entropy

MDPI

Article

Input Pattern Classification Based on the Markov Property of the IMBT with Related Equations and Contingency Tables

István Finta [1,*,†], Sándor Szénási [2,†] and Lóránt Farkas [1,†]

1 Nokia, Bell Labs, Bókay street 36–42, H-1083 Budapest, Hungary; lorant.farkas@nokia-bell-labs.com
2 Department of Informatics, J. Selye University, Bratislavská cesta 3322, SK-94501 Komárno, Slovakia; szenasis@ujs.sk
* Correspondence: istvan.finta@nokia-bell-labs.com
† These authors contributed equally to this work.

Received: 1 December 2019; Accepted: 17 February 2020; Published: 21 February 2020

Abstract: In this contribution, we provide a detailed analysis of the search operation for the Interval Merging Binary Tree (IMBT), an efficient data structure proposed earlier to handle typical anomalies in the transmission of data packets. A framework is provided to decide under which conditions IMBT outperforms other data structures typically used in the field, as a function of the statistical characteristics of the commonly occurring anomalies in the arrival of data packets. We use in the modeling Bernstein theorem, Markov property, Fibonacci sequences, bipartite multi-graphs, and contingency tables.

Keywords: data structure; balanced binary tree; bipartite graph; Bernstein theorem, Fibonacci sequence; Markov property; state space; contingency table

1. Introduction

In large-scale or distributed systems, different types of data are generated locally, in the computer nodes composing the system. However, if the generated data also has to be handled or stored in a remote location, then the data has to be broken into packets and transmit over the network towards their destination. Since neither the transmission networks themselves, nor the consisting devices are perfect, some of the packets may arrive out-of-order, while others may be lost permanently. Additionally, packets that are mistakenly considered to be permanently lost during the transmission are sent several times. This may result in duplication on the receiving side.

Applications need to be prepared to handle out-of-order delivery, packet duplication, and packet loss, to an extent depending on the application specifics. A video application might neglect them overall, while a banking application might have very strict requirements to mitigate them. The efficiency of handling these phenomena depends on the underlying data structures used for the administration of packet arrival.

Binary search trees [1] are widely used in several fields of computer science. Their behavior is well studied, and under certain conditions, e.g., when assuming that the trees are balanced, the average cost of a search operation as a function of the number of nodes can be easily estimated. Nevertheless, traditional BSTs store all the keys. However, there are situations when it is more important to decide whether a key is already in the data structure or not, such as in the case of an Extract Transform Load (ETL) application. Hash tables [1] represent a solution in which the average time complexity of search operation is $O(1)$. However, in so called long running systems, the requirement to keep all the keys for a long period to ensure duplication-free operation, translates to proportionally high storage space requirement, like CHORD [2].

To detect packet loss and duplication, a special, tree-like data structure was proposed, the Interval Merging Binary Tree (IMBT) [3], appropriate for cases when the keys are sequentially ordered by design, or there is a possibility to map the keys onto the natural numbers.

In [4], we performed an analysis of IMBT in a scenario where packets arrive out-of-order according to arrival processes most frequently experienced in practical cases. We applied this data structure over a stream processing framework, like Storm [5].

As part of an additional analysis [6] the full state-space of IMBT has also been determined. In'contrast to a data structure like AVL balanced BST [7] or Red Black balanced tree [8], in the case of IMBT, it is not possible to determine the average complexity of the search operation as the exclusive function of the already stored keys because states are more than one-dimensional, even if the tree is balanced. Therefore, for each number of stored keys, N, we had to give an estimate of the possible number of different average time complexities. The state-space is an exponential function of N. An additional outcome of the analysis revealed that the contingency table [9] is, by design, suitable to classify different statistical input key distributions [10,11]. Therefore by the association of a particular occurrence of the contingency table with N it is possible to determine the average time complexity of the operation in question.

The purpose of this contribution is to give a classification of the average cost of search operation based on the Markov property [12,13] of the process represented by the number of nodes in the tree which depends on the pattern of packet losses/duplications at the input. Since the observed random variables are dependent, the weak dependent related Bernstein-criterion/theorem [14,15] is also considered during the examinations. We use contingency tables to visualize these evolutions.

The paper is divided as follows. In Section 2 we briefly summarize the data processing environment and the main characteristics of IMBT, then we introduce the role of contingency tables. Then in Section 3 we discuss the two main cases, first when the gaps in the originally continuous string of keys are permanent ones, and second when the loss of keys are temporary ones and gaps are gradually filled. In Section 3.1 we introduce the case of permanent gaps and in Section 3.2 we describe the case of temporary gaps. Section 4 contains proofs and deductions related to the formulated theorems. In Section 5 the theoretical results will be compared to existing data structures. Additionally, in this section we introduce the outcome of some selected simulations, which confirm the theories.

2. Interval Merging Binary Tree and Contingency Table

2.1. The Data Processing Environment and the Motivation Behind IMBT

Consider an environment where the group of distributed measuring instruments, say that k instances ($M_1, M_2, \ldots M_k$), emit their measurement reports R_{Mi} periodically, as seen in Figure 1a. Each instrument has its own unique identity. Our goal is to collect, normalize, and transport these measurement reports, along with guaranteed duplication filtering and high-speed (near real-time) processing. To achieve this, first, we apply the following mapping rule: During the first period the R_{M1} report from M_1 is associated with number 1 ($R_{M1} \longmapsto 1$), and subsequently $R_{M2} \longmapsto 2, \ldots, R_{Mk} \longmapsto k$. Supposing that the time of report generation is part of the unique identity, like in case of attribute-based naming (or one can easily extract it from the report), from the second period we apply the mapping $R_{M1} \longmapsto (k+1)$, $R_{M2} \longmapsto (k+2), \ldots, R_{Mk} \longmapsto (k+k)$, from the third period: $R_{M1} \longmapsto (2k+1)$, $R_{M2} \longmapsto (2k+2), \ldots, R_{Mk} \longmapsto (2k+k)$, etc., based exclusively on the unique identity and the time of the report generation as shown in Figure 1b.

Thermometers of the national weather service might be an example of endpoints for such a system.

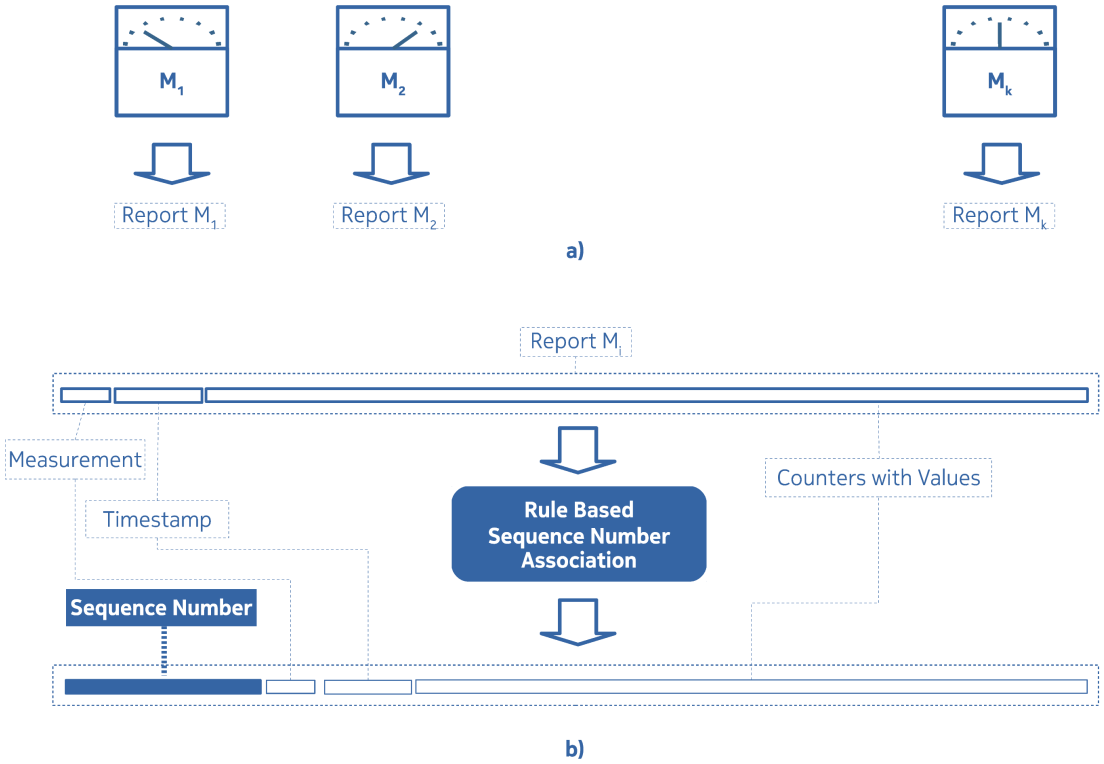

Figure 1. (**a**) Meters with periodically emitted measurement reports. (**b**) Rule-based sequence number association.

