algorithms

Article

Text Indexing for Faster Gapped Pattern Matching

Md Helal Hossen [1], Daniel Gibney [1], and Sharma V. Thankachan [2,*]

[1] Department of Computer Science, University of Texas at Dallas, Richardson, TX 75080-3021, USA; mdhelal.hossen@utdallas.edu (M.H.); daniel.gibney@utdallas.edu (D.G.)
[2] Department of Computer Science, College of Engineering, North Carolina State University, Raleigh, NC 27695, USA
* Correspondence: svalliy@ncsu.edu; Tel.: +1-(919)-513-0816

Abstract: We revisit the following version of the Gapped String Indexing problem, where the goal is to preprocess a text $T[1\,..\,n]$ to enable efficient reporting of all occ occurrences of a gapped pattern $P = P_1[\alpha\,..\,\beta]P_2$ in T. An occurrence of P in T is defined as a pair (i, j) where substrings $T[i\,..\,i + |P_1|)$ and $T[j\,..\,j + |P_2|)$ match P_1 and P_2, respectively, with a gap $j - (i + |P_1|)$ lying within the interval $[\alpha\,..\,\beta]$. This problem has significant applications in computational biology and text mining. A hardness result on this problem suggests that any index with polylogarithmic query time must occupy near quadratic space. In a recent study [STACS 2024], Bille et al. presented a sub-quadratic space index using space $\widetilde{\mathcal{O}}(n^{2-\delta/3})$, where $0 \le \delta \le 1$ is a parameter fixed at the time of index construction. Its query time is $\widetilde{\mathcal{O}}(|P_1| + |P_2| + n^{\delta} \cdot (1 + \text{occ}))$, which is sub-linear per occurrence when $\delta < 1$. We show how to achieve a gap-sensitive query time of $\widetilde{\mathcal{O}}(|P_1| + |P_2| + n^{\delta} \cdot (1 + \text{occ}^{1-\delta}) + \sum_{g\in[\alpha..\beta]} \text{occ}_g \cdot g^{\delta})$ using the same space, where occ_g denotes the number of occurrences with gap g. This is faster when there are many occurrences with small gaps.

Keywords: text indexing; string algorithms; gapped pattern matching

check for updates

Citation: Hossen, M.H.; Gibney, D.; Thankachan, S.V. Text Indexing for Faster Gapped Pattern Matching. *Algorithms* **2024**, *17*, 537. https://doi.org/10.3390/a17120537

Academic Editor: Adele Anna Rescigno

Received: 9 October 2024
Revised: 13 November 2024
Accepted: 21 November 2024
Published: 23 November 2024

1. Introduction

Let $T[1\,..\,n]$ be a string (called the text) over a polynomially sized alphabet and $P = P_1[\alpha\,..\,\beta]P_2$ be a *gapped pattern* , where P_1 and P_2 are strings and $[\alpha\,..\,\beta]$ is an integer interval called the *gap range*. An occurrence of P in T is represented as a pair (i, j) such that $T[i\,..\,i + |P_1|) = P_1$, $T[j\,..\,j + |P_2|) = P_2$ with gap $j - (i + |P_1|) \in [\alpha\,..\,\beta]$. Locating occurrences of gapped patterns has numerous applications in computational biology [1–7] and text mining [8–11]. The algorithmic variant of the gapped pattern matching problem is well studied [12,13] and can be solved in $\widetilde{\mathcal{O}}(n + m + \text{occ})$ time ($\widetilde{\mathcal{O}}(\cdot)$ suppresses polylogarithmic factors, in particular, $(\log n)^k = \widetilde{\mathcal{O}}(1)$ for any constant k.) [3,6,14–16].

This work focuses on an indexing version of the problem where the text T is known during preprocessing. The gapped pattern $P = P_1[\alpha\,..\,\beta]P_2$ is provided as a query. Formally, we consider Problem 1.

Problem 1 (Gapped String Indexing)**.**
Preprocess: A text $T[1\,..\,n]$.
Query: Given a gapped pattern $P = P_1[\alpha\,..\,\beta]P_2$*, report all occurrences of* P *in* T.

In recent work, Bille et al. [17] showed that for all $0 \le \delta \le 1$, an index for Problem 1 can be constructed occupying $\widetilde{\mathcal{O}}(n^{2-\delta/3})$ or $\widetilde{\mathcal{O}}(n^{3-2\delta})$ space, and answering queries $\widetilde{\mathcal{O}}(|P_1| + |P_2| + n^{\delta} \cdot (\text{occ} + 1))$ time. Our main result is an index that requires similar space but achieves query times parameterized by the gaps present in the occurrences. This is stated formally in Theorem 1 below. In particular, our result improves the case where gaps for most occurrences are small.

Theorem 1. *For all $0 \leq \delta \leq 1$, there exists an index for Gapped String Indexing that occupies $\widetilde{\mathcal{O}}(n^{2-\delta/3})$ space and answers queries in time*

$$\widetilde{\mathcal{O}}(|P_1| + |P_2| + n^{\delta} \cdot (1 + \mathrm{occ}^{1-\delta}) + \sum_{g \in [\alpha\,..\,\beta]} \mathrm{occ}_g \cdot g^{\delta})$$

where occ_g is the number of occurrences with gap g.

Our main technique revolves around solving the following bounded variant of Gapped String Indexing, which may be of independent interest.

Problem 2 (Bounded Gapped String Indexing)**.**
Preprocess: A text $T[1\,..\,n]$ and integer G.
Query: Given a gapped pattern $P = P_1[\alpha\,..\,\beta]P_2$ where $\beta < G$, report all occurrences of P in T.

For Problem 2, we provide a solution with space and query complexity stated in Theorem 2.

Theorem 2. *For every $0 \leq \delta \leq 1$, there exists an index for Bounded Gapped String Indexing that occupies $\widetilde{\mathcal{O}}(n^{2-\delta/3})$ space and answers queries in time*

$$\widetilde{\mathcal{O}}(|P_1| + |P_2| + n^{\delta} \cdot (1 + \mathrm{occ}^{1-\delta}) + G^{\delta} \cdot \mathrm{occ})$$

where occ is the size of the output.

To prove Theorem 2, we build on previous results for the Gapped Set Intersection Problem (defined formally in Section 2), developing techniques for the bounded-gap case. We utilize a blocking technique based on binary trees, which when combined with a generalized form of the Kraft–McMillan inequality, allows us to achieve an improved query time (Lemma 9). Our preprocessing techniques differ from those of Bille et al. [17] in that we rely on the bounded nature of the gaps to perform the proposed blocking technique. Indeed, Theorem 2 is accomplished through extra preprocessing (blocking and building data structures for blocks) that is possible only when an upper bound $G > \beta$ is known in advance. As described in Section 3.6, Theorem 1 is then achieved by applying Theorem 2 for different ranges of G.

