Gapped String Indexing in Subquadratic Space and Sublinear Query Time

Bille, Philip; Gørtz, Inge Li; Lewenstein, Moshe; Pissis, Solon P.; Rotenberg, Eva; Steiner, Teresa Anna

Published in:
Proceedings of the 41st International Symposium on Theoretical Aspects of Computer Science (STACS 2024)

Link to article, DOI:
10.4230/LIPIcs.STACS.2024.16

Publication date:
2024

Document Version
Publisher's PDF, also known as Version of record

Gapped String Indexing in Subquadratic Space and Sublinear Query Time

Philip Bille
Technical University of Denmark, Lyngby, Denmark

Inge Li Gørtz
Technical University of Denmark, Lyngby, Denmark

Moshe Lewenstein
Bar-Ilan University, Ramat-Gan, Israel

Solon P. Pissis
CWI, Amsterdam, The Netherlands
Vrije Universiteit, Amsterdam, The Netherlands

Eva Rotenberg
Technical University of Denmark, Lyngby, Denmark

Teresa Anna Steiner
Technical University of Denmark, Lyngby, Denmark

Abstract

In Gapped String Indexing, the goal is to compactly represent a string S of length n such that for any query consisting of two strings P_1 and P_2, called *patterns*, and an integer interval $[\alpha, \beta]$, called *gap range*, we can quickly find occurrences of P_1 and P_2 in S with distance in $[\alpha, \beta]$. Gapped String Indexing is a central problem in computational biology and text mining and has thus received significant research interest, including parameterized and heuristic approaches. Despite this interest, the best-known time-space trade-offs for Gapped String Indexing are the straightforward $\mathcal{O}(n)$ space and $\mathcal{O}(n + \text{occ})$ query time or $\Omega(n^2)$ space and $\tilde{\mathcal{O}}(|P_1| + |P_2| + \text{occ})$ query time.

We break through this barrier obtaining the first interesting trade-offs with polynomially subquadratic space and polynomially sublinear query time. In particular, we show that, for every $0 \leq \delta \leq 1$, there is a data structure for Gapped String Indexing with either $\tilde{\mathcal{O}}(n^{2-\delta/3})$ or $\tilde{\mathcal{O}}(n^{3-2\delta})$ space and $\tilde{\mathcal{O}}(|P_1| + |P_2| + n^{\delta} \cdot (\text{occ} + 1))$ query time, where occ is the number of reported occurrences.

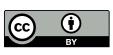
As a new fundamental tool towards obtaining our main result, we introduce the Shifted Set Intersection problem: preprocess a collection of sets $S_1, \ldots, S_k$ of integers such that for any query consisting of three integers i, j, s, we can quickly output YES if and only if there exist $a \in S_i$ and $b \in S_j$ with $a + s = b$. We start by showing that the Shifted Set Intersection problem is equivalent to the indexing variant of 3SUM (3SUM Indexing) [Golovnev et al., STOC 2020]. We then give a data structure for Shifted Set Intersection with gaps, which entails a solution to the Gapped String Indexing problem. Furthermore, we enhance our data structure for deciding Shifted Set Intersection, so that we can support the reporting variant of the problem, i.e., outputting all certificates in the affirmative case. Via the obtained equivalence to 3SUM Indexing, we thus give new improved data structures for the reporting variant of 3SUM Indexing, and we show how this improves upon the state-of-the-art solution for Jumbled Indexing [Chan and Lewenstein, STOC 2015] for any alphabet of constant size $\sigma > 5$.

2012 ACM Subject Classification Theory of computation → Pattern matching

Keywords and phrases data structures, string indexing, indexing with gaps, two patterns

Digital Object Identifier 10.4230/LIPIcs.STACS.2024.16

Related Version *Full Version*: https://arxiv.org/abs/2211.16860



Funding *Philip Bille*: Supported by the Independent Research Fund Denmark (DFF-9131-00069B).
Inge Li Gørtz: Supported by the Independent Research Fund Denmark (DFF-9131-00069B).
Solon P. Pissis: Supported by the PANGAIA (No 872539) and ALPACA (No 956229) projects.
Eva Rotenberg: Supported by the Danish Research Council grants DFF-9131-00069B "Adaptive Compressed Computation" and DFF-9131-00044B "Dynamic Network Analysis".
Teresa Anna Steiner: Supported by the VILLUM FONDEN grant VIL51463.

1 Introduction

The classic string indexing (or text indexing) problem [28, 19] is to preprocess a string S into a compact data structure that supports efficient pattern matching queries; i.e., *decide* if the pattern occurs or not in S or *report* the set of all positions in S where an occurrence of the pattern starts. An important variant of practical interest is the Gapped String Indexing problem; the goal is to preprocess a string S of length n into a compact data structure, such that for any query consisting of two patterns P_1 and P_2, and a gap range $[\alpha, \beta]$, one can quickly find occurrences of P_1 and P_2 in S with distance in $[\alpha, \beta]$.

Searching for patterns with gaps is of great importance in computational biology [11, 30, 24, 46, 43, 48, 50]. In DNA sequences, a structured DNA motif consists of two smaller conserved sites (patterns P_1 and P_2) separated by a spacer, that is, a non-conserved spacer of mostly fixed or slightly variable length (gap range $[\alpha, \beta]$). Thus, by introducing an efficient data structure for Gapped String Indexing, one can preprocess a DNA sequence into a compact data structure that facilitates efficient subsequent searches to DNA motifs.

Searching for patterns with gaps appears also in the area of text mining [42, 37, 44, 54]. Here, the task of finding co-occurrences of words is important, because they can indicate semantic proximity or idiomatic expressions in the text. A co-occurrence is a sequence of words (patterns $P_1, P_2, \ldots, P_k$) that occur in close proximity (gap range $[\alpha, \beta]$ or, most often, gap range $[0, \beta]$). By giving a data structure for Gapped String Indexing, one can pre-process large bodies of text into a compact data structure that facilitates efficient subsequent co-occurrence queries for the first non-trivial number of patterns, i.e., for $k = 2$.

The algorithmic version of the problem, where a string and a *single query* is given as input, is well-studied [10, 9, 24, 45, 48, 51]. The indexing version, which is arguably more useful in real-world applications, is also much more challenging: Standard techniques yield an $\mathcal{O}(n)$-space and $\mathcal{O}(n + \mathrm{occ})$-query time solution (see Appendix A) or an $\mathcal{O}(n^3)$-space and $\mathcal{O}(|P_1| + |P_2| + \mathrm{occ})$-query time solution, where occ is the size of the output. A more involved approach yields $\tilde{\mathcal{O}}(n^2)$ space and $\tilde{\mathcal{O}}(|P_1| + |P_2| + \mathrm{occ})$ query time (see Appendix B).

The Combinatorial Pattern Matching community has been unable to improve the above-mentioned long-standing trade-offs. Precisely because of this, practitioners have typically been engineering algorithms or studying heuristic approaches [4, 11, 12, 24, 29, 44, 45, 46, 48, 50, 54]. Theoreticians, on the other hand, have typically been solving restricted or parameterized variants [49, 5, 31, 38, 8, 35, 7, 17, 39, 25, 40]. Thus, breaking through the quadratic-space linear-time barrier for gapped string indexing is likely to have major consequences both in theory and in practice. This work is dedicated to answering the following question:

Is there a subquadratic-space and sublinear-query time solution for Gapped String Indexing?

1.1 Results

We assume throughout the standard word-RAM model of computation and answer the above question in the affirmative. Let us start by formally defining the *existence variant* of Gapped String Indexing as follows:

Gapped String Indexing
Preprocess: A string S of length n.
Query: Given two strings P_1 and P_2 and an integer interval $[\alpha, \beta]$, output YES if and only if there exists a pair (i, j) such that P_1 occurs at position i of S, P_2 occurs at position $j \geq i$ of S and $j - i \in [\alpha, \beta]$.

Similarly, in the *reporting variant* of Gapped String Indexing, a query answer is all pairs satisfying the above conditions.

Gapped String Indexing w. Reporting
Preprocess: A string S of length n.
Query: Given two strings P_1 and P_2 and an integer interval $[\alpha, \beta]$, output all pairs (i, j) such that P_1 occurs at position i of S, P_2 occurs at position $j \geq i$ of S and $j - i \in [\alpha, \beta]$.