In the following, we consider the case when the reporting period might differ instrument group by instrument group. Let us denote by $rp_1, rp_2, \ldots, rp_l$ the different reporting periods in increasing order (longer period is marked by higher index). For the sake of simplicity, suppose that all the longer periods are integer multiples of all the shorter ones. That is, $rp_2 = a \times rp_1$, $rp_3 = b \times rp_1$ and $rp_3 = c \times rp_2$, such that $a, b, c \in \mathbb{N}$. Since $b \times rp_1 = c \times rp_2 \Rightarrow b = ac$. In this case, it is also easy to create a set of substitution rules, aided by the fact that the measurements can be linearized. Under linearization, here we mean such an operation where during the determination of the assignable intervals (contiguous sequence of integer numbers) of measurements with a longer period, we recursively consider that ranges of sequential numbers, whose ranges were assigned before to the shorter periods measurements.

One can think about the linearization as the dimension reduction of two-dimensional data with Hilbert space-filling curve [16].

Examples for such a mixed reporting environments might be the IoT (Internet of Things) reporting entities of modern agro-meteorology systems, where different measurement features require different measurement periods; or a physical experiment, where different measurements, observing the same event, depend on the feasible/available granularity.

The mobile networks can also be considered such a system, in which mixed/heterogeneous measurement periods exist: In the case of a mobile network, ten thousand network elements continuously measure several parameters (signal strength, dropped packages, dropped connections, location updates, etc.) with a different emitting period. In an extensive mobile network, the geographical distribution is also an important factor. Due to its distributed nature, there is no guarantee that during the raw data collection all the raw data will travel in the same order on the same path towards the central repository, even from the same source. Therefore, out-of-order arrival or temporary/final loss and retransmission also might take place. To be able to process the results near real-time, the application of a fast filtering method is essential. Due to the near real-time criterion, a traditional database is out of the question. By the application of a constant time hash table, the filtering speed could be satisfied, however, considering the enormous amount of data it can

become a bottleneck from a memory consumption point of view. To reduce the memory consumption and increase the throughput of a processing system that operates in the above described/characterized environment, the IMBT is introduced, as seen in Figure 2.

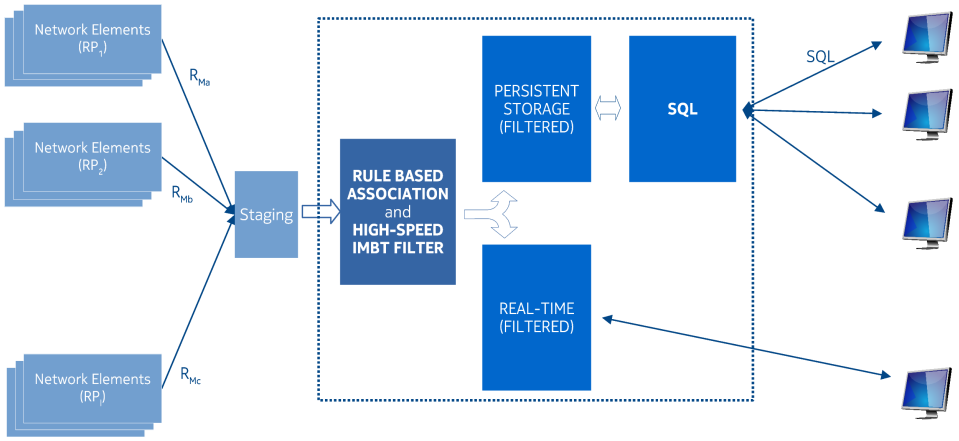

Figure 2. Mobile network: an example for mixed measurement periods.

In an idealized environment, when all the reports eventually arrive, as we shall see, the height of the tree is stabilized around an expected result. Therefore, the time complexity of the average search operation can be considered to be constant, that is, $O(1)$.

However, the measuring environments are usually far from ideal. Therefore, for instance, due to the failure of a network element, the measurement reports might not have been generated at all, or the reports are lost eventually over the transmission, and final gaps appear in an ideally continuous interval, which causes a variously increasing average cost of the search operations.

2.2. Interval Merging Binary Tree

We denote by interval a sequence of contiguous integer numbers, where the numbers represent the keys, ordinal numbers counting a sequence of data packets received by a host in the network. Interval Merging Binary Tree is such a binary tree, in which, nodes are composed of nonoverlapping intervals that are not neighbors of each other. If we are only interested in if a certain packet arrived or not, it is enough to store only the extremes of the intervals instead of storing all the keys, as seen in Figure 3.

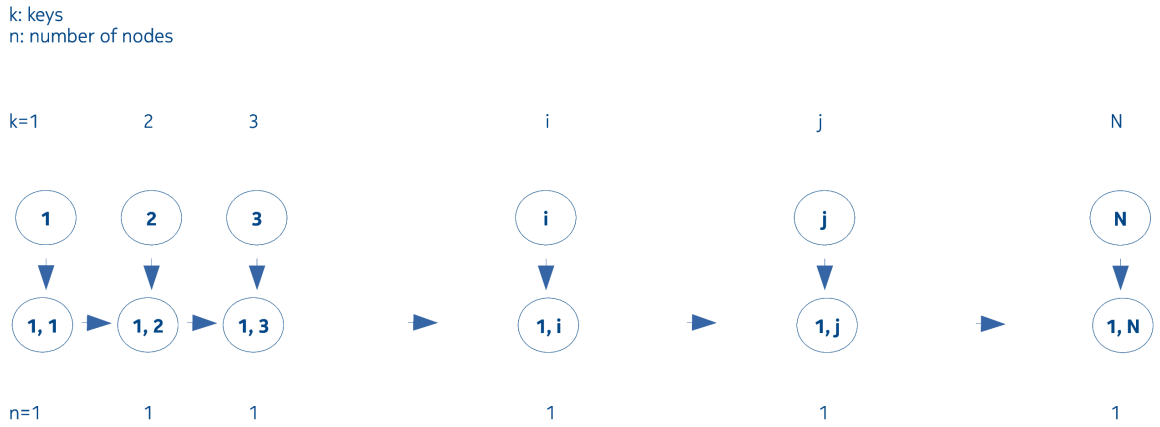

Figure 3. Interval Merging Binary Tree (IMBT) number of keys increasing and the interval evolving while the number of nodes is constant.

From the perspective of storage requirements, IMBT outperforms other data structures, storing all the keys if the average number of keys covered by the nodes of the tree is higher than 2. An additional data structure is required if we want to make an operation on the values associated to the individual keys, once it is decided, based on IMBT, whether a packet has arrived already, because IMBT reduces the space of keys to a space of intervals.

To count the possible arrangements of intervals we need to assess the number of distinct ways a number N of nodes can be partitioned into the sum of integers smaller than N. In the field of integer partitioning [17] the Hardy–Ramanujan number [18] represents an upper estimate for that:

$$\lim_{N\to\infty} p(N) \approx \frac{1}{4N\sqrt{3}} e^{\pi\sqrt{2N/3}} \tag{1}$$

Now let us consider the fixed number of different keys N. Assuming that all the keys might be equally searched for, if N was stored in a traditional balanced binary search tree, then the average cost of a search operation can be expressed by the following formula:

$$A(N) = log_2(N) \tag{2}$$

However, from Equation (1) it is clear that, by knowing only the already stored keys (that is N), we can get $p(N)$ number of different combination of nodes, regarding interval lengths. We have not yet taken into account the effect of balancing either. Let us consider a simple example in which $N = 4$ and let us denote the individual keys by k_1, k_2, k_3, k_4. Based on the possible number of neighbors (neighborship being equivalent to the fact that the second packet is the immediate next packet to the first one), the following scenarios can be distinguished:

- None of the keys are neighbors of each other,
- Two of them are neighbors and the other two are not,
- Two of them are neighbors and the remaining two are neighbors as well,
- Three of them are neighbors and one is not,
- All the keys are neighbors of each other.

We can see that these scenarios actually translate to the integer partitions of N when $N = 4$: $N = 1+1+1+1$, $N = 2+1+1$, $N = 2+2$, $N = 3+1$ $N = 4$; that is $p(N = 4) = 5$ in this case. Three out of five scenarios are shown in Figure 4. Figure 4a corresponds to $k_1 = 1, k_2 = 3, k_3 = 5, k_4 = 7$. Figure 4b corresponds to $k_1 = 1, k_2 = 2, k_3 = 3, k_4 = 4$; while Figure 4c corresponds to $k_1 = 1, k_2 = 3$, $k_3 = 2, k_4 = 5$ arrival patterns. N represents discrete time of arrival of the packets. The remaining axes display the interval length of each node at a given time instant, represented by N.