1.1. Previous Work

A long line of research has contributed to the current results on Gapped String Indexing. Many of these place some restrictions on the problem. The earliest results are by Peterlongo et al. [18], and are for the heavily restricted version where lengths $|P_1|$, $|P_2|$, and gap $g = \alpha = \beta$ are known for preprocessing. Given these restrictions, their solution uses $\mathcal{O}(n)$ space and achieves an optimal query time of $\mathcal{O}(m + \mathrm{occ})$.

In a slightly generalized variant where only the gap length $g = \alpha = \beta$ is given at preprocessing, Iliopoulos and Rahman [19] present an index using space $\mathcal{O}(n \log^{1+\epsilon} n)$, where $\epsilon > 0$ is an arbitrarily small constant, with query time $\mathcal{O}(m + \log\log n + \mathrm{occ})$. For the same case where g is known in advance, Bille and Gørtz [17] introduced an improved approach, achieving optimal query time with $\mathcal{O}(n \log^{\epsilon} n)$ space. In the case where only an upper bound G is given on β, the problem can be reduced to 3D range searching with an index using $\widetilde{\mathcal{O}}(Gn)$ space and $\widetilde{\mathcal{O}}(|P_1| + |P_2| + \mathrm{occ})$ query time [20]. Conditioned on the Strong Set-Disjointness Conjecture [21], Bille et al. [22] also demonstrated that any solution with $\widetilde{\mathcal{O}}(|P_1| + |P_2| + \mathrm{occ})$ query time must use $\widetilde{\mathcal{O}}(Gn)$ space. Recently, Ganguly et al. [23] proposed a variant (called Bounded Ratio Gapped String Indexing) where the gap β satisfies $\beta \leq \gamma \cdot (|P_1| + |P_2|)$. Here, γ represents a gap ratio fixed at index construction time. Under this relaxed constraint, an index can be constructed occupying $\widetilde{\mathcal{O}}(\gamma \cdot n)$ space and having $\widetilde{\mathcal{O}}(|P_1| + |P_2| + \mathrm{occ})$ query time.

The framework employed by Bille et al. [17] utilizes the results for 3-SUM Indexing by Golovnev [24] (see also [25]). In 3-SUM indexing, one needs to preprocess two sets of integers, S_1 and S_2, so that given a query integer c, one can efficiently determine if there exists $a \in S_1$ and $b \in S_2$ such that $a + b = c$. The reduction from Gapped String Indexing to 3-SUM Indexing, through a series of intermediate problems, forms the basis of both [17] and this work. Leveraging one of these intermediate problems (between Gapped Indexing and 3-SUM Indexing) alleviates some steps for this work relative to Bille et al.'s results [17].

1.2. Notation and Technical Preliminaries

We use $[i\,..\,j]$ to refer to the set $\{i, i+1, \ldots, j-1, j\}$ and $[i\,..\,j)$ to the set $\{i, i+1, \ldots, j-1\}$. For an array of integers A, we use $A[i\,..\,j]$ to denote the subarray $A[i] \cdots A[j]$ and $A[i\,..\,j)$ to denote $A[i] \cdots A[j-1]$.

For a string T, we use $T[i]$ to refer to the i^{th} symbol in T, $T[i\,..\,j]$ to denote the substring $T[i] \cdots T[j]$, and $T[i\,..\,j)$ the substring $T[i] \cdots T[j-1]$. We call a substring of the form $T[i\,..\,n]$ a suffix of T and a substring of the form $T[1\,..\,i]$ a prefix of T. We use T^R to denote the reverse of the string T. The suffix tree [26] of a string $T[1\,..\,n]$ is a compact tree of all suffixes with leaves in corresponding lexicographic order. The suffix array, denoted $\mathrm{SA}[1\,..\,n]$, is defined such that $T[\mathrm{SA}[i]\,..\,n]$ is the i^{th} suffix when all suffixes are sorted in lexicographic order. For a pattern P, its suffix range $[a\,..\,b]$ is the maximal range such that for all $h \in [a\,..\,b]$, $T[\mathrm{SA}[h]\,..\,n]$ has prefix P. The suffix range exists if P occurs in T; otherwise, the suffix range is empty.

For convenience, we assume that all strings are over a polynomially-sized integer alphabet so that the suffix tree and suffix array can be constructed in linear time [27]. Given the suffix tree and suffix array, the suffix range of a string $P[1\,..\,m]$ can be found in $\mathcal{O}(m)$ time.

2. A Preliminary Solution

Before introducing our main solution, we first present a preliminary approach. While this solution does not achieve the query efficiency required to prove Theorem 1, it serves as a valuable foundation for our solution. As an initial step, we introduce the following two problem formulations from [17].

Problem 3 (Gapped Set Intersection (with Reporting))**.**
Preprocess: A collection of subsets $S_1, \ldots, S_k$ of total size $N = \sum_{i=1}^{k} |S_i|$ over integer universe $[0\,..\,U]$.
Query: Given (i, j, α, β), report if there exists (report all, resp.) $(a, b) \in S_i \times S_j$ where there exists $s \in [\alpha\,..\,\beta]$ such that $a + s = b$.

We also define the bounded version of Problem 3, analogous to Problem 2. Specifically, the bounded problem formulation provides a collection of subsets $S_1, \ldots, S_k$, and an integer G for preprocessing. A query consists of the tuple (i, j, α, β) with the additional guarantee that $\beta < G$. We will utilize the following results from Bille et al. [17].

Lemma 1 (Index for Gapped Set Intersection (Theorems 5 and 6 from [17]))**.** *For every $0 \le \delta \le 1$, there is a data structure for the Gapped Set Intersection that occupies $\widetilde{\mathcal{O}}(N^{2-\delta/3})$ space and answers existential queries in time $\widetilde{\mathcal{O}}(N^{\delta})$ and reporting queries in time $\widetilde{\mathcal{O}}(N^{\delta} \cdot (\mathrm{occ} + 1))$, where* occ *is the size of the output.*

Lemma 2 allows us to focus on Problem 3 for the majority of the remaining work.