Our main result is the following trade-offs for Gapped String Indexing with reporting:

▶ **Theorem 1.** *For every $0 \leq \delta \leq 1$, there is a data structure for Gapped String Indexing w. Reporting with either:*

(i) $\tilde{\mathcal{O}}(n^{2-\delta/3})$ *space and* $\tilde{\mathcal{O}}(n^{\delta} \cdot (\mathrm{occ} + 1) + |P_1| + |P_2|)$ *query time; or*

(ii) $\tilde{\mathcal{O}}(n^{3-2\delta})$ *space and* $\tilde{\mathcal{O}}(n^{\delta} \cdot (\mathrm{occ} + 1) + |P_1| + |P_2|)$ *query time,*

where occ *is the size of the output.*

For the existence variant, the bounds above hold with $\mathrm{occ} = 1$: we can terminate the querying algorithm as soon as the first witness is reported. (Note that $n^{3-2\delta}$ is smaller than $n^{2-\delta/3}$ for $\delta > 3/5$.) Hence, we achieve the first polynomially *subquadratic space* and polynomially *sublinear query time* for Gapped String Indexing.

Our main technical contribution is a data structure for Gapped Set Intersection w. Reporting, which is a generalization of a problem related to 3SUM Indexing. We then show that our improved result for Gapped String Indexing w. Reporting follows via new connections to Gapped Set Intersection w. Reporting.

We show that the Gapped Set Intersection and 3SUM Indexing problems (and their reporting variants) are equivalent, but with an increase in universe size in the reduction to 3SUM Indexing. Thus, as a result, we obtain a new data structure for the indexing variant of the 3SUM problem [27], which not only outputs an arbitrary certificate, but it outputs *all certificates*. This is an interesting contribution on its own right and also has applications e.g., to the Jumbled Indexing problem [14].

1.2 Overview of Techniques and Paper Organization

As a new fundamental tool towards obtaining our main result for Gapped String Indexing, we introduce the following problem, which we call Shifted Set Intersection (see Figure 1).

Shifted Set Intersection
Preprocess: A collection of k sets $S_1, \ldots, S_k$ of total size $\sum_i |S_i| = N$ of integers from a universe $U = \{1, 2, \ldots, u\}$.
Query: Given i, j, s, output YES if and only if there exist $a \in S_i$ and $b \in S_j$ such that $a + s = b$.

We start by showing that the Shifted Set Intersection problem is equivalent to the indexing variant of 3SUM. We formally define the 3SUM Indexing problem next.

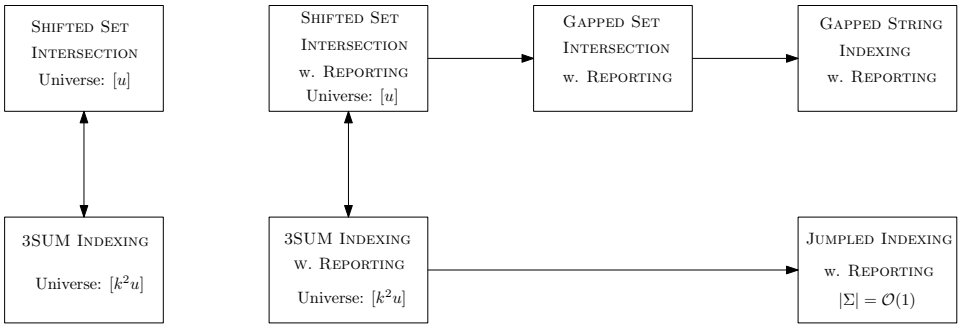


Figure 1 We show that Shifted Set Intersection and 3SUM Indexing are equivalent, and our reduction also transfers to the reporting variants of the two problems. In fact, we solve a problem that is even harder than the Shifted Set Intersection problem; namely, the Gapped Set Intersection problem, which leads to a solution to the Gapped String Indexing problem.

3SUM Indexing
Preprocess: A set A of N integers from a universe $U = \{1, 2, \ldots, u\}$.
Query: Given $c \in U$ output YES if and only if there exist $a, b \in A$ such that $a + b = c$.

The following breakthrough result is known for 3SUM Indexing; see also [36].

▶ **Theorem 2** (Golovnev et al. [27]). *For every $0 \leq \delta \leq 1$, there is a data structure for 3SUM Indexing with space $\tilde{\mathcal{O}}(N^{2-\delta})$ and query time $\tilde{\mathcal{O}}(N^{3\delta})$. In particular, this data structure returns a* certificate*, i.e., for a query c, such that the answer is YES, it can output a pair $a, b \in A$ such that $a + b = c$ in $\tilde{\mathcal{O}}(N^{3\delta})$ time.*

In Section 2, we show a two-way linear-time reduction between Shifted Set Intersection and 3SUM Indexing. In particular, this tells us that the two problems admit the same space-query time trade-offs. While the direction from 3SUM Indexing to Shifted Set Intersection is immediate, the opposite direction requires some careful manipulation of the input collection based on the underlying universe. Furthermore, in the same section, we give a different trade-off for Shifted Set Intersection in the case where the set of possible distances is small, which is based on tabulating large input sets.

The main advantage of the Shifted Set Intersection formulation is that it gives extra flexibility which we can exploit to solve several generalizations of the problem. In particular, in Section 3, we show that we can augment any Shifted Set Intersection instance of k sets and total size N by $\mathcal{O}(k \log N)$ extra sets, such that we can represent any subset of consecutive elements in a set of the original instance by at most $\mathcal{O}(\log N)$ sets in the augmented instance. We use this to solve the reporting variant of the Shifted Set Intersection problem, called Shifted Set Intersection w. Reporting, where instead of deciding whether two elements of a given shift exist, we report all such elements. The idea is to first locate one output pair using the solution for the existence variant, then split the sets into elements smaller or bigger than the output element, and recurse accordingly.

Shifted Set Intersection w. Reporting
Preprocess: A collection of k sets $S_1, \ldots, S_k$ of total size $\sum_i |S_i| = N$ of integers from a universe $U = \{1, 2, \ldots, u\}$.
Query: Given i, j, s, output all pairs (a, b) such that $a \in S_i$ and $b \in S_j$ and $a + s = b$.

We show the following result.

▶ **Theorem 3.** *For every $0 \leq \delta \leq 1$, there is a data structure for* Shifted Set Intersection *w.* Reporting *with $\tilde{\mathcal{O}}(N^{2-\delta/3})$ space and $\tilde{\mathcal{O}}(N^{\delta} \cdot (\mathrm{occ}+1))$ query time, where* occ *is the size of the output.*

As a consequence, we also give a reporting data structure for 3SUM Indexing, which may be of independent interest. Let us first formalize the problem.

3SUM Indexing w. Reporting
Preprocess: A set A of N integers from a universe $U = \{1, 2, \ldots, u\}$.
Query: Given $c \in U$ output all pairs $(a, b) \in A$ such that $a + b = c$.

▶ **Corollary 4.** *For every $0 \leq \delta \leq 1$, there is a data structure for 3SUM* Indexing *w.* Reporting *with $\tilde{\mathcal{O}}(N^{2-\delta/3})$ space and $\tilde{\mathcal{O}}(N^{\delta} \cdot (\mathrm{occ}+1))$ query time, where* occ *is the size of the output.*

Furthermore, in Section 4, we show that we can augment any Shifted Set Intersection instance by $\mathcal{O}(k \log u)$ sets to solve the more general problem of Gapped Set Intersection, where instead of a *single shift*, the query asks for an interval of allowed shifts.

Gapped Set Intersection
Preprocess: A collection of k sets $S_1, \ldots, S_k$ of total size $\sum_i |S_i| = N$ of integers from a universe $U = \{1, 2, \ldots, u\}$.
Query: Given i, j and an integer interval $[\alpha, \beta]$, output YES if and only if there exist $a \in S_i$, $b \in S_j$ and $s \in [\alpha, \beta]$ such that $a + s = b$.

We solve this problem by first considering an approximate variant of the problem, where the interval length is a power of two plus one, and we allow false positives in an interval of roughly twice the size. Then we carefully cover the query interval by $\mathcal{O}(\log u)$ approximate queries to obtain Theorem 5.