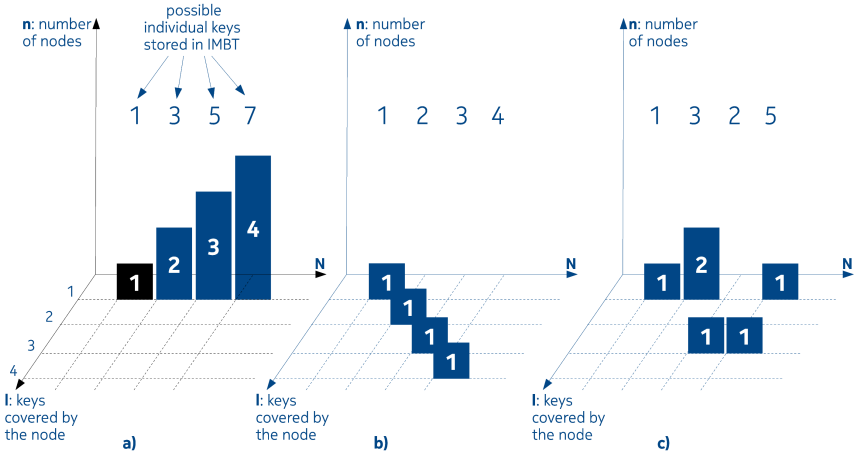

Figure 4. IMBT examples of evolving types for $N = 1 \ldots 4$. (**a**) No neighbors, (**b**) all keys are neighbors, (**c**) three keys are neighbors, one key is not.

In [6] we have proven that these arrangements with or without balancing can be mapped onto the rows and columns of a contingency table in the following way.

2.3. The Role of Contingency Tables on the Analysis of IMBT

Let us take a snapshot of IMBT and sort the intervals according to their length in monotone increasing order. Let n be the number of nodes in the IMBT. Let us denote by $l_i \in L$ the length of the intervals belonging to a node n_i from the IMBT, where L is a multi-set. We can denote by j the number of distinct interval lengths. To all j we can assign a number $d(j)$ that is the number of intervals with the same length associated to j. It is necessarily true that $j \leq n$. These numbers $d(j)$ are located in the header of the contingency table, that is, on the top of each column.

Now let us sort the intervals according to the distance of the nodes associated to them from the root node in a monotonic increasing order. Let us denote by $s_i \in S$ the number of comparisons required to reach the left hand value of an arbitrary node n_i, where S is a multi-set. The previously introduced s_i is the distance. Then let us create as many w_i, $i \in 1 \ldots k$ for as many distinct distances there are. Finally let us assign to all w_i a number $d(w_i)$ meaning the number of intervals/nodes with the same distance from the root node. It is necessarily true that $k \leq n$. These $d(w_i)$ numbers are located in the right side of the contingency table, that is in the end of each row.

Two examples are presented in the following. First we represent a traditional, completely balanced BST in a contingency table, Figure 5.

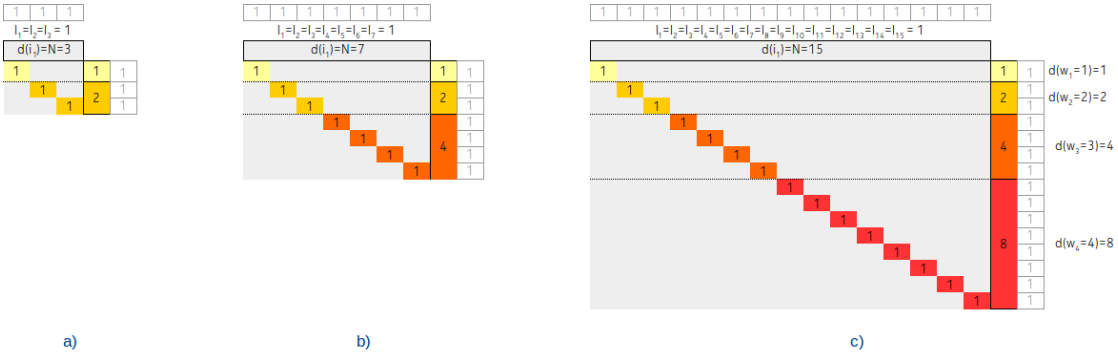

Figure 5. The balanced binary search tree related contingency tables. The (**a**–**c**) cases belong to the three different N values: 3, 7, and 15.

In the second example, we show an IMBT and the related contingency table. For the sake of simplicity, suppose that there is a completely balanced IMBT with seven nodes, and the lengths of all the intervals are the same. That is, with the above notation: $l_1 = l_2 = l_3 = l_4 = l_5 = l_6 = l_7 = a$. See Figure 6.

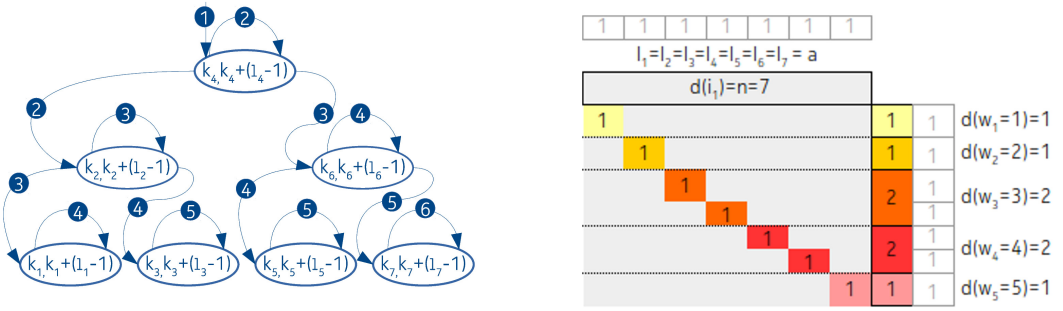

Figure 6. IMBT with seven nodes and the related contingency table. The arrows and the filled circles are marking the order and number of comparisons during the search operations.

As can be seen in the figure, due to the equal length intervals, there is only one j and the number of intervals with the same length associated to it is $d(j) = 7$. Of course, as we shall see, there are many arrangements where the interval lengths are different.

We can recognize two deviations between the contingency tables associated to BSTs and IMBTs. First, in the header of the contingency table belonging to the traditional BST, $N = n$ number of keys appear. However in case of IMBT, the number of nodes n is present, since this number is not necessarily equivalent with the number of keys N covered by the tree so far. The second deviation is that although both trees are balanced, the number of nodes with the same distance from the root in case of BST follows the series of powers of two, but in case of IMBT it is a combination of a Fibonacci sequence and an additional term, as shown in [6].

3. Arrangements Related Conditions, Theorems, and Equations

By now we know a model in which we can count the average cost of search operation by simple multiplication of the corresponding values in the contingency table.

In the following, we will define some metrics and through these we will examine the evaluation of the contingency table.

Let us denote by N the number of keys stored in the IMBT so far just like above. However, unlike above, to be able to examine the evolution of the number of nodes in the tree, we introduce a random variable, V_i, which is the instantaneous number of vertices (nodes previously) in the tree at time instant i, where $i \in 1 \dots N$. It is obvious from the definition that $V_1 = 1$ and V_i can be mapped to states in a stochastic matrix. The distribution of the lengths of the intervals is affected by the homogeneity and the finite/infinite nature of the stochastic matrix.

We define the series of instantaneous average interval lengths by the following formula:

$$\bar{L}_i = \frac{l_1^i + l_2^i + \dots + l_{V_i}^i}{V_i}, \tag{3}$$

where i is the time instant ($i \in 1 \dots N$) and l_k^i is the length of the interval stored by node k at time instant i. $\bar{L}_i$ is a random variable as well.

We assume that for the series of $\bar{L}_i$ the following constraints are true:

- There is an expected value a, to which the individual random variables, a_i, stochastically converge to as N tends to infinity, where $a = \lim_{N \to \infty} (a_1 + a_2 + \dots + a_N)/N$.
- There is a $c = (\sigma_1^2 + \sigma_2^2 + \dots + \sigma_N^2)/n$ independently from N, where σ_i is the standard deviation of the interval lengths at time instant i.
- There is a non-negative function $r(x)$ for which $r(0) = 1, \lim N \to \infty (r(1) + r(2) + \dots + r(n))/n = 0$, and additionally $|corr(\bar{L}_i, \bar{L}_i)| \leq r(|i - j|), i, j \geq 1$.

The conditions mentioned above together constitute the Bernstein-theorem [14]. According to the theorem, if the three constraints are simultaneously met, then the weak law of large numbers is true.

Using the Bernstein theorem as a starting point, we can identify two types of completely different input pattern classes, for which the behavior of contingency tables is examined and the cost of average search operation is determined:

- In the first case only a nonrecurring, transient, infinite state stochastic matrix can be composed based on the associated states V_i. Additionally we assume that the Bernstein-theorem is true for the series of $\bar{L}_i$,
- In the second case, we assume that based on the state V_i, it is possible to create a stochastic matrix which has a finite state-space, is aperiodic, irreducible (that is ergodic), and recurrent.