Lemma 2 (Reduction to Bounded Gapped Set Intersection (Adapted from [17]))**.** *Assume there is a data structure for the Bounded Gapped Set Intersection (with Reporting) using $s(N)$ space that answers existential queries in time $t(N)$ and reporting queries in time $t(N) \cdot (1 + \mathrm{occ})$. Then, there exists a data structure for Bounded Gapped String Indexing using $\widetilde{\mathcal{O}}(n + s(n))$ space. It can answer existential queries in time $\widetilde{\mathcal{O}}(|P_1| + |P_2| + t(n))$ and reporting queries in time*

$\widetilde{\mathcal{O}}(|P_1| + |P_2| + t(n) \cdot (1 + \mathrm{occ}))$, *where* n *is the length of the input text and* occ *is the size of the output.*

For completeness, we sketch an adaptation of the proof from [17].

Proof. We begin with the array $S_2[1\,..\,n]$ such that $S_2[i] = \mathrm{SA}[i]$, where $\mathrm{SA}[\cdot]$ is the suffix array of T. We also define the array $S_1[1\,..\,n]$, where $S_1[i] = n - \mathrm{SA}[i]^R + 1$, and $\mathrm{SA}[\cdot]^R$ is the suffix array of the reverse string T^R. Both arrays are decomposed into subsets corresponding to dyadic intervals of the form $[1 + \kappa \cdot 2^j \,..\, (\kappa + 1) \cdot 2^j]$, where $0 \le \kappa \le \lfloor n/2^j \rfloor - 1$ and $0 \le j \le \lfloor \log n \rfloor$. These subsets serve as the input to the Gapped Set Intersection instance. Notably, the sum of the subset cardinalities is $\mathcal{O}(n \log n)$.

Given a query $P_1[\alpha\,..\,\beta]P_2$, we decompose the suffix range for P_1^R into a collection of $\mathcal{O}(\log n)$ dyadic-sized subarrays of S_1, corresponding to precomputed subsets. We denote this collection by $\mathcal{A}$. Similarly, we decompose the suffix range of P_2 into a collection of $\mathcal{O}(\log n)$ dyadic intervals corresponding to precomputed subsets of S_2 denoted by $\mathcal{B}$. We then perform $\mathcal{O}(\log^2 n)$ queries (For notational brevity here, we abuse notation slightly. The actual query would be on the indices corresponding to subsets A and B.) (A, B, α, β) for all $A \in \mathcal{A}$ and $B \in \mathcal{B}$. It is important to note that the bounds α and β remain unchanged throughout the reduction. □

We next present a preliminary solution for the Bounded Gapped Set Intersection with space $\widetilde{\mathcal{O}}(nG^{1-\delta/3})$ and reporting time $\widetilde{\mathcal{O}}(|P_1| + |P_2| + n/G^{1-\delta} + G^{\delta}\mathrm{occ})$.

2.1. The Data Structure

Let $A[1\,..\,N]$ be an array containing the elements of $\cup_{i=1}^{k} S_i$ in sorted order where we now define $N = \left|\cup_{i=1}^{k} S_i\right|$. We subdivide A into overlapping blocks of size $2G$, with each consecutive block overlapping by G elements. Formally, for $i \in [1\,..\,\lfloor N/G \rfloor)$, we define block $B_i = A[1 + (i-1)G \,..\, 1 + (i+1)G)$. See Figure 1.

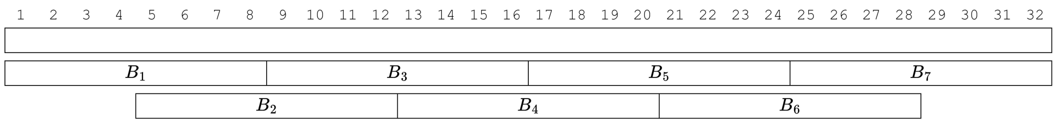

Figure 1. A preliminary blocking scheme for $N = 32$ and $G = 4$.

Given a query (i, j, α, β) where $\beta < G$, consider the following: if $(a, b) \in S_i \times S_j$ and there exists an $s \in [\alpha\,..\,\beta]$ such that $a + s = b$, then, since $s \le \beta < G$, a and b must lie in the same block. Consequently, we construct the data structure from Lemma 1 for each block, B_h, using the subsets $S'_1 = S_1 \cap B_h, \ldots, S'_k = S_k \cap B_h$, excluding empty subsets. Thus, the number of subsets in a given block may be fewer than k. We store in sorted order, for each block B_h, the original subset indices i for each non-empty subset $S'_t = S_i \cap B_h$. We also store the associated t value in the same sorted order.

2.2. Querying

Given query (i, j, α, β), we iterate through $h \in [1\,..\,\lfloor N/G \rfloor]$. For block B_h, we first determine if $S_i \cap B_h$ is empty. This can be accomplished using binary search over the stored indices for B_h, which were described above. If $S_i \cap B_h = \emptyset$, we are finished for B_h as there are no solutions in this block. Otherwise, we obtain t_i such that $S'_{t_i} = B_h \cap S_i$. Similarly, we perform a binary search for j over the indices for B_h. If $S_j \cap B_h = \emptyset$, we are finished for B_h. Otherwise, we obtain t_j such that $S'_{t_j} = B_h \cap S_j$. We then make the query $(t_i, t_j, \alpha, \beta)$ to the data structure for block B_h and store the reported solutions.

After all blocks are processed, a final sort of the stored solutions is performed and the duplicates are removed. We then output the resulting list of occurrences.

2.3. Analysis

The above approach requires $\widetilde{\mathcal{O}}(G^{2-\delta/3})$ space per block, resulting in an overall space requirement of $\widetilde{\mathcal{O}}((N/G) \cdot G^{2-\delta/3}) = \widetilde{\mathcal{O}}(NG^{1-\delta/3})$. Reporting all occurrences within a single block B_h takes $\widetilde{\mathcal{O}}(G^{\delta}(\mathrm{occ}_h + 1))$, where occ_h represents the number of occurrences in B_h. The total time across all blocks is $\widetilde{\mathcal{O}}(G^{\delta} \cdot N/G + G^{\delta}\mathrm{occ}) = \widetilde{\mathcal{O}}(N/G^{1-\delta} + G^{\delta}\mathrm{occ})$. Note that each solution occurs in at most two blocks, so the final sorting and duplicate removal step does not change the asymptotic query time.

Applying the reduction in Lemma 2, this yields a solution for Bounded Gapped String Indexing with Reporting that requires $\widetilde{\mathcal{O}}(nG^{1-\delta/3})$ space and achieves a query time of $\widetilde{\mathcal{O}}(|P_1| + |P_2| + n/G^{1-\delta} + G^{\delta}\mathrm{occ})$. However, this does not provide the desired query time complexity, particularly for large values of G. We now present an improved solution.

3. An Improved Solution

The basis of the improved solution is to carefully decompose the array A (defined in Section 2) to avoid having to check every block for occurrences as was done in the preliminary solution. We rely heavily on techniques from [28] and its extensions in [22,29].