▶ **Theorem 5.** *For every $0 \leq \delta \leq 1$, there is a data structure for* Gapped Set Intersection *with $\tilde{\mathcal{O}}(N^{2-\delta/3})$ space and $\tilde{\mathcal{O}}(N^{\delta})$ query time.*

We combine the reduction underlying Theorem 5 with the reporting solution underlying Theorem 3 to obtain a solution for the reporting variant of Gapped Set Intersection.

Gapped Set Intersection w. Reporting
Preprocess: A collection of k sets $S_1, \ldots, S_k$ of total size $\sum_i |S_i| = N$ of integers from a universe $U = \{1, 2, \ldots, u\}$.
Query: Given i, j and an integer interval $[\alpha, \beta]$, output all pairs (a, b) such that $a \in S_i$, $b \in S_j$ and there is an $s \in [\alpha, \beta]$ such that $a + s = b$.

▶ **Theorem 6.** *For every $0 \leq \delta \leq 1$, there is a data structure for* Gapped Set Intersection *w.* Reporting *with $\tilde{\mathcal{O}}(N^{2-\delta/3})$ space and $\tilde{\mathcal{O}}(N^{\delta} \cdot (\mathrm{occ}+1))$ query time, where* occ *is the size of the output.*

In Section 5, we finally use all of these acquired tools to solve the Gapped String Indexing problem. In particular, we cover the suffix array of string S in dyadic subintervals, which we then preprocess into the data structure from Theorem 6. Note that the total size of these sets is $\mathcal{O}(n \log n)$, where n is the length of S. Moreover, any interval of consecutive elements in the suffix array can be covered by $\mathcal{O}(\log n)$ sets in the instance. Putting everything together we obtain our main result (Theorem 1).

In Section 6, we show a reduction from 3SUM Indexing to Jumbled Indexing, giving a data structure for Jumbled Indexing which improves on the state-of-the-art solution for constant alphabet sizes $\sigma > 5$. We give new upper bounds for the existence and the reporting variants of Jumbled Indexing. In order to define Jumbled Indexing, we need the following notion of a *histogram* of a string:

▶ **Definition 7.** *Given a string S over an alphabet Σ, a histogram h of S is a vector of dimension $|\Sigma|$ where every entry is the number of times the corresponding letter occurs in S.*

Jumbled Indexing w. Reporting
Preprocess: A string S of length n over an alphabet Σ.
Query: For any pattern histogram $P \in (\mathbb{N}_0)^{\lvert\Sigma\rvert}$, return all substrings of S whose histogram is P.

We call a substring of S whose histogram is P an *occurrence of P in S*. In the *existence variant* of the Jumbled Indexing problem, the query answer is YES or NO depending on whether there is at least one occurrence of P in S. We show the following result:

▶ **Corollary 8.** *Given a string S of length n over an alphabet of size $\sigma = \mathcal{O}(1)$, for every $0 \leq \delta \leq 1$, there is a data structure for Jumbled Indexing w. Reporting with $\tilde{\mathcal{O}}(n^{2-\delta})$ space and $\tilde{\mathcal{O}}(n^{3\delta} \cdot (\mathrm{occ}+1))$ query time, where* occ *is the size of the output. The bounds are $\tilde{\mathcal{O}}(n^{2-\delta})$ space and $\tilde{\mathcal{O}}(n^{3\delta})$ query time for the existence variant.*

The best-known previous bounds for the existence variant of Jumbled Indexing use $\mathcal{O}(n^{2-\delta})$ space and $\mathcal{O}(m^{(2\sigma-1)\delta})$ query time [34], where m is the norm of the queried pattern (i.e., the sum of histogram entries), or $\tilde{\mathcal{O}}(n^{2-\delta})$ space and $\tilde{\mathcal{O}}(n^{\delta(\sigma+1)/2})$ query time [14]. Interestingly, the reporting variant of Jumbled Indexing is, in general, significantly harder than the existence variant: There is a recent (unconditional) lower bound stating that any data structure for Jumbled Indexing w. Reporting with $\mathcal{O}(n^{0.5-o(1)} + \mathrm{occ})$ query time needs $\Omega(n^{2-o(1)})$ space, and this holds even for a binary alphabet [1]. By our reduction, this in particular implies that a data structure with $\tilde{\mathcal{O}}(N^{2-\delta})$ space and $\tilde{\mathcal{O}}(N^{3\delta} + \mathrm{occ})$ query time is not possible for 3SUM Indexing w. Reporting. For a more complete overview, see Section 6. We conclude this paper in Section 7 with some future proposals.

In Appendix C, we show a better trade-off for the related Smallest Shift problem.

Smallest Shift
Preprocess: A collection of k sets $S_1, \ldots, S_k$ of total size $\sum_i \lvert S_i\rvert = N$ of integers from a universe $U = \{1, 2, \ldots, u\}$.
Query: Given i, j, output the smallest s such that there exists $a \in S_i$ and $b \in S_j$ with $a + s = b$.

▶ **Proposition 9.** *There is a data structure for Smallest Shift with $\mathcal{O}(N)$ space and $\tilde{\mathcal{O}}(\sqrt{N})$ query time. Moreover, the data structure can be constructed in $\mathcal{O}(N\sqrt{N})$ time.*

Gapped Set Intersection reduces to Smallest Shift in the special case where we only allow query intervals of the form $[0, \beta]$. Together with our other results, this reduction yields a $\tilde{\mathcal{O}}(N)$ space and $\tilde{\mathcal{O}}(|P_1| + |P_2| + \sqrt{N} \cdot (\mathrm{occ}+1))$ query time trade-off for Gapped String Indexing if the query intervals are restricted to $[0, \beta]$. However, let us remark that for this restricted version of the problem, a $\tilde{\mathcal{O}}(N)$ space and $\tilde{\mathcal{O}}(|P_1|+|P_2|+\sqrt{N \cdot (\mathrm{occ}+1)}+\mathrm{occ})$ query time trade-off already follows from [7].

Subsequent work. Since the first publication of this paper [6], Aronov et al. [3] proposed a combination of range searching and the Fiat-Naor inversion scheme [21] (which was has already been used to prove Theorem 2), recovering our main result (Theorem 1) as a corollary.

2 3SUM Indexing is Equivalent to Shifted Set Intersection

Here we show a two-way linear-time reduction between 3SUM Indexing and Shifted Set Intersection. We thus obtain the same space-time bounds for the two problems.

2.1 From 3SUM Indexing to Shifted Set Intersection

▶ **Fact 10.** *Any instance of 3SUM Indexing of input size N and universe size u can be reduced to a Shifted Set Intersection instance of total size $\mathcal{O}(N)$ with universe size $\mathcal{O}(u)$ in time $\mathcal{O}(N)$.*

Proof. We denote the 3SUM Indexing instance by $A = \{a_1, a_2, \ldots, a_N\}$ (the input set). We construct the Shifted Set Intersection instance: $S_1 = A$ and $S_2 = \{-a_1, -a_2, \ldots, -a_N\}$ in $\mathcal{O}(N)$ time. For any 3SUM Indexing query c, we construct the Shifted Set Intersection query (S_2, S_1, c)[1] in $\mathcal{O}(1)$ time, and observe that: $a_i + a_j = c \iff c - a_i = a_j$. Thus the answer to the 3SUM Indexing query is YES if and only if the answer to the Shifted Set Intersection query is YES. ◀

2.2 From Shifted Set Intersection to 3SUM Indexing

▶ **Theorem 11.** *Any instance of Shifted Set Intersection with $\sum_{i=1}^{k} |S_i| = N$ and universe size u can be reduced to a 3SUM Indexing instance of input size $\mathcal{O}(N)$ and universe size $\mathcal{O}(k^2 u)$ in time $\mathcal{O}(N)$.*

Proof. Let us start by considering an alternative yet equivalent formulation of 3SUM Indexing. Preprocess two sets A and B from a universe $U' = \{1, 2, \ldots, u'\}$ to answer queries of the following form: Given an element $c \in U'$, output YES if and only if there exist $(a, b) \in A \times B$ such that $a + b = c$. To see why it is equivalent, notice that the formulation with one set reduces trivially to this formulation by setting $A = B$. For the other direction, given A and B from universe U', define $A' = A \cup \{b + 2u' \mid b \in B\}$. We verify that querying $c + 2u'$ on A' for any $c \in [2, \ldots, 2u']$ in the 3SUM Indexing formulation with one set is equivalent to querying c on A and B in the 3SUM Indexing formulation with two sets. We now reduce Shifted Set Intersection to the 3SUM Indexing formulation with two sets.