The satisfaction of the first criterion implies that the average interval length is upper bounded. As a result, the variance of the interval lengths is also upper bounded, therefore, the scenario in which we have a composition of a large interval with continuously increasing length with increasing number of small ones, is not valid. Rather, as N increases, there will be an increasing number of gaps between the intervals, associated with permanently missing keys, that is, keys where the probability of arrival of the associated packet converges to zero. Both from the previous fact and from Equation (3) it is obvious that V_i is proportionally increasing as well.

The satisfaction of the second criterion implies the presence of temporary gaps only: in spite of the increasing N the finite number of nodes implies that the instantaneous average interval length is increasing, that is the number of gaps is upper bounded.

3.1. Permanent Gaps

In the case of permanent gaps, the mean of the average interval length is a constant value, a. We do not exclude the possibility of having temporary gaps, due to the out-of-order arrival of packets. As a result, the header $d(i_i)$ in the associated contingency table will follow a kind of distribution. Taking into account the effect of temporary gaps in the analysis would make the analysis more complex, but their effect is minor, therefore they will be discarded in the subsequent.

3.1.1. Linked List Arrangement

In this realization, our additional assumption against the keys is that there is a smallest one.

The tree degenerated into a linked list and three associated contingency tables with $V_i = n = 3$, $V_i = n = 7$ and $V_i = n = 15$ are shown in Figure 7.

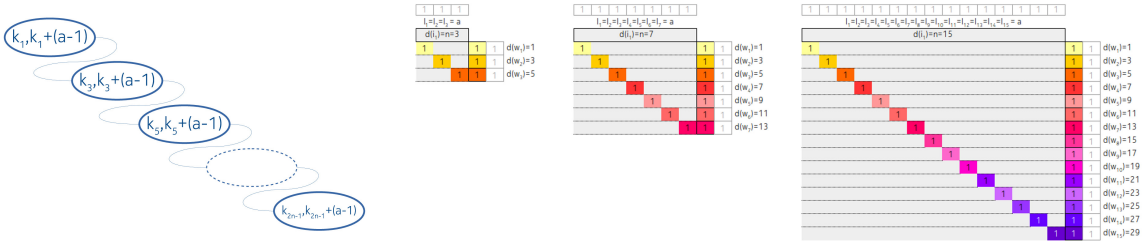

Figure 7. Linked list degenerated IMBT and three associated contingency tables.

Theorem 1. *In the case when there is no shuffling and no balancing at all, the tree degenerates to a linked list and*

$$A(N,a) = \frac{N}{a} + \frac{a-1}{a}. \tag{4}$$

From the contingency table, it is clearly visible that the w_i follows the sequence of odd numbers and $d(w_i)$ remains constant.

3.1.2. Completely Balanced Arrangement

We assume a completely balanced tree with the number of node power of 2, that is $n = 2^l - 1$. Three associated contingency tables with $V_i = n = 3$, $V_i = n = 7$ and $V_i = n = 15$ are shown in Figure 8.

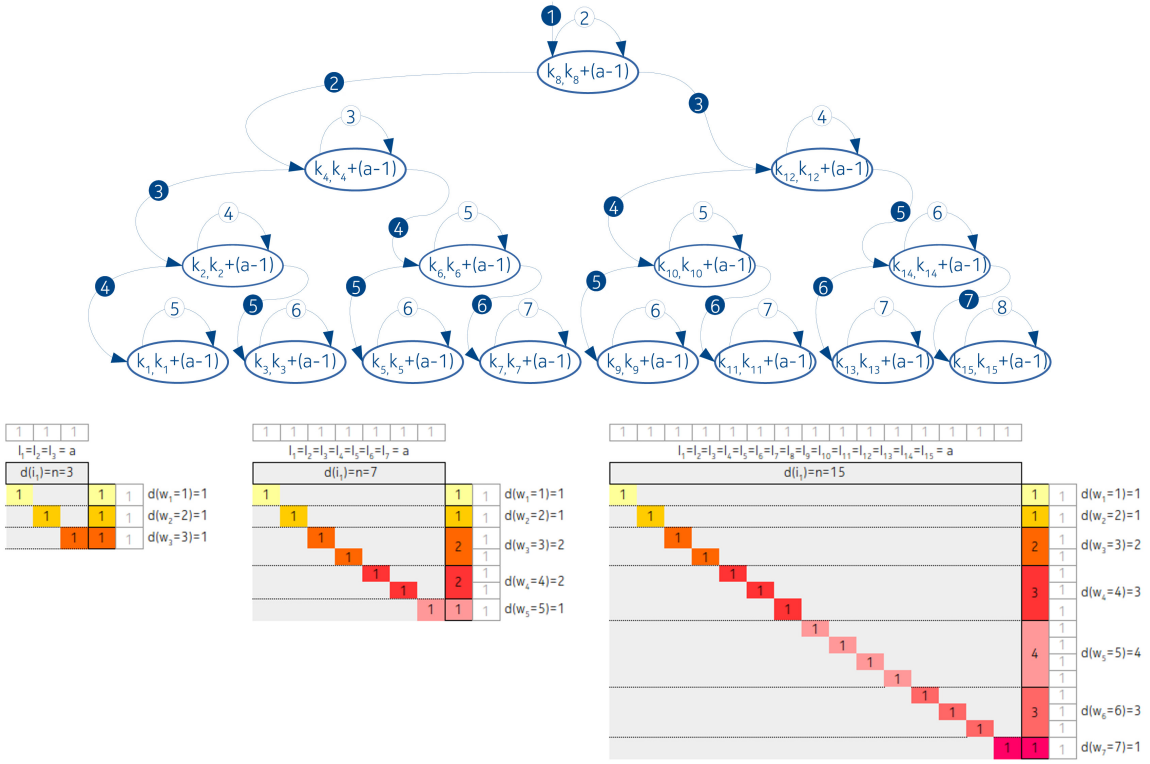

Figure 8. Completely balanced IMBT and three associated contingency tables.

Theorem 2. *With the above conditions, the average cost of the search operation can be expressed with the following formula:*

$$A(N,a) = \frac{3}{2} log_2\left(\frac{N}{a}+1\right) + \frac{3a}{2N} log_2\left(\frac{N}{a}+1\right) - \frac{a+1}{a}. \tag{5}$$

As is visible from the figure and as we have proven in [6], the balancing has a typical fingerprint in the $d(w_i)$ distribution: It follows the Fibonacci sequence until the middle of the rows on the way from the top rows to the bottom rows.

Supposing that $N >> 0$, and $a > 0$. Then we can apply the following simplification on Equation (5):

$$A(N,a) \approx \frac{3}{2}\log_2(\frac{N}{a}) = \frac{3}{2}\log_2(N) - \frac{3}{2}\log_2(a) \tag{6a}$$

$$= \log_2(N) + \frac{1}{2}\log_2(N) - \frac{3}{2}\log_2(a). \tag{6b}$$

That is, an IMBT with the given criteria will outperform a BST as long as the difference of the second and third term is negative, in other words, provided that:

$$a > \sqrt[3]{N} \tag{7}$$

3.2. Temporary Gaps Only

Let us assume in the following cases that it is possible to compose from V_i a finite state $\{1 \dots p\}$, ergodic (aperiodic, positive recurrent), at least partially recurring stochastic matrix.

Let us denote by M_j the mean recurrence time in state j. If M_j is finite then state j is positive recurrent. The state j in which the expected return time is the smallest one, will represent the number of nodes in the IMBT as a steady-state value. Therefore we can substitute that state with value $n = j$.

Regarding the distribution of the interval lengths, which are necessarily increasing, we will distinguish two possible realizations.

- In the first realization we suppose that the increasing interval lengths are uniformly distributed to nodes, the number of which is fixed.
- In the second realization the distribution is not uniform.

3.2.1. Linked List Like Arrangement

In this subsection, our additional assumption against the keys is that there is a smallest one, which we may always call first and as time goes by the probability that all the keys near the first key have already arrived is increasing.

Since the gaps are temporary ones, out-of-order arrival implies there are keys which arrive late, therefore temporary side-branches might appear over time, outside of the main branch. The length of these temporary branches depends on the statistics of the out-of-order arrival pattern. Subsequent side-branches are not taken into account because their effect is marginal.

Theorem 3. *With the distribution characterized above, the data structure degenerates into a linked list. Suppose that the increasing lengths are uniformly distributed across the nodes. Then, due to the uniformly distributed increasing lengths, the average cost of the search operation does not depend on N:*

$$A(N, a) = n. \tag{8}$$

Figure 7 accurately describes this scenario as well.

In the following, we suppose that the distribution of the lengths is not uniform, but node-heavy, meaning that every node contains a single key only, except one, which contains all the remaining $N - n + 1$ keys. That heavy node can reside at the tail, in the middle, or at the root of the linked list degenerated tree.