3.1. The Data Structure

We will construct a tree data structure over the array A. Each node in the tree will have an associated subarray of A. We construct the tree structure over the array A recursively as follows: The tree's root is associated with the entire array $A[1\,..\,N]$. We designate the root's midpoint as $m = \lfloor (1+N)/2 \rfloor$. The root node is given a middle child, that is a leaf representing the subarray $A[m-G+1\,..\,m+G]$. We then recursively create two child subtrees: the left child subtree corresponds to $A[1\,..\,m]$, and the right child subtree corresponds to $A[m+1\,..\,N]$. If at any point the size of a subarray is at most $2G$, we treat the node as a leaf node. See Figure 2. For each node in the tree, we create the Gapped Set Intersection Data Structure from Lemma 1. For each leaf node, we create the Gapped Set Intersection Data Structure outlined in Lemma 1. See Algorithm 1 for pseudocode. Like in Section 2, these data structures are constructed over the non-empty subsets of each block, and we maintain the mapping from the query i and j to the corresponding non-empty subset if it exists. These details are omitted from the pseudocode.

Algorithm 1 Construction Algorithm

```
1:  procedure CONSTRUCT($A, G, l, r$)
2:      if $r - l + 1 > 2G$ then
3:          $v \leftarrow$ CREATE_INTERNAL_NODE($A, l, r$)
4:          $m \leftarrow \lfloor (l+r)/2 \rfloor$
5:          $v$.middle_child $\leftarrow$ CREATE_LEAF_NODE($A, m-G+1, m+G$)
6:          $v$.left_child $\leftarrow$ CONSTRUCT($A, G, l, m$)
7:          $v$.right_child $\leftarrow$ CONSTRUCT($A, G, m+1, r$)
8:      else
9:          $v \leftarrow$ CREATE_LEAF_NODE($A, l, r$)
10:     end if
11:     return $v$
12: end procedure

13: procedure CONSTRUCT($A$, $G$)
14:     $root \leftarrow$ CONSTRUCT($A, G, 1, N$)
15: end procedure
```

In terms of notation, we call the leaves created in Line 5 of Algorithm 1 *middle children leaves*. For a node v with associated subarray $A[l\,..\,r]$, we call $m = \lfloor (l+r)/2 \rfloor$ the *midpoint* of v.

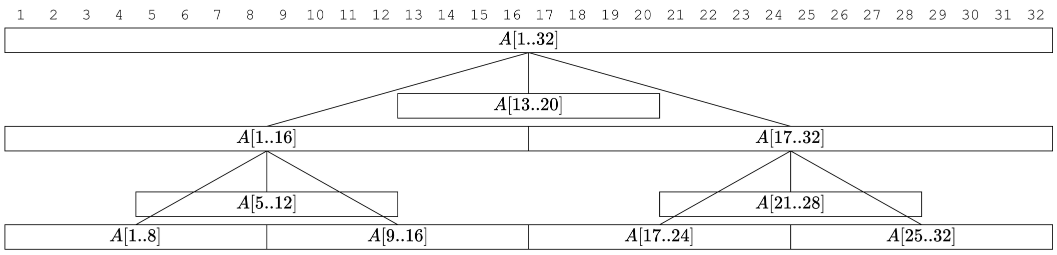

Figure 2. Tree structure constructed by Algorithm 1 with $N = 32$ and $G = 4$.

3.2. Querying

To query the tree structure, we begin at the root. We first use the data structure from Lemma 1 to check whether any occurrence is contained in the current nodes' associated subarray. If the current node is a leaf and it contains an occurrence, we report all occurrences using the data structure from Lemma 1. If the current node is not a leaf and contains an occurrence, we recursively search all of its children. This is shown in pseudocode in Algorithm 2.

Algorithm 2 Query Algorithm

```
1:  procedure SEARCH($v, i, j, \alpha, \beta$)
2:      if not $v$.contains_occurrence($i, j, \alpha, \beta$) then
3:          return
4:      else if $v$.contains_occurrence($i, j, \alpha, \beta$) and $v$ is leaf then
5:          $v$.report_all_occurrences($i, j, \alpha, \beta$)
6:      else if $v$.contains_occurrence($i, j, \alpha, \beta$) and $v$ is internal node then
7:          SEARCH($v$.middle_child, $i, j, \alpha, \beta$)
8:          SEARCH($v$.left_child, $i, j, \alpha, \beta$)
9:          SEARCH($v$.right_child, $i, j, \alpha, \beta$)
10:     end if
11: end procedure

12: procedure QUERY($i, j, \alpha, \beta$)
13:     SEARCH($root, i, j, \alpha, \beta$)
14: end procedure
```

3.3. Correctness

The key observation from Figure 2 is that the union of the leaf nodes in the tree constructed by Algorithm 1 resembles the blocking scheme described in Section 2 (see Figure 1 for comparison). Based on this observation, we now formalize the following key lemmas.

Lemma 3. *Every subarray A' of size at most G is contained in the subarray of some leaf.*

Proof. For the sake of contradiction, assume that subarray A' is not contained in the subarray of any leaf. Let v be the node of maximum height that contains A'. Let l and r denote the bounds for the subarray for node v, and let $m = \lfloor (l + r)/2 \rfloor$.

Since A' is not contained in any leaf, it must be that A' is not fully contained within the range $[m - G + 1 \,..\, m + G]$. Therefore, A' either starts in the range $[l \,..\, m - G]$ or ends in the range $[m + G + 1 \,..\, r]$. In the former case, since $m - G + |A'| - 1 \leq m - G + G = m$, we have that A' must be contained within the subarray of v.left_child, which contradicts the assumption that v is the highest node containing A'. In the latter case, since $m + G + 1 - |A'| + 1 \geq m + G + 1 - G + 1 > m + 1$, we have that A' must be contained within the subarray of v.right_child, which again contradicts the assumption that v is the highest node containing A'. □

The correctness of the above query procedure then follows from the fact that every block of size at most G is contained within the subarray of some leaf node v. All ancestors u of leaf v will report that they contain an occurrence, allowing the DFS traversal to continue until leaf v is reached and its occurrences are reported.

3.4. Space Analysis

First, considering only the Gapped Set Intersection Data Structure from Lemma 1 on non-leaf nodes, this requires space logarithmic factors from

$$\sum_{i=0}^{\log N} 2^i \left(\frac{N}{2^i}\right)^{2-\delta/3} = N^{2-\delta/3} \sum_{i=0}^{\log N} \left(\frac{1}{2^{1-\delta/3}}\right)^i.$$

Since $0 \leq \delta \leq 1$, we have $1/2^{1-\delta/3} < 1$, and the geometric series converges to a constant. Hence, ignoring leaf nodes, the space is $\widetilde{\mathcal{O}}(N^{2-\delta/3})$.

