, 18(2):143–154, 1979. doi:10.1016/0022-0000(79)90044-8.

4 George Casella, Christian P. Robert, and Martin T. Wells. *Generalized Accept-Reject Sampling Schemes*, volume Volume 45 of *Lecture Notes–Monograph Series*, pages 342–347. Institute of Mathematical Statistics, Beachwood, Ohio, USA, 2004. doi:10.1214/lnms/1196285403.

5 Joan Feigenbaum, Sampath Kannan, Martin J. Strauss, and Mahesh Viswanathan. An approximate L_1-difference algorithm for massive data streams. *SIAM Journal on Computing*, 32(1):131–151, 2002. doi:10.1137/S0097539799361701.

6 Philippe Flajolet and G. Nigel Martin. Probabilistic counting algorithms for data base applications. *Journal of Computer and System Sciences*, 31(2):182–209, 1985. doi:10.1016/0022-0000(85)90041-8.

7 Anna C. Gilbert, Sudipto Guha, Piotr Indyk, Yannis Kotidis, S. Muthukrishnan, and Martin J. Strauss. Fast, small-space algorithms for approximate histogram maintenance. In *Proceedings of the Thiry-Fourth Annual ACM Symposium on Theory of Computing*, STOC '02, pages 389–398, New York, NY, USA, 2002. Association for Computing Machinery. doi:10.1145/509907.509966.

8 Piotr Indyk. Stable distributions, pseudorandom generators, embeddings, and data stream computation. *J. ACM*, 53(3):307–323, May 2006. doi:10.1145/1147954.1147955.

9 George Marsaglia, Wai Wan Tsang, and Jingbo Wang. Fast generation of discrete random variables. *Journal of Statistical Software, Articles*, 11(3):1–11, 2004. doi:10.18637/jss.v011.i03.

10 Kuldeep S. Meel, N.V. Vinodchandran, and Sourav Chakraborty. *Estimating the Size of Union of Sets in Streaming Models*, pages 126–137. Association for Computing Machinery, New York, NY, USA, 2021. doi:10.1145/3452021.3458333.

11 S. Muthukrishnan and Martin Strauss. *Approximate Histogram and Wavelet Summaries of Streaming Data*, pages 263–281. Springer Berlin Heidelberg, Berlin, Heidelberg, 2016. doi:10.1007/978-3-540-28608-0_13.

12 Noam Nisan. Pseudorandom generators for space-bounded computation. *Combinatorica*, 12(4):449–461, 1992. doi:10.1007/BF01305237.

13 Athanasios Papoulis. *Probability, Random Variables and Stochastic Processes.* McGraw-Hill, 1984.

14 A. Pavan and Srikanta Tirthapura. Range-efficient counting of distinct elements in a massive data stream. *SIAM Journal on Computing*, 37(2):359–379, 2007. doi:10.1137/050643672.

15 Mihai Pundefinedtraşcu and Mikkel Thorup. The power of simple tabulation hashing. *J. ACM*, 59(3), June 2012. doi:10.1145/2220357.2220361.

16 Christian P. Robert and George Casella. *Monte Carlo Statistical Methods*, page 43. Springer New York, 2004. doi:10.1007/978-1-4757-4145-2_2.

17 Florin Rusu and Alin Dobra. Pseudo-random number generation for sketch-based estimations. *ACM Trans. Database Syst.*, 32(2):11–es, June 2007. doi:10.1145/1242524.1242528.

18 Gokarna Sharma, Costas Busch, Ramachandran Vaidyanathan, Suresh Rai, and Jerry L. Trahan. Efficient transformations for Klee's measure problem in the streaming model. *Computational Geometry*, 48(9):688–702, 2015. doi:10.1016/j.comgeo.2015.06.007.

19 James Stewart. *Calculus: Early Transcendentals.* Brooks/Cole, 4 edition, 1999.

20 He Sun and Chung Keung Poon. Two improved range-efficient algorithms for F_0 estimation. *Theoretical Computer Science*, 410(11):1073–1080, 2009. Algorithms, Complexity and Models of Computation. doi:10.1016/j.tcs.2008.10.031.

21 Mikkel Thorup and Yin Zhang. Tabulation based 4-universal hashing with applications to second moment estimation. In *Proceedings of the Fifteenth Annual ACM-SIAM Symposium on Discrete Algorithms*, SODA '04, pages 615–624, USA, 2004. Society for Industrial and Applied Mathematics.

22 Srikanta Tirthapura and David Woodruff. Rectangle-efficient aggregation in spatial data streams. In *Proceedings of the 31st ACM SIGMOD-SIGACT-SIGAI Symposium on Principles*

of Database Systems, PODS '12, pages 283–294, New York, NY, USA, 2012. Association for Computing Machinery. doi:10.1145/2213556.2213595.

23 Reini Urban and et al. Smhasher: Hash function quality and speed tests. GitHub repository, https://github.com/rurban/smhasher. accessed on Jul 23, 2021.

24 Huayi Wang, Jingfan Meng, Long Gong, Jun Xu, and Mitsunori Ogihara. MP-RW-LSH: An efficient multi-probe lsh solution to ANNS-L1. *Proc. VLDB Endow.*, 14(13):3267–3280, September 2021. doi:10.14778/3484224.3484226.

25 Yi Wang. wyhash: The dream fast hash function and random number generators. GitHub repository, https://github.com/wangyi-fudan/wyhash. Accessed on Feb 9, 2021.