Theorem 4. *Then the average costs of search operations in case of node-heavy arrangements are the followings:*

$$\lim_{N\to\infty} A_{tail\ heavy}(N, a) = 2n. \tag{9a}$$

$$\lim_{N\to\infty} A_{middle\ heavy}(N, a) = n. \tag{9b}$$

$$\lim_{N\to\infty} A_{root\ heavy}(N, a) = 2. \tag{9c}$$

The related arrangements and contingency tables are shown at Figures 9 and 10 respectively.

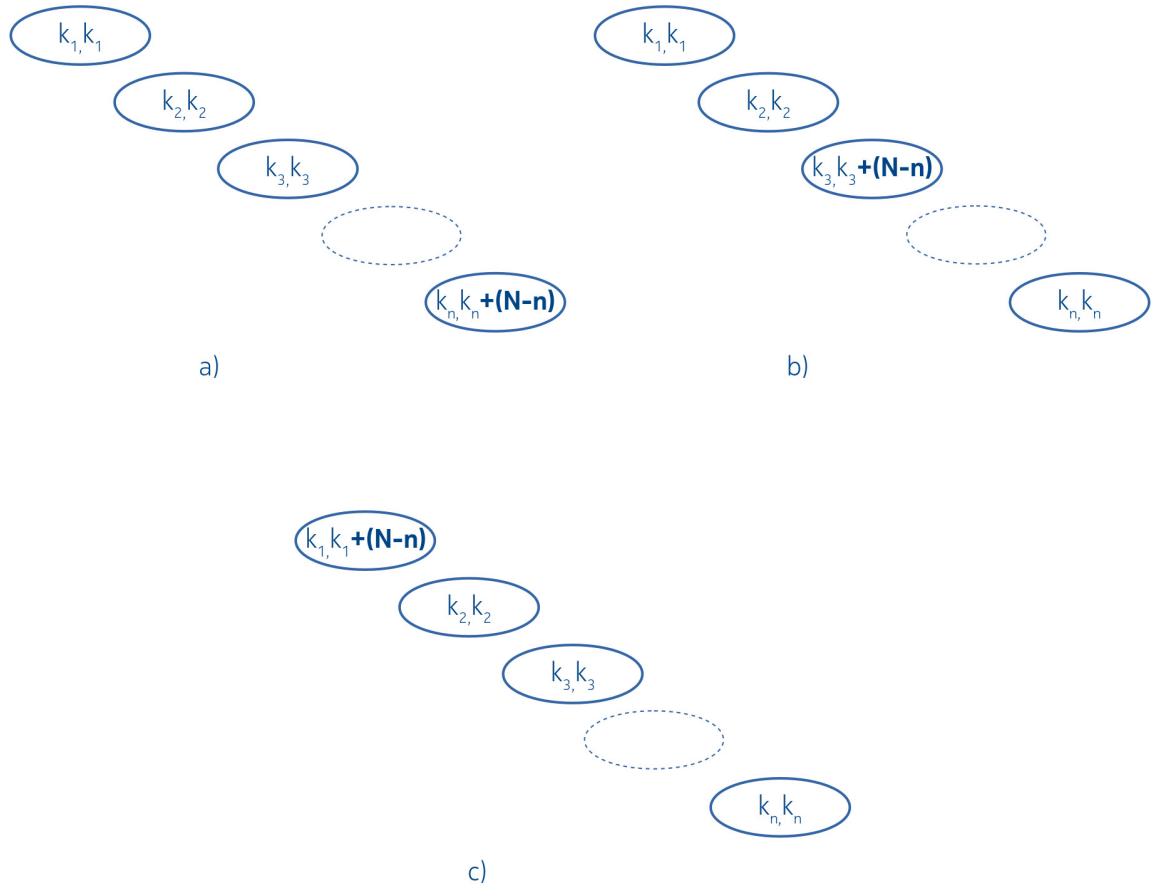

Figure 9. The linked list degenerated IMBT with heavy nodes. (**a**) is tail heavy, (**b**) is middle heavy and (**c**) is root heavy IMBT.

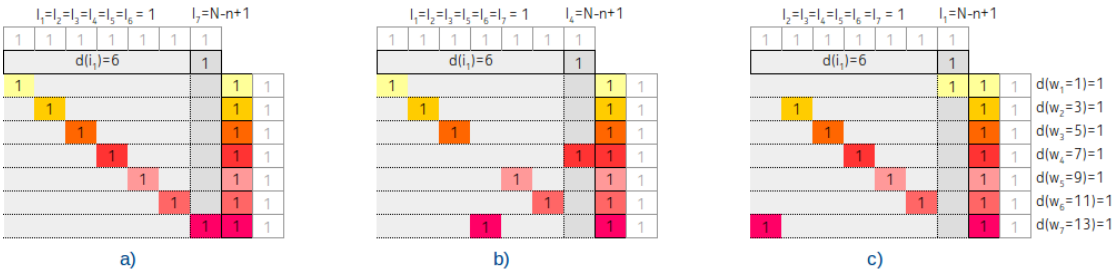

Figure 10. The associated contingency tables of linked list degenerated IMBT with heavy nodes. (**a**) is tail heavy, (**b**) is middle heavy and (**c**) is root heavy contingency table.

3.2.2. Completely Balanced Arrangement

Two scenarios are considered:

- The increasing lengths of intervals are uniformly distributed across the tree,
- The distribution of lengths follows an exponential distribution.

The first case is fairly simple. Since the number of nodes is fixed with value n, it is easy to see the following.

Theorem 5. *considering a completely balanced IMBT the average cost of search operation is such a constant, which is proportional with the logarithm of n. Based on Equation* (6b)*, considering that a* $>>$ 0 *(since a increasing infinitely) we get:*

$$A(N,a) \approx C(n) - 1. \tag{10}$$

From the formula it is visible that, just like in case of Theorem 3, the $A(N, a)$ is independent from N.

The other case is a little bit trickier: it depends on the length distribution. Suppose that the nodes with smaller key values hold the longer intervals, while nodes with the highest key values hold the shorter intervals. Additionally, the rightmost interval is always one. Compensating this constraint without increasing the number of nodes the leftmost interval always absorbs the surplus.

We do not give the related formula and the associated deduction here, but consider the construction of a similar, however, "more realistic" arrangement in the next subsection.

3.2.3. An Exception: Completely Balanced Arrangement, Temporary Gaps, Infinite Nodes, and Increasing Average

In this subsection, we introduce an arrangement, where none of the two criteria from Section 2 hold: the average length of the intervals is increasing, along with the increasing number of nodes. Our initial assumption is that the interval lengths are exponential according to power of two. Additionally the longest interval has the smallest left-hand key value, the second longest interval has the second smallest left-hand key value and so on. Moreover we suppose that the length of the shortest interval is always 2^0.

Since we are examining asymptotic results we apply the simplification that only the right side distances will be taken into account for the determination of the full weight of IMBT. That is, if the length of an interval is $l_i (= 2^i)$ then, with this approach we weight the right distances of a node by the full l_i, instead of $l_i - 1 (= 2^i - 1)$. However, it is easy to see that as $N \to \infty$ this difference becomes insignificant.

By considering a tree nodes arrangement and applying the above constraints we obtain interval lengths of $2^0, 2^1, 2^2$. As we stipulated before, the longest interval has the smallest left key value. Therefore, in a balanced IMBT 2^2 interval has the distance from the root 3 comparisons (we take into account the right hand distances). The 2^1 interval is the root node, therefore, we count with 2 comparisons. The 1 key interval is in the right side of the balanced IMBT, therefore to reach the majority of that keys requires 4 comparisons.

As we stated before, during the calculations we assumed that the following conditions are hold:

- Any key can be the subject of the search operation with equal probability,
- The key is already in the tree.

Therefore, during the determination of the average cost of search operation, the actually expected value of comparisons is calculated. According to our assumption the probability of we are looking for key k is $1/N$, where $k \in \{1 \dots N\}$. Since during the *jth* comparison several keys can be found the *jth* comparisons has to be weighted with the length of the intervals. According to the above mentioned the $A(N)$ average cost is $1/N$ multiplied by the sum of weighted nodes, where a particular weight, which belongs to a single node is the multiplication of the distance and the length of the interval. The sum of weighted nodes, that is the total weight, is denoted by TW.