Next, we include the leaves. We first show that the number of leaves is $\mathcal{O}(N/G)$.

Lemma 4. *Every non-middle child leaf's associated subarray has size at least G.*

Proof. Suppose, for the sake of contradiction, there exists a leaf u that has a subarray size less than G. Let v be the parent of u with range l to r and midpoint m. If u is a left child, it has a subarray size

$$m - l + 1 = \left\lfloor \frac{l+r}{2} \right\rfloor - l + 1 < G.$$

The above implies

$$\frac{l+r}{2} - 1 - l + 1 < G,$$

which leads to $r - l < 2G$. However, this implies $r - l + 1 \leq 2G$, so v had a subarray size of at most $2G$. In such a case, our algorithm would not recursively create a left child for v, a contradiction.

Similarly, if u is a right child with subarray size less than G, it has size

$$r - (m+1) + 1 = r - \left(\left\lfloor \frac{l+r}{2} \right\rfloor + 1 \right) + 1 < G,$$

meaning $r - \lfloor (l+r)/2 \rfloor < G$. This implies $r - (l+r)/2 < G$. Hence, $r - l < 2G$. Again, we conclude v has a subarray size small enough that our algorithm would not recursively create a right child for v, a contradiction. $\square$

Lemma 5. *The number of leaves in the tree structure created by Algorithm 1 is $\mathcal{O}(N/G)$.*

Proof. Since all leaves created as non-middle children have disjoint associated subarrays, their union represents a set of cardinality N, and (by Lemma 4) each represents a disjoint subset of size at least G. Therefore, there are at most $\mathcal{O}(N/G)$ non-middle child leaves. Next, because the tree (still excluding middle children) is a binary tree, the total number of internal nodes is also $\mathcal{O}(N/G)$. Furthermore, including the middle child leaves at most doubles the total number of nodes in the tree. $\square$

Because each leaf contains a subarray of size $\mathcal{O}(G)$, the space for the data structures for each leaf is $\widetilde{\mathcal{O}}(G^{2-\delta/3})$. Combined with Lemma 5, the total space for leaves is $\widetilde{\mathcal{O}}(G^{2-\delta/3} \cdot N/G) = \widetilde{\mathcal{O}}(G^{1-\delta/3} \cdot N)$, which, since $G \leq N$, is also $\widetilde{\mathcal{O}}(N^{2-\delta/3})$.

3.5. Query Time Analysis

We now analyze the run time of Algorithm 2. Our first step can be seen as a modification of the Kraft–McMillan inequality.

Lemma 6. *For a rooted binary tree $\mathcal{T} = (V, E)$, let $h(v)$ denote the height of node v in $\mathcal{T}$. Then*

$$\sum_{v \in V} 2^{-h(v)} \le 1 + \log |V|.$$

Proof. We use induction on the tree height. The base case holds with a single node having height 0 since $2^0 = 1 \le 1 + \log 1 = 1$. For an arbitrary tree $\mathcal{T} = (V, E)$ with $|V| > 1$ nodes, let the left subtree of the root be $\mathcal{T}_{\mathcal{L}} = (V_L, E_L)$ with relative height function h_L, and the right child of the root, $\mathcal{T}_R = (V_R, E_R)$, with relative height function h_R. Then,

$$\begin{aligned}
\sum_{v \in V} 2^{-h(v)} &= 2^0 + \sum_{v \in V_L} 2^{-(h_L(v)+1)} + \sum_{v \in V_R} 2^{-(h_R(v)+1)} \\
&= 1 + \frac{1}{2} \sum_{v \in V_L} 2^{-h_L(v)} + \frac{1}{2} \sum_{v \in V_R} 2^{-h_R(v)} \\
&\le 1 + 1 + \frac{1}{2} \log |V_L| + \frac{1}{2} \log |V_R| && \text{(By Inductive Hypothesis)} \\
&= 1 + 1 + \log \sqrt{|V_L||V_R|} \\
&\le 1 + 1 + \log \frac{|V_L| + |V_R|}{2} && \text{(Inequality of Arithmetic and Geometric Means)} \\
&= 1 + \log(|V_L| + |V_R|) \\
&\le 1 + \log(1 + |V_L| + |V_R|) = 1 + \log |V|. \quad \square
\end{aligned}$$

Lemma 7. *Let s be the number of leaves for which we have to run the "report_all_occurrences" subroutine in Algorithm 2. Let $\mathcal{V}$ be the set of nodes on which* SEARCH *is executed in Algorithm 2. Then, $|\mathcal{V}| = \mathcal{O}(1 + s \log N)$.*

Proof. The height of the tree constructed by Algorithm 1 is $\mathcal{O}(\log N)$. Each root-to-v path for all v where "report_all_occurrences" is called contributes at most $\mathcal{O}(\log N)$ calls of SEARCH. Thus, the contribution overall of these paths is $\mathcal{O}(1 + s \log N)$. What remains to be counted are nodes where SEARCH is called and "contains_occurrence" reports there are no occurrences. For these, observe that each node on the root-to-v path for all v where "report_all_occurrences" is called has at most two children where this can be the case. Thus, including these nodes at most triples the total number of nodes on which SEARCH is called. □

We take s and $\mathcal{V}$ as defined in Lemma 7. First, we consider the time used by calls to "contains_occurrence". Because the size of a subarray for a node v at height $h(v)$ is $\mathcal{O}(N/2^{h(v)})$, by Lemma 1, a call to "contains_occurrence" for a node v at height $h(v)$ requires time $\tilde{\mathcal{O}}((N/2^{h(u)})^{\delta})$. The combined time used for "contains_occurrence" calls is polylogarithmic factors from

$$\sum_{v \in \mathcal{V}} \left(\frac{N}{2^{h(u)}} \right)^{\delta} = N^{\delta} \sum_{v \in \mathcal{V}} \left(\frac{1}{2^{h(v)}} \right)^{\delta}. \tag{1}$$

We next apply Hölder's inequality to obtain the bound

$$\sum_{v \in \mathcal{V}} \left(\frac{1}{2^{h(v)}} \right)^{\delta} = \sum_{v \in \mathcal{V}} \left(\frac{1}{2^{h(v)}} \right)^{\delta} 1^{1-\delta} \le \left(\sum_{v \in \mathcal{V}} \frac{1}{2^{h(v)}} \right)^{\delta} \left(\sum_{v \in \mathcal{V}} 1 \right)^{1-\delta}.$$

Applying Lemma 6 and 7, we can further bound this as

$$\left(\sum_{v \in \mathcal{V}} \frac{1}{2^{h(v)}} \right)^{\delta} \left(\sum_{v \in \mathcal{V}} 1 \right)^{1-\delta} \le (1 + \log |\mathcal{V}|)^{\delta} (1 + s \log N)^{1-\delta}.$$

Substituting into Equation (1) , we obtain that the time used for "contains_occurrence" calls is $\widetilde{\mathcal{O}}(N^{\delta}(1+s)^{1-\delta})$.