We denote the Shifted Set Intersection instance by $S_1, \ldots, S_k$ (the input collection over universe $U = \{1, 2, \ldots, u\}$) and $\sum_{i \in [k]} |S_i|$ by N. We construct the following instance of the above 3SUM Indexing reformulation in $\mathcal{O}(N + k) = \mathcal{O}(N)$ time:

$$A = \{e + j \cdot (k+1) \cdot 2u \mid e \in S_j,\ 1 \le j \le k\},\ B = \{-e + i \cdot 2u \mid e \in S_i,\ 1 \le i \le k\}.$$

The 3SUM Indexing instance has input size $2N = \Theta(N)$ and universe size $U' = \mathcal{O}(k^2 \cdot u)$.

Let us denote a query of Shifted Set Intersection by $Q(S_i, S_j, s)$. Now, we construct the 3SUM Indexing query $Q_{\text{3SUM}}(s + (j \cdot (k+1) + i) \cdot 2u)$ in $\mathcal{O}(1)$ time. The following claim concludes the proof.

[1] We may write (S_i, S_j, c) for a Shifted Set Intersection query instead of (i, j, c) for clarity.

▷ Claim 12. $Q(S_i, S_j, s)$ outputs YES if and only if $Q_{3\text{SUM}}(s + (j \cdot (k+1) + i) \cdot 2u)$ outputs YES.

Proof. ($\Rightarrow$): Let $e_1 \in S_i$, $e_2 \in S_j$ such that $e_1 + s = e_2$. Then $a = e_2 + j \cdot (k+1) \cdot 2u \in A$, $b = -e_1 + i \cdot 2u \in B$, and $a + b = s + (j \cdot (k+1) + i) \cdot 2u$. Thus $Q_{3\text{SUM}}(s + (j \cdot (k+1) + i) \cdot 2u)$ outputs YES.

($\Leftarrow$): Assume there is $a \in A$ and $b \in B$ such that $a + b = s + (j \cdot (k+1) + i) \cdot 2u$. By definition of A and B, there exist i', j', and $e_2 \in S_{j'}$, $e_1 \in S_{i'}$ such that $a = e_2 + j' \cdot (k+1) \cdot 2u$ and $b = -e_1 + i' \cdot 2u$, thus $s + (j \cdot (k+1) + i) \cdot 2u = (e_2 - e_1) + (j' \cdot (k+1) + i') \cdot 2u$. Since $-u < s < u$ and $-u < (e_2 - e_1) < u$, we have that $j \cdot (k+1) + i = j' \cdot (k+1) + i'$ and $e_2 - e_1 = s$. Since $i \leq k$ and $i' \leq k$, we have $j = j'$ and $i = i'$. Thus $Q(S_i, S_j, s)$ outputs YES. ◁

◀

By employing Theorem 2 we obtain the following corollary:

▶ **Corollary 13.** *For every $0 \leq \delta \leq 1$, there is a data structure for Shifted Set Intersection with space $\tilde{\mathcal{O}}(N^{2-\delta})$ and query time $\tilde{\mathcal{O}}(N^{3\delta})$.*

We also show how we can obtain a different trade-off in the case where the number of possible shifts is bounded by some Δ (this gives us a better trade-off for some δ in the application of Gapped String Indexing). In this case, call all sets of size at most N^δ *small*, and other sets *large*. Note that there are at most $N^{1-\delta}$ large sets. For every pair of large sets and any possible shift, we precompute the answer, using space $\mathcal{O}(\Delta N^{2-2\delta})$. Additionally, we store a dictionary on every set using space $\mathcal{O}(N)$ (using e.g., perfect hashing [23]). For a query $Q(S_i, S_j, s)$, if both sets S_i and S_j are large, we look up the precomputed answer. If one set is small, wlog S_i, we check if $a + s \in S_j$ for every $a \in S_i$ using the dictionary. Note that in particular, $\Delta < u$. This gives the following lemma:

▶ **Lemma 14.** *For every $0 \leq \delta \leq 1$, there is a data structure for Shifted Set Intersection with space $\mathcal{O}(u \cdot N^{2-2\delta})$ and query time $\mathcal{O}(N^\delta)$.*

3 Reporting: 3SUM Indexing and Shifted Set Intersection

In this section, we explain how we can answer reporting queries for both Shifted Set Intersection and 3SUM Indexing, as defined in Section 1. The following fact is trivial.

▶ **Fact 15.** *There is a data structure for Shifted Set Intersection w. Reporting with $\mathcal{O}(N)$ space and $\mathcal{O}(|S_i| + |S_j|)$ query time.*

The other extreme trade-off is also straightforward.

▶ **Lemma 16.** *There is a data structure for Shifted Set Intersection w. Reporting with $\mathcal{O}(N^2)$ space and $\mathcal{O}(\text{occ})$ query time, where* occ *is the size of the output.*

Proof. For every existing shift s, we save all the pairs of sets (A, B) for which $Q(A, B, s) =$YES using perfect hashing [23]. For every such pair, we save all pairs $(a, b) \in A \times B$, such that $a + s = b$, in a list. There are $\mathcal{O}(N^2)$ pairs (a, b) in total. The space is thus $\mathcal{O}(N^2)$. ◀

We next prove the main result (Theorem 3) of this section.

▶ **Theorem 3.** *For every $0 \leq \delta \leq 1$, there is a data structure for* Shifted Set Intersection w. Reporting *with $\tilde{\mathcal{O}}(N^{2-\delta/3})$ space and $\tilde{\mathcal{O}}(N^{\delta} \cdot (\mathrm{occ}+1))$ query time, where* occ *is the size of the output.*

▶ **Corollary 4.** *For every $0 \leq \delta \leq 1$, there is a data structure for 3SUM* Indexing w. Reporting *with $\tilde{\mathcal{O}}(N^{2-\delta/3})$ space and $\tilde{\mathcal{O}}(N^{\delta} \cdot (\mathrm{occ}+1))$ query time, where* occ *is the size of the output.*

Proof of Theorem 3. The main idea behind the proof is the following: We use the fact that the data structure for 3SUM Indexing from Theorem 2 returns a *certificate*, i.e., for a query $c \in U$ such that the answer to the query is YES it can additionally output a pair (a, b) satisfying $a + b = c$. By our reduction from Shifted Set Intersection, when we query for S_i, S_j and s we can return $a \in S_i$ and $b \in S_j$ such that $a + s = b$. We then conceptually split S_i and S_j into two sets each, those elements in S_i which are smaller than a, those which are bigger, and similarly for S_j and b. We then want to use the fact that if $a' + s = b'$ and $a' < a$, then $b' < b$ and vice versa, to recurse on the smaller subsets. However, we cannot afford to preprocess all subsets of any set S_i into the Shifted Set Intersection data structure. To solve this issue, we partition each set into the subsets which correspond to the dyadic intervals on the rank of the elements in sorted order, and preprocess them into our Shifted Set Intersection data structure.

In detail, our data structure is defined as follows: Let $S_1, \ldots, S_k$ be the input to Shifted Set Intersection w. Reporting. We build the data structure for Shifted Set Intersection from Corollary 13 containing, additionally to $S_1, \ldots, S_k$, the following subsets: Let $S = \{s_1, s_2, \ldots, s_m\}$ be a set in our Shifted Set Intersection w. Reporting instance, where $s_1 < s_2 < \cdots < s_m$.

We partition S into subsets which correspond to the dyadic intervals on the rank. That is, for $j = 0, \ldots, \lfloor \log m \rfloor$, we partition S into $\{s_{i+1}, s_{i+2}, \ldots, s_{i+2^j}\}$, for all $i = \kappa \cdot 2^j$ and $0 \leq \kappa \leq \lfloor m/2^j \rfloor - 1$. We call these sets the *dyadic subsets* of S. Note that for a fixed j, the union of these sets is a subset of S and thus has size at most m. Hence, the total size of all dyadic subsets is $\mathcal{O}(m \log m)$.