Let us assign the s_1, s_2 and s_3 to the above numbers, respectively. That is, $s_1 = 3, s_2 = 2$ and $s_3 = 4$. The approximate value of the total weight of the tree is the following:

$$TW = s_1 2^0 + s_2 2^1 + s_3 2^2 \tag{11}$$

Now by extending our examination to a $n = 7$ nodes arrangement, the longest interval is 2^6. The comparison weights depend on the lengths and the distances from the root, therefore in this case we obtain:

$$\begin{gathered} TW = (s_1 + 2)2^0 + (s_2 + 2)2^1 + (s_3 + 2)2^2 + (s_1 + 1)2^{0+4} + (s_2 + 1)2^{1+4} + \\ (s_3 + 1)2^{2+4} + (s_2 + 0)2^3 \end{gathered} \tag{12}$$

From the above two equations we can formulate the recursive extension/composition rule: Take the given expression which is valid for n nodes. To determine the $2n+1$ nodes arrangement, first copy the whole formula and increase by 2 the multipliers of the powers. Then add the formula with the multipliers increased with one and powers increased by 2. According to modification 2, increase the multiplier values by 1. Additionally, add the corresponding base 2 value to the exponents. Finally add the missing new root member to the expression. To make it more understandable, here we give an extension of Equation (12).

$$\begin{aligned} TW = {} & (s_1+2+2)2^0 + (s_2+2+2)2^1 + (s_3+2+2)2^2 + \\ & + (s_2+0+2)2^3 + (s_1+1+2)2^{0+4} + (s_2+1+2)2^{1+4} + (s_3+1+2)2^{2+4} + \\ & + (s_1+2+1)2^{0+8} + (s_2+2+1)2^{1+8} + (s_3+2+1)2^{2+8} + \\ & + (s_2+0+1)2^{3+8} + (s_1+1+1)2^{0+4+8} + (s_2+1+1)2^{1+4+8} + (s_3+1+1)2^{2+4+8} + \\ & + (s_2+0+0)2^7. \end{aligned} \tag{13}$$

This is such a recursive rule, that it affects both the multipliers and the exponents. The related contingency table is shown in Figure 11. In the figure, we indicated the number of steps that are required to achieve the interval opening keys, instead of the closing. That is, every weight associated values in the column are shifted by one.

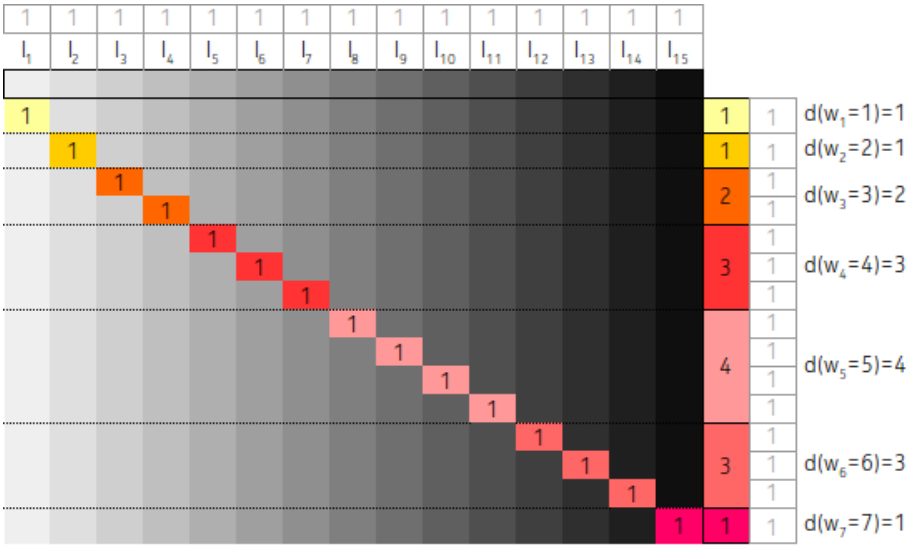

Figure 11. The contingency tables of IMBT where all the interval lengths are different.

Based on the figure and Equation (13) we can say that

$$\begin{aligned} & d(w_1) = 1, \; where \; w_1 = (s_2+0+0) = 2 \\ & d(w_2) = 1, \; where \; w_2 = (s_2+0+1) = 3 \\ & d(w_3) = 2, \; where \; w_3 = (s_2+0+2) = (s_2+1+1) = 4 \\ & d(w_4) = 3, \; where \; w_4 = (s_2+1+2) = (s_2+2+1) = (s_1+1+1) = 5 \end{aligned} \tag{14}$$

etc.

Since the average length of the intervals is strictly tied to N, the average cost of search operation is depending exclusively on N, that is

$$A(N) = \frac{1}{N} \times TW_N. \tag{15}$$

4. Proofs and Deductions

In this section we give the proofs and deductions of the above results and theorems. During the deductions, it is assumed that any key has the same probability to be searched for. That is $P(the\ key\ we\ are\ looking\ for\ is\ k_m) = \frac{1}{N}, where\ m \in (1 \dots N)$.

4.1. Permanent Gaps

4.1.1. Linked List Arrangement

Proof of Theorem 1: Every node contains two keys, since a node covers an interval and the keys represent the borders of that particular interval. Let us assume that a node in the tree covers a keys on average. Then we can denote by k_i the starting key and by $k_i + (a-1)$ the ending key. Additionally, due to the non-overlapping feature of IMBT, it is also trivial that $(k_i + a - 1) < k_{i+1}$.

First let us see the $a = 1$ case. In this case in the linked-list degenerated data structure the starting and the ending keys are equal. As a consequence: if the key to be searched for is not equal with k_i then the second comparison with the right value of that particular node is necessary but, due to $k_i == k_i + a - 1$, the outcome of the comparison is always false.

Since the number of the nodes is $n = N/a$, and here $a = 1$, therefore $n = N$. Based on this, we can write that the average cost of SEARCH is equal to the expected value:

$$\begin{aligned} A(N) &= \frac{1}{N}\sum_{i=1}^{n}(2i-1) \\ &= \frac{1}{N}\Big[2\frac{n(n+1)}{2} - n\Big] = N. \end{aligned} \tag{16}$$

Now, we examine the $a \geq 2$ case. It is still valid that $n = N/a$. Analogous to the previous deduction we can write that the average cost of the SEARCH operation is:

$$\begin{aligned} A(N,a) &= \frac{1}{N}\sum_{i=1}^{n}(2i-1) + 2i(a-1) \\ &= \frac{1}{N}\Big[n(n-1) - n + (a-1)n(n-1)\Big] \\ &= \frac{N+a-1}{a} = \\ &= \frac{N}{a} + \frac{a-1}{a}. \end{aligned} \tag{17}$$

It is visible that substituting 1 into a we will return Equation (16).

An additional consequence is that as a grows the equation tends to:

$$\frac{N}{a} + \frac{a-1}{a} \approx \frac{N}{a} + 1. \tag{18}$$

With the deduction above, Theorem 1 is proved. □

4.1.2. Completely Balanced Arrangement

From now on, the number of layers or levels is denoted by l (in contrast to the previous notation of lengths). The root node is the $l = 1$.

Proof of Theorem 2: Based on our notations we can write that:

$$\begin{aligned} n &= \frac{N}{a}, \\ l &= log_2(n+1) = log_2(\frac{N}{a}+1). \end{aligned} \tag{19}$$

During the deduction we will use the following identity:

$$\sum_{i=1}^{n} i2^i = n(2^{n+1}-2) - (2^{n+1}-4) + (n-1)2. \tag{20}$$

Based on the relations (Equation (19)) we can define the layer level sums:

$$2^{i-1} + 2^i(a-1) + (i-1)2^{i-2}3 + (i-1)2^{i-2}3(a-1), \tag{21}$$

where i means the i^{th} level in the tree. However, this form is not suitable for equal transformations. Therefore, we split the formula into a fixed member which is the first node and the layer level members. Due to this split we will use an incremented i and the counter in the sum will last to $l-1$ instead of l:

$$\begin{aligned} A(N,a) =& \frac{1}{N}\Big[1 + 2(a-1) + \sum_{i}^{l-1} 2^i + (a-1)2^{i+1} + 3i2^{i-1} + 3(a-1)i2^{i-1}\Big] = \\ =& \frac{1}{N}\Big[1 + 2(a-1) + \sum_{i}^{l-1} 2a2^i - 2^i + \frac{3a}{2}i2^i\Big] = \\ =& \frac{1}{N}\Big[1 + a - a2^l - 2^l + \frac{3a}{2}l2^l\Big]. \end{aligned} \tag{22}$$

Since

$$\begin{aligned} a - a2^l =& -a(2^l - 1) = -N \\ 1 - 2^l =& -\frac{N}{a} \end{aligned} \tag{23}$$

we can write that

$$\begin{aligned} A(N,a) =& \frac{1}{N}\Big[\frac{3a}{2}l2^l - N - \frac{N}{a}\Big] \\ =& \frac{3a}{2N}(\frac{N}{a}+1)log_2(\frac{N}{a}+1) - 1 - \frac{1}{a} \\ =& \frac{3}{2}log_2\Big(\frac{N}{a}+1\Big) + \frac{3a}{2N}log_2\Big(\frac{N}{a}+1\Big) - \frac{a+1}{a}. \end{aligned} \tag{24}$$

With the deductions above, Theorem 2 is proved. □

4.2. Temporary Gaps Only

4.2.1. Linked List Arrangement

The proof of Theorem 3 is easily derivable from Equation (18), with the substitution of $n = \frac{N}{a}$:

$$A(N,a) = \frac{N}{a} + \frac{a-1}{a} \approx n + 1 = C. \square \tag{25}$$

The proofs of node heavy cases, Theorem 4, are the following.