Next, we consider the time used by calls to "report_all_occurrences". By Lemma 1, each leaf v on which "report_all_occurrences" is called takes time $\widetilde{\mathcal{O}}(G^{\delta}(1+\mathrm{occ}_v))$, where occ_v denotes the number of occurrences contained in the subarray for node v. To bound the time complexity, we should bound the number of times an occurrence can be reported over all blocks. To this end, we prove Lemma 8.

Lemma 8. *Every subarray $A' = A[a\,..\,b]$ of size $b-a+1 \leq G$ has a non-empty intersection with $\mathcal{O}(1)$ leaf's subarrays.*

Proof. Consider first the tree structure without any middle child leaves. In this case, the leaves are all disjoint and, by Lemma 4, have subarray size at least G. Hence, at most two non-middle children leaves have non-empty intersections with A'.

Now, we incorporate the middle children. We consider the middle children leaves as being ordered according to their midpoint.
Claim: The difference between consecutive midpoints of middle children leaves is at least G. To see this, consider a middle child of u with midpoint $m_u = \lfloor (l_u + r_u)/2 \rfloor$ and a middle child of v with midpoint $m_v = \lfloor (l_v + r_v)/2 \rfloor$ immediately preceding m_u in the order. If v is in the left subtree of u, since

$$m_v + G < \frac{l_v + r_v}{2} + \frac{r_v - l_v + 1}{2} = r_v + \frac{1}{2},$$

we have $m_v + G \leq r_v \leq m_u$, where the last inequality follows from Line 6 in Algorithm 1. Hence, $m_u - m_v \geq G$.

If, on the other hand, v is not in the left subtree of u, then since the middle child of v is ordered before the middle child of u, v cannot be in the right subtree of u. If v is the parent of u, then by a similar argument,

$$m_u - G + 1 > \frac{l_u + r_u}{2} - 1 - \frac{r_u - l_u + 1}{2} + 1 = l_u - \frac{1}{2}$$

and we have $m_u - G + 1 \geq l_u \geq m_v + 1$, where the last inequality follows from Line 7 in Algorithm 1. Hence, $m_u - m_v \geq G$. In any other remaining cases, u and v must share either some lowest common ancestor or an intermediate vertex on the path from v to u. Call this vertex w. Observe that the middle child of w sits in the ordering between the middle children of u and v, a contradiction that makes the last remaining cases impossible.

Applying the above claim, we first observe that the range of possible midpoints of middle children leaves that can have a non-empty intersection with A' is $[a-G\,..\,b+G-1]$. Therefore, an upper bound on the number of middle children leaves that can have a non-empty intersection with A' is given by

$$\left\lceil \frac{b+G-1-(a-G)+1}{G} \right\rceil = \left\lceil \frac{b-a+2G}{G} \right\rceil = \left\lceil \frac{b-a}{G} \right\rceil + 2 = 3.$$

Hence, at most three middle children leaves have intersections with A'. Combined with at most two non-middle children leaves intersecting A', we arrive at the desired result. $\square$

As a result of Lemma 8, each occurrence is reported at most $\mathcal{O}(1)$ times, and the removal of potential duplicates does not affect the total asymptotic time complexity. Over all "report_all_occurrences" calls, the time is $\widetilde{\mathcal{O}}(G^{\delta}(s+\mathrm{occ}))$.

Summing the time for calls to "contains_occurrence" and "report_all_occurrences" we obtain a total time of $\widetilde{\mathcal{O}}(N^{\delta}(1+s)^{1-\delta} + G^{\delta}(s+\mathrm{occ}))$. Because each leaf on which "report_all_occurrences" is called contains at least one occurrence, and by Lemma 8, each occurrence is contained in $\mathcal{O}(1)$ leaves, we have $s = \mathcal{O}(\mathrm{occ})$. Furthermore, we have $(1+\mathrm{occ})^{1-\delta} = \mathcal{O}(1+\mathrm{occ}^{1-\delta})$. This gives us the following lemma.

Lemma 9. *For every* $0 \leq \delta \leq 1$, *there is a data structure for the Bounded Gapped Set Intersection with Reporting that occupies* $\widetilde{\mathcal{O}}(N^{2-\delta/3})$ *space and answers queries in time*

$$\widetilde{\mathcal{O}}(N^{\delta}(1+\mathrm{occ}^{1-\delta})+G^{\delta}\cdot\mathrm{occ})$$

where occ *is the size of the output.*

Combining Lemma 9 with the reduction used in Lemma 2, we obtain the result in Theorem 2, which is an $\widetilde{\mathcal{O}}(n^{2-\delta/3})$ space index for Bounded Gapped String Indexing with $\widetilde{\mathcal{O}}(|P_1|+|P_2|+n^{\delta}(1+\mathrm{occ}^{1-\delta})+G^{\delta}\cdot\mathrm{occ})$ query time.

3.6. Obtaining Theorem 1

We now apply Theorem 2 to obtain Theorem 1. For convenience, we assume that n is a power of two (if not, we can pad T with extra symbols # not in T's alphabet until its length is a power of two. We can accomplish this while at most doubling its length). We construct the data structure from Lemma 9 for all G in 1, 2, 4, 8, ..., n. The space required across all data structures is polylogarithmic factors from

$$\sum_{i=0}^{\log n} n^{2-\delta/3} = n^{2-\delta/3}\sum_{i=0}^{\log n} 1 = \widetilde{\mathcal{O}}(n^{2-\delta/3}).$$