▶ **Observation 17.** *Any subset of S of the form $\{x \in S \mid a \leq x \leq b\}$ is the union of at most $2 \log |S|$ dyadic subsets of S.*

The data structure for Shifted Set Intersection w. Reporting consists of the Shifted Set Intersection data structure on all sets $S_1, \ldots, S_k$ and their dyadic subsets. Further, for each dyadic subset, we store its minimum and its maximum element as auxiliary information. The total size of all dyadic subsets is $\mathcal{O}(N \log N)$. The space is thus $\tilde{\mathcal{O}}(N^{2-\delta/3})$.

To perform a query on S_i, S_j, s, we first perform a query on the Shifted Set Intersection data structure for the same inputs. If it returns NO, we are done. If it returns YES, let a and b be the certificate pair such that $a + s = b$. Define $A_1 = \{a' \in S_i \mid a' < a\}$ and $A_2 = \{a' \in S_i \mid a' > a\}$, and similarly $B_1 = \{b' \in S_j \mid b' < b\}$ and $B_2 = \{b' \in S_j \mid b' > b\}$. Then any additional solution pair (a', b') has to either satisfy $a' \in A_1$ and $b' \in B_1$ or $a' \in A_2$ and $b' \in B_2$. Further, by Observation 17, we can decompose A_1, A_2, B_1, and B_2 into $\mathcal{O}(\log N)$ dyadic subsets in $\mathcal{O}(\log N)$ time. Call these decompositions $\mathcal{A}_1$, $\mathcal{A}_2$, $\mathcal{B}_1$, $\mathcal{B}_2$. We could now recurse on any (A, B) where $A \in \mathcal{A}_1$ and $B \in \mathcal{B}_1$ or $A \in \mathcal{A}_2$ and $B \in \mathcal{B}_2$, dividing into $\mathcal{O}(\log^2 N)$ subproblems. Since every time before we recurse we obtain a new certificate, this brings the query time to $\tilde{\mathcal{O}}(N^{\delta} \cdot (\mathrm{occ}+1))$, which proves the theorem. We nevertheless show next (Lemma 18) how we can reduce the recursion pairs to $\mathcal{O}(\log N)$.

Fix $i \in \{1,2\}$. For $A' \in \mathcal{A}_i$, let $a_{\min}$ be its minimum element and $a_{\max}$ its maximum element. For $B' \in \mathcal{B}_i$ define $b_{\min}$ and $b_{\max}$ analogously. We say A' and B' *match* if $[a_{\min}+s, a_{\max}+s]$ and $[b_{\min}, b_{\max}]$ have a non-empty intersection.

▶ **Lemma 18.** *There are $\mathcal{O}(\log N)$ matching pairs (A', B'), $A' \in \mathcal{A}_i$, $B' \in \mathcal{B}_i$, and $i \in \{1,2\}$.*

Proof. For a fixed $A' \in \mathcal{A}_i$, consider all B' such that A' matches $B' \in \mathcal{B}_i$. Since the intervals $[b_{\min}, b_{\max}]$ are disjoint, all except at most two B' that match A' have the property that the corresponding interval $[b_{\min}, b_{\max}]$ is fully contained in $[a_{\min}+s, a_{\max}+s]$. Hence, those B' match *only* A'. Let $\mathcal{A}_i = \{A_i^1, \ldots, A_i^{k_1}\}$ and $|\mathcal{B}_i| = k_2$ with $k_1, k_2 = \mathcal{O}(\log N)$. The number of matching pairs is given by

$$\sum_{j=1}^{k_1}(\text{Number of } B' \text{ matching } A_i^j) \leq 2k_1 + \sum_{j=1}^{k_1}(\text{Number of } B' \text{ matching only } A_i^j) \leq 2k_1 + k_2.$$

The last inequality follows since the sets $\{B' \text{ matching only } A_i^j\}$ form disjoint subsets of $\mathcal{B}_i$. ◀

For one A', we can identify all matching B' in $\mathcal{O}(\log N)$ time using the precomputed information (i.e., searching for the predecessor of $a_{\min}+s$ in the minima of B_i and the successor of $a_{\max}+s$ in the maxima of B_i). In total all matching pairs can be identified in time $\mathcal{O}(\log^2 N)$. Thus, instead of recursing on all pairs (A', B'), where $A' \in \mathcal{A}_i$ and $B' \in \mathcal{B}_i$, we can recurse only on the matching pairs, which saves a $\log N$ factor in the query time. ◀

4 Gapped Set Intersection

Here we show how to solve the more general problem of Gapped Set Intersection, where we allow any shift within a given interval. Recall that the problem is defined in Section 1.

The main idea is to use the solution from Corollary 13 on $\mathcal{O}(\log u)$ carefully constructed Shifted Set Intersection instances. To do this, we first show how we can approximately answer Gapped Set Intersection queries for query intervals whose length is a power of two plus one (approximately in the sense that we allow false positives within a larger interval), then use $\mathcal{O}(\log u)$ such queries to "cover" $[\alpha, \beta]$.

We formally define *approximate* Gapped Set Intersection.

Approximate Gapped Set Intersection
Preprocess: A collection of k sets $S_1, \ldots, S_k$ of total size $\sum_i |S_i| = N$ of integers from a universe $U = \{1, 2, \ldots, u\}$ and a *level* l with $1 \leq l \leq \log u$.
Query: Given i, j and a *center distance* $d = \kappa \cdot 2^l$, for some positive integer κ, output YES or NO such that:
- The output is YES if there exists a pair $a \in S_i$, $b \in S_j$ such that $b - a \in [d - 2^{l-1}, d + 2^{l-1}]$;
- The output is NO if there exists no pair $a \in S_i$, $b \in S_j$ such that $b - a \in (d - 2^l, d + 2^l)$.

Note that by the definition of the Approximate Gapped Set Intersection problem, if for a query i, j and $d = \kappa \cdot 2^l$ there exists a pair a pair $a \in S_i$, $b \in S_j$ such that $b - a \in (d - 2^l, d + 2^l)$, but no pair $a \in S_i$, $b \in S_j$ such that $b - a \in [d - 2^{l-1}, d + 2^{l-1}]$, we may answer either YES or NO.

▶ **Lemma 19.** *Assume there is a data structure for Shifted Set Intersection with s space and t query time. Then there is a data structure for Approximate Gapped Set Intersection with $\mathcal{O}(s)$ space and $\mathcal{O}(t)$ query time.*

Proof. For any set S_i in the collection, define $S'_i = \{\lfloor a/2^{l-1} \rfloor, a \in S_i\}$ and build the assumed data structure for Shifted Set Intersection on sets $S'_1, \ldots, S'_k$. To answer a query $(i, j, d = \kappa \cdot 2^l)$ on Approximate Gapped Set Intersection, make the queries $(i, j, 2\kappa - 1)$, $(i, j, 2\kappa)$ and $(i, j, 2\kappa + 1)$ on the Shifted Set Intersection instance. Return YES if and only if one of the queries to the Shifted Set Intersection instance returns YES.

To show correctness, first notice that if we return YES, then there exist $a \in S_i$ and $b \in S_j$ satisfying $2\kappa - 1 \leq \lfloor b/2^{l-1} \rfloor - \lfloor a/2^{l-1} \rfloor \leq 2\kappa + 1$. This implies $2\kappa - 2 \leq \lfloor b/2^{l-1} \rfloor - \lfloor a/2^{l-1} \rfloor - 1 < \lfloor b/2^{l-1} \rfloor - a/2^{l-1} \leq b/2^{l-1} - a/2^{l-1}$ and $b/2^{l-1} - a/2^{l-1} \leq b/2^{l-1} - \lfloor a/2^{l-1} \rfloor < \lfloor b/2^{l-1} \rfloor - \lfloor a/2^{l-1} \rfloor + 1 \leq 2\kappa + 2$. Thus, $b - a \in (\kappa \cdot 2^l - 2^l, \kappa \cdot 2^l + 2^l) = (d - 2^l, d + 2^l)$.