According to arrangement *a)* we can write that $n = N/a$, and

$$\begin{aligned} A(N,a) &= \sum_{i=1}^{n} \frac{2i-1}{N} + 2n\frac{N-n}{N} \\ &= N\Big[\frac{2a-1}{a^2}\Big] = n\frac{2a-1}{a}. \end{aligned} \tag{26}$$

It is visible from the results that as N and a grow along with $n = const.$ the value tends to $2n$, which is not surprising considering that accumulation is possible merely in the last element.

Considering arrangement *b)* we will get that

$$\begin{aligned} A(N,a) &= \sum_{i=1}^{n} \frac{2i-1}{N} + \frac{n+1}{2}\frac{N-n}{N} \\ &= \frac{1}{2}\frac{N+a-1}{a}. \end{aligned} \tag{27}$$

Comparing the result to Equation (25) we can see that the average SEARCH cost is half of that of the uniformly distributed one.

Arrangement *c)* can be expressed by the following equation:

$$\begin{aligned} A(N,a) &= 2\frac{N-n}{N}\sum_{i=1}^{n}\frac{2i-1}{N} \\ &= 2\frac{N-n}{N} + \frac{2}{N}\sum_{i=1}^{n} i - \frac{(n-1)}{N} \\ &= 2\frac{N-n}{N} + \frac{2}{N}\Big(\frac{n(n+1)}{2}\Big) - \frac{(n-1)}{N} \\ &= 2\frac{N-n}{N} + \frac{n(n+1)}{N} - \frac{(n-1)}{N} \end{aligned} \tag{28}$$

The result shows that by increasing N, next an $n = const.$, $A(N,a)$ tends to 2:

$$lim_{N\to\infty} A(N,a) = 2. \tag{29}$$

The proof of Theorem 4 is completed. □

4.2.2. Completely Balanced Arrangement

The Theorem 5 can be easily proven based on Equation (6b):

$$A(N,a)\frac{3}{2}log_2\Big(\frac{N}{a}+1\Big) + \frac{3a}{2N}log_2\Big(\frac{N}{a}+1\Big) - \frac{a+1}{a} \tag{30}$$

By replacing $\frac{N}{a}$ with n, and considering that during the examinations $n = const.$:

$$\begin{aligned} A(n = const., a) &= \frac{3}{2}log_2\Big(n+1\Big) + \frac{3}{2n}log_2\Big(n+1\Big) - \frac{a+1}{a} \\ &= c_1 + c_2 - \frac{a+1}{a}. \end{aligned} \tag{31}$$

As $a >> 0$ (since a is increasing infinitely) we get the following approximation:

$$A(N,a) \approx C(n) - 1. \square \tag{32}$$

During the proofs above, we restricted our examinations to such cases where the requested key has already stored in the data structure. Of course the missing keys would modify the results:

the interval length of the gaps should be considered, instead of the interval length of the nodes. Here, the weight of the half open intervals should be handled by care.

5. Evaluations and Simulations

Above, we analyzed the average time complexity of search operation of IMBT considering various input patterns. In the following, to put the IMBT in context, first we compare the cost of the search with the case of using other data structures in wide use. Throughout the comparisons, we consider the effect of permanent gaps and temporary gaps which, as we shall see, play a role only in the case of IMBT.

Then, we present the results of the selected simulations. In this case, we can point out hidden gains caused by the reduced number of nodes. These reduce, for instance, the cost of a tree rotation.

5.1. Evaluations

We compare IMBT with AVL balanced BST, hash tables, and Splay trees [19]. AVL was selected because AVL-based balancing has been applied by us for IMBT as well in our simulations. Hash was selected since its average time complexity of search operation is a constant time. Splay tree was chosen because it has a self-balancing structure influenced by the access pattern.

5.1.1. Permanent Gaps

Table 1 compares the performance of IMBT (balanced—IMBT-B, linked list degenerated—IMBT-LLD) with data structures, in case there are permanent gaps.

Table 1. Evaluation of time and space complexity of data structures in case of permanent gaps.

	Number of Keys	**Number of Nodes**	**Space Complexity**	**Search Operation Time Complexity (Average)**	**Search Operation Time Complexity (Worst Case)**
AVL Balanced BST	N	$n = N$	$O(N)$	$O(log(N))$	$O(log(N))$
Splay tree	N	$n = N$	$O(N)$	$O(log(N))$	$O(log(N))$
Hash	N	$n = N$	$O(N)$	$O(1)$	$O(N)$
IMBT-LLD	N	$n = \frac{N}{a}$	$O(\frac{2}{a}N) = O(N)$	$O(\frac{N}{a} + \frac{a-1}{a}) = O(N)$	$O(\frac{2}{a}N) = O(N)$
IMBT-B	N	$n = \frac{N}{a}$	$O(\frac{2}{a}N) = O(N)$	$O(\frac{3}{2}\log_2(\frac{N}{a})) = O(log(N))$	$O(2 \times \log_2(\frac{N}{a})) = O(log(N))$

As we can see from the table, the permanent gaps reduce the IMBT-LLD to a linked list with reduced number of nodes. The only gain here might be the constant divisor factor. In case a is high enough, the space complexity is a fraction of all other data structures.

Not surprisingly, IMBT-LLD has the worst performance on search time of all the data structures. For IMBT-B even the worst case has O(log(N)) time complexity, while a gain can originate from the reduced space complexity which can be significant, depending on the value of a.

5.1.2. Temporary Gaps Only

Table 2 compares the performance of IMBT (balanced—IMBT-B, linked list degenerated—IMBT-LLD) with data structures, in case there are temporary gaps. From the three variants of IMBT-LLD, only the middle node-heavy variant will be covered.

Table 2. Evaluation of time and space complexity of data structures in case of temporary gaps.

	Number of Keys	Number of Nodes	Space Complexity	Search Operation Time Complexity (Average)	Search Operation Time Complexity (Worst Case)
AVL Balanced BST	N	$n = N$	$O(N)$	$O(log(N))$	$O(log(N))$
Splay tree	N	$n = N$	$O(N)$	$O(log(N))$	$O(log(N))$
Hash	N	$n = N$	$O(N)$	$O(1)$	$O(N)$
Middle node-heavy IMBT-LLD	N	$n = const.$	$O(2n = const.) = O(1)$	$O(n = const.) = O(1)$	$O(2n = const.) = O(1)$
IMBT-B	N	$n = const.$	$O(2n = const.) = O(1)$	$O(C(n = const.) - 1) = O(1)$	$O(C(n = const.) - 1) = O(1)$

The stochastically constant height of IMBT leads to stochastically constant search cost and constant space complexity.

Therefore, where it is possible, it is worth it to apply artificial maintenance insertion operations to eliminate the permanent gaps. Of course this is an application-specific decision, when, for instance, the user is aware that a reporting instrument failed, a range of keys surely would never arrive to the tree, otherwise the artificially inserted values override the real ones. Through a well-designed maintenance insertion the height of the tree probably can be kept under a certain limit.

5.2. Simulations

Below, some selected simulations are presented. A mixed statistical model was applied that was introduced in detail in [4]: The permanent gaps are induced by different Bernoulli distributions and temporary gaps are generated through distinct extents of shuffling with the help of distinct λ parameters of the exponential distribution.

During the simulations 1 million keys are always emitted, however, in case of permanent gaps/loss, a prefixed ratio/amount of the keys did not arrive to the trees.

During the simulations we investigated the evolution of the number of nodes, number of rotations, number of merges, etc., over the received keys and/or as a function of λ. In this contribution, we limit our attention to the number of nodes as a function of received keys and λ.

We show the number of nodes over the received keys with different λ parameters, Figure 12. The ratio of the permanent loss is 10 percent.

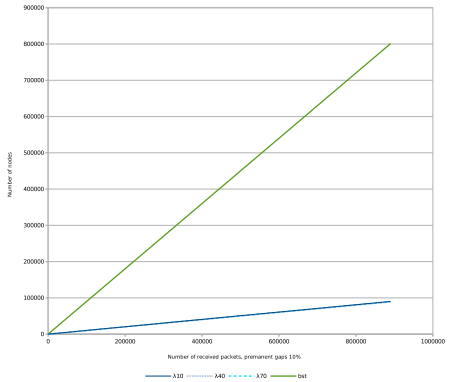

Figure 12. Time series of number of nodes for balanced BST and IMBT for permanent gaps.

In the case of BST, the required number of nodes is equal with the received number of keys.

Regarding IMBT, along with uniformly distributed gaps, the required number of nodes is proportional with N/a to store the keys which is practically independent from λ. It is a permanent space complexity gain compared to other data structures.

Taking into account the uniformly distributed character of the gaps and considering the Equations (6b) and (7), the IMBT provides time complexity advantage on average up to $N = a^3$ keys. Over that, only the space complexity gain remains. Otherwise, from a time complexity point of view IMBT behaves just like other BSTs.