To answer a query $P_1[\alpha\,..\,\beta]P_2$, we split $[\alpha\,..\,\beta]$ into logarithmically many ranges $R_1 = [a_1\,..\,b_1] = [\alpha\,..\,2^{\lceil\log\alpha\rceil}]$, $R_2 = [a_2\,..\,b_2] = [2^{\lceil\log\alpha\rceil}+1\,..\,2^{\lceil\log\alpha\rceil+1}]$, ..., $R_k = [a_k\,..\,b_k] = [2^{\lfloor\log\beta\rfloor}\,..\,\beta]$, where in the case $\beta \leq 2^{\lceil\log\alpha\rceil}$, no split is performed. The query $P_1[\alpha\,..\,2^{\lceil\log\alpha\rceil}]P_2$ is given to the data structure for $G = 2^{\lceil\log\alpha\rceil}$. By Theorem 2, it reports occurrences in time $\widetilde{\mathcal{O}}(|P_1|+|P_2|+n^{\delta}(1+\mathrm{occ}_{R_1})^{1-\delta}+2^{\lceil\log\alpha\rceil}\mathrm{occ}_{R_1})$, where occ_{R_1} is the number of occurrences with a gap in the range $R_1 = [\alpha\,..\,2^{\lceil\log\alpha\rceil}]$. Continuing in this fashion for each split, the overall complexity is polylogarithmic factors from

$$\begin{aligned}
&\sum_{i=1}^{k}(|P_1|+|P_2|+n^{\delta}(1+\mathrm{occ}_{R_i})^{1-\delta}+b_i^{\delta}\mathrm{occ}_{R_i})\\
&\qquad\leq \widetilde{\mathcal{O}}(|P_1|+|P_2|)+n^{\delta}\sum_{i=1}^{k}(1+\mathrm{occ}_{R_i}^{1-\delta})+\sum_{i=1}^{k}b_i^{\delta}\mathrm{occ}_{R_i}\\
&\qquad= \widetilde{\mathcal{O}}(|P_1|+|P_2|+n^{\delta}(1+\mathrm{occ}^{1-\delta}))+\sum_{i=1}^{k}b_i^{\delta}\mathrm{occ}_{R_i}
\end{aligned}$$

where occ_{R_i} is the number of occurrences with a gap in range R_i. The last equality holds since $\sum_{i=1}^{k}\mathrm{occ}_{R_i}^{1-\delta} = \widetilde{\mathcal{O}}(\mathrm{occ}^{1-\delta})$.

We observe that for a given range R_i, $\mathrm{occ}_{R_i} = \sum_{g\in R_i}\mathrm{occ}_g$, where occ_g is the number of occurrences with gap exactly g. Furthermore, $b_i \leq 2g$ for all $g \in R_i$. Hence,

$$\left(\frac{b_i}{2}\right)^{\delta}\mathrm{occ}_{R_i} = \left(\frac{b_i}{2}\right)^{\delta}\sum_{g\in R_i}\mathrm{occ}_g \leq \sum_{g\in R_i} g^{\delta}\mathrm{occ}_g.$$

We conclude that

$$\sum_{i=1}^{k}\left(\frac{b_i}{2}\right)^{\delta}\mathrm{occ}_{R_i} \leq \sum_{i=1}^{k}\sum_{g\in R_i} g^{\delta}\mathrm{occ}_g = \sum_{g\in[\alpha..\beta]} g^{\delta}\mathrm{occ}_g,$$

giving us an overall query time complexity of

$$\widetilde{O}\left(|P_1| + |P_2| + n^{\delta}(1 + \mathrm{occ}^{1-\delta}) + \sum_{g \in [\alpha\,..\,\beta]} g^{\delta}\mathrm{occ}_g\right)$$

as desired. This completes the proof of Theorem 1.

4. Conclusions

We have presented an index for Gapped String Indexing with a reporting time parameterized by the gap lengths of the occurrences. Potential directions for further development include the following:

- Establishing matching conditional lower bounds based on the Strong Set-Disjointness Conjecture or other conjectures used in fine-grained complexity.
- Extensions to the multi-gap case: That is, preprocess a text to answer queries of the form $P_1[\alpha_1\,..\,\beta_1]\ldots[\alpha_{k-1}\,..\,\beta_{k-1}]P_k$. It is not immediate how to adapt Problem 3-based techniques to this setting.
- Extensions to the bounded ratio gapped setting of Ganguly et al. [23].

We acknowledge the likelihood that the gap-sensitive approach proposed here may have a worse query time compared to the prior approach by Bille et al. [17] for large gap values due to polylogarithmic factors. In such instances, it may be advantageous to consider some form of a meta-algorithm that employs our gap-sensitive approach for small gaps and the algorithm of Bille et al. [17] for larger gaps. We leave this as a possible direction for future research.

Author Contributions: Conceptualization, M.H.H., D.G. and S.V.T.; methodology, M.H.H., D.G. and S.V.T.; validation, M.H.H., D.G. and S.V.T.; formal analysis, M.H.H., D.G. and S.V.T.; investigation, M.H.H., D.G. and S.V.T.; writing—original draft preparation, M.H.H., D.G. and S.V.T.; writing—review and editing, M.H.H., D.G. and S.V.T.; supervision, D.G. and S.V.T.; project administration, D.G. and S.V.T. All authors have read and agreed to the published version of the manuscript.

Funding: This research was funded by U.S. National Science Foundation (NSF) award number CCF-2315822.

Data Availability Statement: No new data were created or analyzed in this study. Data sharing is not applicable to this article.

Acknowledgments: This research is supported in part by the U.S. National Science Foundation (NSF) award CCF-2315822.

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

References

1. Bucher, P.; Bairoch, A. A generalized profile syntax for biomolecular sequence motifs and its function in automatic sequence interpretation. In Proceedings of the Second International Conference on Intelligent Systems for Molecular Biology, Stanford University, Stanford, CA, USA, 14–17 August 1994; pp. 53–61.
2. Hofmann, K.; Bucher, P.; Falquet, L.; Bairoch, A. The PROSITE database, its status in 1999. *Nucleic Acids Res.* **1999**, *27*, 215–219. [CrossRef] [PubMed]
3. Fredriksson, K.; Grabowski, S. Efficient algorithms for pattern matching with general gaps, character classes, and transposition invariance. *Inf. Retr.* **2008**, *11*, 335–357. [CrossRef]
4. Myers, E.W. Approximate Matching of Network Expressions with Spacers. *J. Comput. Biol.* **1996**, *3*, 33–51. [CrossRef] [PubMed]
5. Mehldau, G.; Myers, G. A system for pattern matching applications on biosequences. *Bioinformatics* **1993**, *9*, 299–314. [CrossRef] [PubMed]
6. Navarro, G.; Raffinot, M. Fast and Simple Character Classes and Bounded Gaps Pattern Matching, with Applications to Protein Searching. *J. Comput. Biol.* **2003**, *10*, 903–923. [CrossRef] [PubMed]
7. Pissis, S.P. MoTeX-II: Structured MoTif eXtraction from large-scale datasets. *BMC Bioinform.* **2014**, *15*, 235. [CrossRef] [PubMed]
8. Miner, G.; Delen, D.; Elder, J.; Fast, A.; Hill, T.; Nisbet, R.A. *Practical Text Mining and Statistical Analysis for Non-Structured Text Data Applications*; Academic Press: Boston, MA, USA, 2012.
9. Kroeger, P.R. *Analyzing Grammar: An Introduction*; Cambridge University Press: Cambridge, UK, 2005.
10. Manning, C.D.; Schütze, H. *Foundations of Statistical Natural Language Processing*; MIT Press: Cambridge, MA, USA, 1999.