Next, assume there is a pair $a \in S_i$ and $b \in S_j$ such that $b - a \in [d - 2^{l-1}, d + 2^{l-1}] = [\kappa \cdot 2^l - 2^{l-1}, \kappa \cdot 2^l + 2^{l-1}]$. Then clearly $b/2^{l-1} - a/2^{l-1} \in [2\kappa - 1, 2\kappa + 1]$. Similar to above, we have $\lfloor b/2^{l-1} \rfloor - \lfloor a/2^{l-1} \rfloor \leq b/2^{l-1} - \lfloor a/2^{l-1} \rfloor < b/2^{l-1} - a/2^{l-1} + 1 \leq 2\kappa + 2$ and $2\kappa - 2 \leq b/2^{l-1} - a/2^{l-1} - 1 < \lfloor b/2^{l-1} \rfloor - a/2^{l-1} \leq \lfloor b/2^{l-1} \rfloor - \lfloor a/2^{l-1} \rfloor$. Thus, $\lfloor b/2^{l-1} \rfloor - \lfloor a/2^{l-1} \rfloor \in (2\kappa - 2, 2\kappa + 2)$ and, because $\lfloor b/2^{l-1} \rfloor - \lfloor a/2^{l-1} \rfloor$ is an integer, $\lfloor b/2^{l-1} \rfloor - \lfloor a/2^{l-1} \rfloor \in [2\kappa - 1, 2\kappa + 1]$. Thus we answer YES in this case. ◀

▶ **Corollary 20.** *For every $0 \leq \delta \leq 1$, there is a data structure for Approximate Gapped Set Intersection with $\tilde{\mathcal{O}}(N^{2-\delta/3})$ space and $\tilde{\mathcal{O}}(N^{\delta})$ query time.*

We now show how to reduce Gapped Set Intersection to $\mathcal{O}(\log u)$ Approximate Gapped Set Intersection instances, giving the following theorem:

▶ **Theorem 5.** *For every $0 \leq \delta \leq 1$, there is a data structure for Gapped Set Intersection with $\tilde{\mathcal{O}}(N^{2-\delta/3})$ space and $\tilde{\mathcal{O}}(N^{\delta})$ query time.*

Proof. Given $S_1, \ldots, S_k$, we build a Shifted Set Intersection data structure and an Approximate Gapped Set Intersection data structure for $S_1, \ldots, S_k$ and every level l with $1 \leq l \leq \log u$. Assume we want to answer a query for S_i, S_j and $[\alpha, \beta]$. We show how to answer the query using $\mathcal{O}(\log(\beta - \alpha))$ Approximate Gapped Set Intersection queries. We query Approximate Gapped Set Intersection for i, j and different choices of l and d. We call a query i, j and d to Approximate Gapped Set Intersection of level l a *2^l-approximate query centered at d*. We say the query *covers* the interval $[d - 2^{l-1}, d + 2^{l-1}]$. We call all elements in interval $(d - 2^l, d + 2^l)$ *uncertain*. Now, for the interval $[\alpha, \beta]$, we want to find $\mathcal{O}(\log(\beta - \alpha)) = \mathcal{O}(\log u)$ queries which together *cover* $[\alpha, \beta]$ such that there are no uncertain elements outside of $[\alpha, \beta]$. In detail:

We show how to cover a continuous interval $[\alpha, \alpha + \Delta]$ for growing Δ over several *phases* (inspect Figure 2). In the lth phase, we use a constant number of 2^l-approximate queries.

- If at any point we cover $[\alpha, \alpha + \Delta]$ such that $\Delta \geq (\beta - \alpha)/2$, we stop.
- If we never cover past $\alpha + \Delta$ with $\Delta \geq (\beta - \alpha)/2$ in phase l, we do exactly three 2^l-approximate queries in that phase:
- In phase 0, we do regular Shifted Set Intersection queries for α, $\alpha + 1$, $\alpha + 2$.
- In phase l, if we have covered up until $\alpha + \Delta$ in phase $l - 1$, we center the first 2^l-approximate query at the largest $\kappa \cdot 2^l$ such that $\kappa \cdot 2^l - 2^{l-1} \leq \alpha + \Delta$. Then we center the next two queries at $(\kappa + 1) \cdot 2^l$ and $(\kappa + 2) \cdot 2^l$.
- When we stop, we do a symmetric process starting from β.

▷ Claim 21. We stop after $\mathcal{O}(\log(\beta - \alpha))$ phases, after which we will have covered the full interval $[\alpha, \alpha + \Delta]$ for $\Delta \geq (\beta - \alpha)/2$.

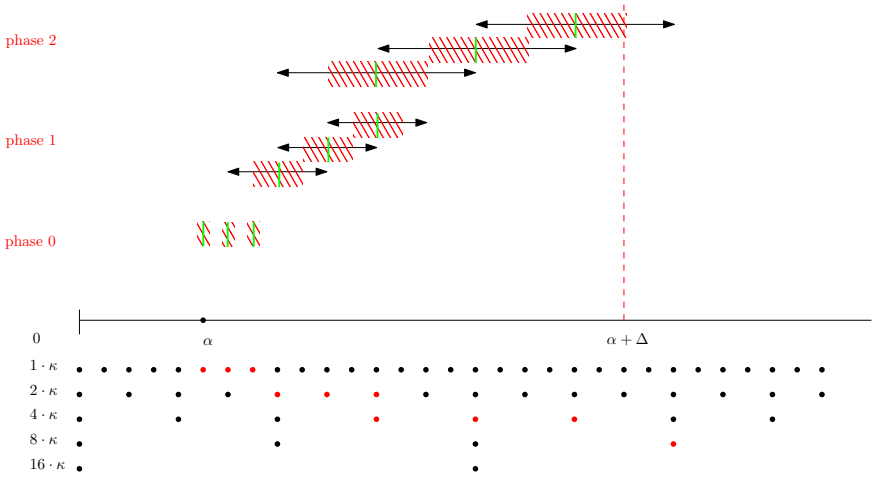


Figure 2 Illustration of the phases from the proof of Theorem 5: At the lth phase of the algorithm, we use at most three 2^l-approximate queries to cover at least $2 \cdot 2^l$ elements in $[\alpha, \beta]$ which were not covered by previous phases. The centers of the queries are shown by short vertical line segments. The covered elements for each query are shown dashed, and the uncertain elements are shown by the black horizontal arrows. The dots at the bottom show all potential centers for the approximate queries; the ones our algorithm uses are marked in red.

Proof. By the choice of the first query in each phase, the interval we cover with that query starts at some position $\kappa \cdot 2^l - 2^{l-1} \leq \alpha + \Delta$. Thus, there are no gaps. The first interval covers up until $\kappa \cdot 2^l + 2^{l-1} = (\kappa+1) \cdot 2^l - 2^{l-1} > \alpha + \Delta$. Thus, only the first query of a phase covers part of an interval that was covered in a previous phase, while the next two queries cover $2 \cdot 2^l$ elements which were not covered in a previous phase. Thus, after $\mathcal{O}(\log(\beta - \alpha))$ phases, we have covered at least until $\alpha + (\beta - \alpha)/2$. ◁

▷ Claim 22. All uncertain elements introduced during the algorithm are included in $[\alpha, \beta]$.

Proof. By the argument before, if we enter phase l and have not stopped, we have covered at least $2\sum_{j=0}^{l-1} 2^j = 2(2^l - 1) = 2^{l+1} - 2$ elements, i.e., $\Delta \geq 2^{l+1} - 2$. Further, $\Delta < (\beta - \alpha)/2$. The first query in phase l is centered at $\kappa \cdot 2^l$ with $\kappa \cdot 2^l - 2^{l-1} \leq \alpha + \Delta < \kappa \cdot 2^l + 2^{l-1}$. Thus, $\kappa \cdot 2^l \leq \alpha + \Delta + 2^{l-1}$ and $\kappa \cdot 2^l \geq \alpha + \Delta - 2^{l-1} + 1$ and the uncertain interval $(\kappa \cdot 2^l - 2^l, \kappa \cdot 2^l + 2^l)$ is contained in $[\alpha + \Delta - 2^{l-1} + 2 - 2^l,\ \alpha + \Delta + 2^{l-1} + 2^l - 1]$. Since $\Delta \geq 2^{l+1} - 2 \geq 2^l + 2^{l-1} - 2$, we have that $\alpha + \Delta - 2^{l-1} - 2^l + 2 \geq \alpha$ and thus there are no uncertain elements smaller than α. Since $\Delta + 2^l + 2^{l-1} - 1 \leq 2\Delta + 1 \leq \beta - \alpha$, we have that $\alpha + \Delta + 2^l + 2^{l-1} - 1 \leq \beta$ and thus there are no uncertain elements bigger than β.