Next, the number of nodes of the balanced BST and IMBT over the received keys in the case of temporary gaps only is evaluated, Figure 13.

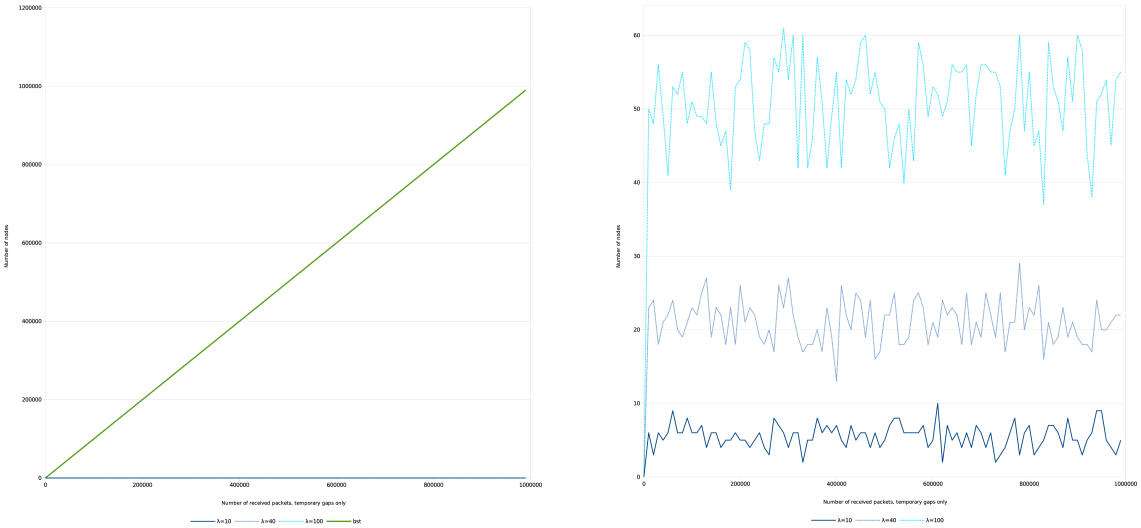

Figure 13. Number of nodes for balanced BST and IMBT for temporary gaps, as a function of the received keys/degree of shuffling.

In the case of BST, the result is similar to the previous one, which is visible on the left side of the figure.

However, in case of IMBT the situation is different. As we theoretically predicted the number of nodes becomes a stationary process in case of IMBT. This is visible on the right side of the figure, which are the magnified results of the bottom of left measurements.

That is, the simulation confirms Equation (32), according to which $A(N, a)$ can be considered (stochastically) constant, that is the time complexity of the search operation is $O(1)$. Moreover, the space complexity can also be considered constant, as stated above. Therefore, the IMBT can be a very efficient data structure both in terms of space/time complexity of the search operation.

During the previous simulations only every 10,000th states were displayed. In the following, for demonstrating purposes, we provide the diagram, where all 1 million states are displayed with $\lambda = 70$. From Figure 14 the stability of the process is clearly visible.

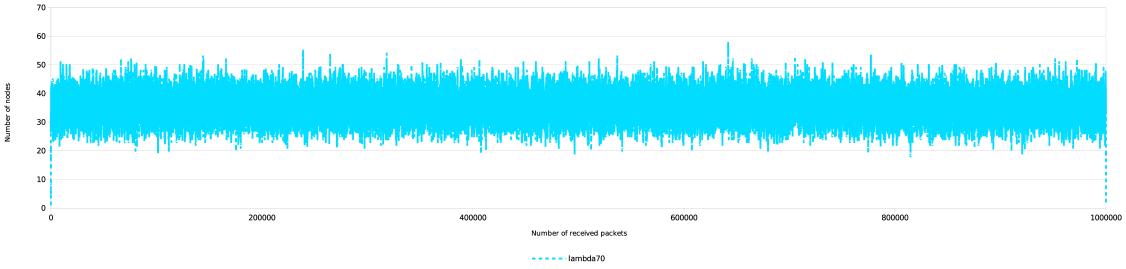

Figure 14. Balanced IMBT, temporary gaps only. Instantaneous number of nodes over received keys.

6. Conclusions

In the contribution above we have mixed together the Markov property based node number behavior and the corresponding contingency tables. In numerous cases, where it was possible,

we gave a two-parameters-based formula to be able to estimate the average cost of search operations. To conclude, in this contribution, two distinct ways have been presented that enable the analysis, in the possession of a-priori knowledge on the traffic pattern, of the performance of the IMBT data structure, and a subsequent decision whether to apply it over another well known data structure, for the particular case.

Author Contributions: Conceptualization, I.F.; Methodology, I.F.; Project administration, L.F.; Resources, S.S. and L.F.; Supervision, S.S. and L.F.; Writing—original draft, I.F.; Writing—review editing, S.S. and L.F.

Funding: This research was supported by the National Research, Development, and Innovation Fund of Hungary within the Quantum Technology National Excellence Program (Project Nr. 2017-1.2.1-NKP-2017-00001).

Conflicts of Interest: The authors declare no conflict of interest.

References

1. Cormen, T.H.; Leiserson, C.E.; Rivest, R.L.; Stein, C. *Introduction to Algorithms*, 3rd ed.; MIT Press and McGraw-Hill: London, UK, 2009; ISBN 0-262-03384-4.
2. Stoica, I.; Morris, R.; Liben-Nowell, D.; Karger, D.R.; Kaashoek, M.F.; Dabek, F.; Balakrishnan, H. Chord: A Scalable Peer-to-peer Lookup Protocol for Internet Applications. *IEEE/ACM Trans. Netw.* **2003**, *11*, 17–32. [CrossRef]
3. Finta, I.; Farkas, L.; Sergyán, S.; Szénási, S. Interval Merging Binary Tree. In Proceedings of the ICA3PP 2017, Helsinki, Finland, 21–23 August 2017. [CrossRef]
4. Finta, I.; Élias, G.; Illés, J. Packet Loss and Duplication Handling in Stream Processing Environment. In Proceedings of the CINTI 2018, Budapest, Hungary, 21–22 Novermber 2018. [CrossRef]
5. STORM—A Distributed Real-Time Computation System. Available online: http://storm.apache.org/documentation/Home.html (accessed on 28 November 2019)
6. Finta, I.; Szénási, S. State-space Analysis of the Interval Merging Binary Tree. *Acta Polytech. Hung.* **2019**, *16*, 71–85. [CrossRef]
7. Adelson-Velsky, G.; Landis, E. An algorithm for the organization of information. In *Doklady Akademii Nauk*; Russian Academy of Sciences: Moscow, Russia, 1962; pp. 263–266.
8. Bayer, R. Symmetric binary B-Trees: Data structure and maintenance algorithms. *Acta Inform.* **1972**, *1*, 290–306. [CrossRef]
9. Lauritzen, S.L. Lectures on Contingency Tables, 2002, Electronic edition, Aalborg University. Available online: http://www.stats.ox.ac.uk/~steffen/papers/cont.pdf (accessed on 28 November 2019)
10. Barvionk, A. Enumerating Contingency Tables via Random Permanents. *Comb. Probab. Comput.* **2008**, *17*, 1–19. [CrossRef]
11. Barvinok, A.; Luria, A.; Samorodnitsky, A.; Yong, A. An approximation algorithm for counting contingency tables. *Random Struct. Algorithms* **2010**, *37*, 25–66. [CrossRef]
12. Chung, K.L. *Markov Chains with Stationary Transition Probabilities*; Springer: Berlin/Heidelberg, Germany, 1960.
13. Meyn, S.P.; Tweedie, R.L. *Markov Chains and Stochastic Stability*; Springer: London, UK, 2012; ISBN 9781447132677.
14. Bernstein, S.N. Theory of Probabilities. Moskva/Leningrad, Russia, 1946.
15. Doukhan, P.; Louhichi, S. A new weak dependence condition and applications to moment inequalities. *Stoch. Process. Their Appl.* **1999**, *84*, 313–342. [CrossRef]
16. Lawder, J.; King, P. Querying Multi-dimensional Data Indexed Using Hilbert Space-Filling Curve. *ACM Sigmod Rec.* **2000**, *30*, 19–24. [CrossRef]
17. Bóna, M. *A Walk Through Combinatorics: An Introduction to Enumeration and Graph Theory*; World Scientific Publishing: Singapore, 2002; pp. 145–164, ISBN 981-02-4900-4.
18. Hardy, G.H.; Ramanujan, S. Asymptotic Equatione in Combinatory Analysis. *Proc. Lond. Math. Soc.* **1918**, *2*, 75–115. [CrossRef]
19. Sleator, D.D.; Tarjan, R.E. Self-Adjusting Binary Search Trees. *J. ACM* **1985**, *32*, 652–686. [CrossRef]