11. Willkomm, J.; Schäler, M.; Böhm, K. Accurate Cardinality Estimation of Co-occurring Words Using Suffix Trees. In Proceedings of the Database Systems for Advanced Applications: 26th International Conference, DASFAA 2021, Taipei, Taiwan, April 11–14 2021; Proceedings, Part II 26; Jensen, C.S., Lim, E.P., Yang, D.N., Lee, W.C., Tseng, V.S., Kalogeraki, V., Huang, J.W., Shen, C.Y., Eds.; Springer International Publishing: Cham, Switzerland, 2021; pp. 721–737.
12. Gusfield, D. *Algorithms on Strings, Trees, and Sequences: Computer Science and Computational Biology*; Cambridge University Press: Cambridge, UK, 1997.
13. Crochemore, M.; Hancart, C.; Lecroq, T. *Algorithms on Strings*; Cambridge University Press: Cambridge, UK, 2007.
14. Bille, P.; Gørtz, I.L.; Vildhøj, H.W.; Wind, D.K. String matching with variable length gaps. *Theor. Comput. Sci.* **2012**, *443*, 25–34. [CrossRef]
15. Bille, P.; Thorup, M. Regular Expression Matching with Multi-Strings and Intervals. In Proceedings of the Twenty-First Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2010, Austin, TX, USA, 17–19 January 2010; Charikar, M., Ed.; SIAM: Philadelphia, PA, USA, 2010; pp. 1297–1308. [CrossRef]
16. Morgante, M.; Policriti, A.; Vitacolonna, N.; Zuccolo, A. Structured Motifs Search. *J. Comput. Biol.* **2005**, *12*, 1065–1082. [CrossRef] [PubMed]
17. Bille, P.; Gørtz, I.L.; Lewenstein, M.; Pissis, S.P.; Rotenberg, E.; Steiner, T.A. Gapped String Indexing in Subquadratic Space and Sublinear Query Time. In Proceedings of the 41st International Symposium on Theoretical Aspects of Computer Science, STACS 2024, Clermont-Ferrand, France, 12–14 March 2024; Beyersdorff, O., Kanté, M.M., Kupferman, O., Lokshtanov, D., Eds.; Schloss Dagstuhl-Leibniz-Zentrum für Informatik: Dagstuhl, Germany, 2024; Volume 289, pp. 16:1–16:21. [CrossRef]
18. Peterlongo, P.; Allali, J.; Sagot, M. Indexing Gapped-Factors Using a Tree. *Int. J. Found. Comput. Sci.* **2008**, *19*, 71–87. [CrossRef]
19. Iliopoulos, C.S.; Rahman, M.S. Indexing Factors with Gaps. *Algorithmica* **2009**, *55*, 60–70. [CrossRef]
20. Lewenstein, M. Indexing with Gaps. In Proceedings of the String Processing and Information Retrieval, 18th International Symposium, SPIRE 2011, Pisa, Italy, 17–21 October 2011; Grossi, R., Sebastiani, F., Silvestri, F., Eds.; Springer: Berlin/Heidelberg, Germany, 2011, Volume 7024, pp. 135–143. [CrossRef]
21. Goldstein, I.; Lewenstein, M.; Porat, E. On the Hardness of Set Disjointness and Set Intersection with Bounded Universe. In Proceedings of the 30th International Symposium on Algorithms and Computation, ISAAC 2019, Shanghai University of Finance and Economics, Shanghai, China, 8–11 December 2019; Lu, P., Zhang, G., Eds.; Schloss Dagstuhl-Leibniz-Zentrum für Informatik: Dagstuhl, Germany, 2019; Volume 149, pp. 7:1–7:22. [CrossRef]
22. Bille, P.; Gørtz, I.L.; Pedersen, M.R.; Steiner, T.A. Gapped Indexing for Consecutive Occurrences. *Algorithmica* **2023**, *85*, 879–901. [CrossRef]
23. Ganguly, A.; Gibney, D.; MacNichol, P.; Thankachan, S.V. Bounded Ratio Gapped String Indexing. In Proceedings of the SPIRE 2024, Puerto Vallarta, Mexico, 23–25 September 2024.
24. Golovnev, A.; Guo, S.; Horel, T.; Park, S.; Vaikuntanathan, V. Data structures meet cryptography: 3SUM with preprocessing. In Proceedings of the 52nd Annual ACM SIGACT Symposium on Theory of Computing, STOC 2020, Chicago, IL, USA, 22–26 June 2020; Makarychev, K., Makarychev, Y., Tulsiani, M., Kamath, G., Chuzhoy, J., Eds.; ACM: New York, NY, USA, 2020; pp. 294–307. [CrossRef]
25. Kopelowitz, T.; Porat, E. The Strong 3SUM-INDEXING Conjecture is False. *arXiv* **2019**, arXiv:1907.11206.
26. Weiner, P. Linear Pattern Matching Algorithms. In Proceedings of the 14th Annual Symposium on Switching and Automata Theory, Iowa City, IA, USA, 15–17 October 1973; IEEE Computer Society: New York, NY, USA, 1973; pp. 1–11. [CrossRef]
27. Farach, M. Optimal Suffix Tree Construction with Large Alphabets. In Proceedings of the 38th Annual Symposium on Foundations of Computer Science, FOCS '97, Miami Beach, FL, USA, 19–22 October 1997; IEEE Computer Society: New York, NY, USA, 1997; pp. 137–143. [CrossRef]
28. Cohen, H.; Porat, E. Fast set intersection and two-patterns matching. *Theor. Comput. Sci.* **2010**, *411*, 3795–3800. [CrossRef]
29. Hon, W.; Shah, R.; Thankachan, S.V.; Vitter, J.S. String Retrieval for Multi-pattern Queries. In Proceedings of the String Processing and Information Retrieval—17th International Symposium, SPIRE 2010, Los Cabos, Mexico, 11–13 October 2010; Chávez, E., Lonardi, S., Eds.; Springer: Berlin/Heidelberg, Germany, 2010; Volume 6393, pp. 55–66. [CrossRef]