If there are subsequent queries in phase l, then before the query we cover up to $\alpha + \Delta$ for $\Delta < (\beta - \alpha)/2$. The next query is centered at $\kappa \cdot 2^l = \alpha + \Delta + 2^{l-1}$, thus the argument as to why we do not introduce uncertain elements past β is analogous. Since $\Delta \geq 2^l$ and $\kappa \cdot 2^l > \alpha + \Delta$, the uncertain interval centered at $\kappa \cdot 2^l$ cannot include elements smaller than α. ◁

Combining Claim 21 and Claim 22, we obtain the theorem. ◀

Next, we show how we can solve Gapped Set Intersection w. Reporting, the reporting variant of the Gapped Set Intersection problem. The reduction is basically the same, but using a solution to Shifted Set Intersection w. Reporting instead of a solution to Shifted Set Intersection. Again, we use an approximate variant of the problem, now defined as follows:

Approximate Gapped Set Intersection w. Reporting
Preprocess: A collection of k sets $S_1, \ldots, S_k$ of total size $\sum_i |S_i| = N$ of integers from a universe $U = \{1, 2, \ldots, u\}$ and a *level* l with $1 \leq l \leq \log u$.
Query: Given i, j and a *center distance* $d = \kappa \cdot 2^l$, for some positive integer κ, output a set P of pairs (a, b) such that $a \in S_i$ and $b \in S_j$ and
- P contains all pairs (a, b), $a \in S_i$, $b \in S_j$ such that $b - a \in [d - 2^{l-1}, d + 2^{l-1}]$;
- P contains no pair (a, b) such that $b - a \notin (d - 2^l, d + 2^l)$.

▶ **Corollary 23.** *For every $0 \leq \delta \leq 1$, there is a data structure for Approximate Gapped Set Intersection w. Reporting with $\tilde{\mathcal{O}}(N^{2-\delta/3})$ space and $\tilde{\mathcal{O}}(N^\delta \cdot |P|)$ query time.*

Proof. The reduction is essentially the same as Lemma 19, just that instead of the Shifted Set Intersection data structure, we use the Shifted Set Intersection w. Reporting data structure from Theorem 3. That is, we construct a Shifted Set Intersection w. Reporting data structure for sets $S'_i = \{\lfloor a/2^{l-1} \rfloor, a \in S_i\}$. Additionally, for any $a' \in S'_i$, we store a list $L_{a'}$ of all values $a \in S_i$ such that $\lfloor a/2^{l-1} \rfloor = a'$. To answer a query $(i, j, d = \kappa \cdot 2^l)$, we query the Shifted Set Intersection w. Reporting data structure for $(i, j, 2\kappa - 1), (i, j, 2\kappa)$ and $(i, j, 2\kappa + 1)$. Whenever one of the queries returns a pair (a', b'), we return *all* pairs (a, b) for $a \in L_{a'}$ and $b \in L_{b'}$. By the same arguments as in the proof of Lemma 19, we never return a pair with $b - a \notin (d - 2^l, d + 2^l)$, and we return all pairs $b - a \in [d - 2^{l-1}, d + 2^{l-1}]$. ◀

▶ **Theorem 6.** *For every $0 \leq \delta \leq 1$, there is a data structure for Gapped Set Intersection w. Reporting with $\tilde{\mathcal{O}}(N^{2-\delta/3})$ space and $\tilde{\mathcal{O}}(N^\delta \cdot (\mathrm{occ} + 1))$ query time, where* occ *is the size of the output.*

Proof. We use the same reduction as in Theorem 5, only using the data structures for Approximate Gapped Set Intersection w. Reporting from Corollary 23 instead of the data structures for Approximate Gapped Set Intersection. We perform exactly the same queries. By the same arguments as in the proof of Theorem 5, we find all pairs (a, b) with $a \in S_i$ and $b \in S_j$ and $b - a \in [\alpha, \beta]$ and only those. However, we might report the same pair many times, so we need to argue about the total size of the output. Note that any single query to Approximate Gapped Set Intersection w. Reporting reports any pair at most once, thus, any pair is reported $\mathcal{O}(\log u)$ times and the total size of the output is $\mathcal{O}(\mathrm{occ} \cdot \log u)$. To avoid multiple outputs we can collect all pairs, sort them, and delete duplicates, before outputting. ◀

5 Gapped String Indexing

Let us recall some basic definitions and notation on strings. An *alphabet* Σ is a finite set of elements called *letters*. A *string* $S = S[1..n]$ is a sequence of letters over some alphabet Σ; we denote the length of S by $|S| = n$. The fragment $S[i..j]$ of S is an *occurrence* of the underlying *substring* $P = S[i] \ldots S[j]$. We also write that P occurs at *position* i in S when $P = S[i] \ldots S[j]$. A *prefix* of S is a fragment of S of the form $S[1..j]$ and a *suffix* of S is a fragment of S of the form $S[i..n]$.

For a string S of length n over an ordered alphabet of size σ, the *suffix array* $\mathsf{SA}[1..n]$ stores the permutation of $\{1, \ldots, n\}$ such that $\mathsf{SA}[i]$ is the starting position of the ith lexicographically smallest suffix of S. The standard SA application is as a text index, in which S is the *text*: given any string $P[1..m]$, known as the *pattern*, the suffix array of S allows us to report all occ occurrences of P in S using only $\mathcal{O}(m \log n + \mathrm{occ})$ operations [41].

We do a binary search in SA, which results in an interval $[s, e)$ of suffixes of S having P as a prefix. Then, $\mathsf{SA}[s\,..\,e-1]$ contains the starting positions of all occurrences of P in S. The SA is often augmented with the LCP array [41] storing the length of longest common prefixes of lexicographically adjacent suffixes. In this case, reporting all occ occurrences of P in S can be done in $\mathcal{O}(m + \log n + \text{occ})$ time [41] (see [18, 22, 47] for subsequent improvements). The suffix array can be constructed in $\mathcal{O}(n)$ time for an integer alphabet of size $\sigma = n^{\mathcal{O}(1)}$ [20]. Given the suffix array of S, we can compute the LCP array of S in $\mathcal{O}(n)$ time [32].

The *suffix tree* of S, which we denote by $\mathsf{ST}(S)$, is the compacted trie of all the suffixes of S [52]. Assuming S ends with a unique terminating symbol, every suffix $S[i\,..\,n]$ of S is represented by a leaf node that we decorate with i. We refer to the set of indices stored at the leaf nodes in the subtree rooted at node v as the *leaf-list* of v, and we denote it by $LL(v)$. Each edge in $\mathsf{ST}(S)$ is labeled with a substring of S such that the path from the root to the leaf annotated with index i spells the suffix $S[i\,..\,n]$. We refer to the substring of S spelled by the path from the root to node v as the *path-label* of v and denote it by $L(v)$. Given any pattern $P[1\,..\,m]$, the suffix tree of S allows us to report all occ occurrences of P in S using only $\mathcal{O}(m \log \sigma + \text{occ})$ operations. We spell P from the root of $\mathsf{ST}(S)$ (to access edges by the first letter of their label, we use binary search) until we arrive (if possible) at the first node v such that P is a prefix of $L(v)$. Then all occ occurrences of P in S are precisely $LL(v)$. The suffix tree can be constructed in $\mathcal{O}(n)$ time for an integer alphabet of size $\sigma = n^{\mathcal{O}(1)}$ [20]. To improve the query time to $\mathcal{O}(m + \text{occ})$ we use randomization to construct a perfect hash table [23] accessing edges by the first letter of their label in $\mathcal{O}(1)$ time.

We next show how to reduce Gapped String Indexing w. Reporting to Gapped Set Intersection w. Reporting. A query for P_1, P_2 and $[\alpha, \beta]$ corresponds to a Gapped Set Intersection w. Reporting query for S_1, S_2 and $[\alpha, \beta]$, where S_1 is the set of occurrences of P_1 in S and S_2 is the set of occurrences of P_2 in S. An obvious strategy would be to preprocess all sets which correspond to leaves below a node in the suffix tree into a Gapped Set Intersection w. Reporting data structure. The issue is that the total size of these sets can be $\Omega(n^2)$. However, any such set corresponds to a consecutive interval within the suffix array. Thus, the strategy is as follows: We store the suffix tree and the suffix array for S. We cover the suffix array in dyadic intervals and preprocess the resulting subarrays into the Gapped Set Intersection w. Reporting data structure. In detail, let D be the set of dyadic intervals covering $[1, n]$. That is, D includes all intervals of the form $[1 + \kappa \cdot 2^j, (\kappa + 1) \cdot 2^j]$ for all $0 \leq \kappa \leq \lfloor \frac{n}{2^j} \rfloor - 1$ and $0 \leq j \leq \lfloor \log n \rfloor$. For any interval $[\gamma_1, \gamma_2] \in D$, we define a set containing the elements in $\mathsf{SA}[\gamma_1\,..\,\gamma_2]$. We preprocess all of these sets into the Gapped Set Intersection w. Reporting data structure. The total size of the sets in the data structure is $\mathcal{O}(n \log n)$. For a query, we find the suffix array intervals (I_1, I_2) for P_1 and P_2, respectively, in $\mathcal{O}(|P_1| + |P_2|)$ time, using standard pattern matching in the suffix tree. Now, let $\mathcal{A}$ be the collection of sets corresponding to dyadic intervals covering I_1 and $\mathcal{B}$ the collection of sets corresponding to dyadic intervals covering I_2. We can find all pairs (i, j) of occurrences of P_1 and P_2 satisfying $j - i \in [\alpha, \beta]$ by querying Gapped Set Intersection w. Reporting for $(A, B, [\alpha, \beta])$ for all $A \in \mathcal{A}$ and $B \in \mathcal{B}$.

In conclusion, we have shown the following:

▶ **Theorem 24.** *Assume there is a data structure for Shifted Set Intersection with $s(N)$ space and $t(N)$ query time, where N is the input size, and which outputs a witness pair $(a, b) \in S_i \times S_j$ satisfying $a + s = b$ for a query (i, j, s). Then there is a data structure for Gapped String Indexing w. Reporting with $\tilde{\mathcal{O}}(n + s(n))$ space and $\tilde{\mathcal{O}}(|P_1| + |P_2| + t(n) \cdot (\text{occ} + 1))$ query time, where n is the length of the input string and* occ *is the size of the output.*

The following two corollaries follow from Theorem 24 together with Theorem 6 and Lemma 14, respectively.

▶ **Corollary 25.** *For every $0 \leq \delta \leq 1$, there is a data structure for* Gapped String Indexing w. Reporting *with $\tilde{\mathcal{O}}(n^{2-\delta/3})$ space and $\tilde{\mathcal{O}}(|P_1| + |P_2| + n^{\delta} \cdot (\mathrm{occ} + 1))$ query time, where* occ *is the size of the output.*

▶ **Corollary 26.** *For every $0 \leq \delta \leq 1$, there is a data structure for* Gapped String Indexing w. Reporting *with $\tilde{\mathcal{O}}(n^{3-2\delta})$ space and $\tilde{\mathcal{O}}(|P_1| + |P_2| + n^{\delta} \cdot (\mathrm{occ} + 1))$ query time, where* occ *is the size of the output.*

Note that $n^{3-2\delta}$ is smaller than $n^{2-\delta/3}$ for $\delta > 3/5$. We have arrived at Theorem 1.

6 Jumbled Indexing

A solution to 3SUM Indexing w. Reporting implies solutions to other problems. As proof of concept, we show here the implications to Jumbled Indexing w. Reporting: preprocess a string S of length n over an alphabet Σ into a compact data structure that facilitates efficient search for substrings of S whose characters have a specified histogram. For instance, $P = acaacabd$ has the histogram $h(P) = (4, 1, 2, 1)$: $h(P)[a] = 4$ because a occurs 4 times in P, $h(P)[b] = 1$, etc. The existence variant, Jumbled Indexing, answers the question of whether such a substring occurs in S or not. We show the following result:

▶ **Corollary 8.** *Given a string S of length n over an alphabet of size $\sigma = \mathcal{O}(1)$, for every $0 \leq \delta \leq 1$, there is a data structure for* Jumbled Indexing w. Reporting *with $\tilde{\mathcal{O}}(n^{2-\delta})$ space and $\tilde{\mathcal{O}}(n^{3\delta} \cdot (\mathrm{occ} + 1))$ query time, where* occ *is the size of the output. The bounds are $\tilde{\mathcal{O}}(n^{2-\delta})$ space and $\tilde{\mathcal{O}}(n^{3\delta})$ query time for the existence variant.*

Before proving the above statement, we will briefly overview the related literature.

Related work. Previous work on Jumbled Indexing has achieved the bounds of $\mathcal{O}(n^{2-\delta})$ space and $\mathcal{O}(m^{(2\sigma-1)\delta})$ query time [34], where m is the norm of the pattern and $\sigma = |\Sigma|$. Later, a data structure with $\tilde{\mathcal{O}}(n^{2-\delta})$ space and $\tilde{\mathcal{O}}(n^{\delta(\sigma+1)/2})$ query time was presented [14].

For the special case of a binary alphabet, more efficient algorithms exist; namely, $\mathcal{O}(n)$ space and $\mathcal{O}(1)$ query time [16]. The data structure in [16] has a preprocessing time of $\mathcal{O}(n^2)$, which can be improved to $\tilde{\mathcal{O}}(n^{1.5})$ using the connection to min-plus-convolution [14, 15].

Jumbled Indexing has also seen interest from the lower-bound side, where [1] shows that if a data structure can report all the occ matches to a histogram query in $\mathcal{O}(n^{0.5-o(1)} + \mathrm{occ})$ time, then it needs to use $\Omega(n^{2-o(1)})$ space, even for the special case of a binary alphabet. In fine-grained complexity, the problem has also received attention; [2] shows that under a 3SUM hardness assumption, for alphabets of size $\omega(1)$, we cannot get $\mathcal{O}(n^{2-\epsilon})$ preprocessing time and $\mathcal{O}(n^{1-\delta})$ query time for any $\epsilon, \delta > 0$. Furthermore, under the now refuted Strong 3SUM Indexing conjecture, [26] gives conditional lower bounds, that are contradicted by our results. In particular, their conditional lower bound (conditioned on the now refuted Strong 3SUM Indexing conjecture) argues against a $\mathcal{O}(n^{2-\frac{2(1-\alpha)}{\sigma-1-\alpha}-\Omega(1)})$ space and $\mathcal{O}(n^{1-\frac{1+\alpha(\sigma-3)}{\sigma-1-\alpha}-\Omega(1)})$ query time solution for any $0 \leq \alpha \leq 1$. Setting $\alpha = 0$ and $\sigma = 9$, this would imply that there does not exist a $\mathcal{O}(n^{2-\frac{1}{4}-\Omega(1)})$ space and $\mathcal{O}(n^{1-\frac{1}{8}-\Omega(1)})$ query time solution. However, setting $\delta = 1/4+\epsilon$ for $\epsilon = 1/48$, we obtain an $\tilde{\mathcal{O}}(n^{2-\frac{1}{4}-\epsilon})$ space and $\tilde{\mathcal{O}}(n^{\frac{3}{4}+3\epsilon}) = \tilde{\mathcal{O}}(n^{\frac{6}{8}+\frac{1}{16}}) = \tilde{\mathcal{O}}(n^{1-\frac{1}{8}-\frac{1}{16}})$ query time, a contradiction. We now return back to the proof of Corollary 8:

Proof of Corollary 8. Let us first remind the reader of the well-known reduction to d-dimensional 3SUM Indexing, as introduced by Chan and Lewenstein in [14]. The d-dimensional 3SUM Indexing problem is defined as follows: Preprocess two sets A and B of N vectors from $[0, \ldots, u-1]^d$ each such that we can efficiently answer queries of the following form: for some $c \in [0, \ldots, u-1]^d$, decide if there exists $a \in A$ and $b \in B$ such that $c = a + b$.

We reduce Jumbled Indexing to σ-dimensional 3SUM Indexing as follows. Let $\sigma = |\Sigma|$. Let A be the set of all histograms of prefixes of the input string S, and similarly, let B be the set of all histograms of suffixes of S. We thus have that the cardinality of these two sets is the length of the string S, i.e., $|A| = |B| = |S| = n$, and may thus build a σ-dimensional 3SUM Indexing data structure for A and B. Additionally, we store the histogram $h(S)$ of S. For a query histogram P, compute $c = h(S) - P$, and query the σ-dimensional 3SUM Indexing data structure for c. Any match a, b returned by the data structure corresponds to a histogram of the complement of some substring p whose histogram is P: a corresponds to the prefix of everything before p, and b corresponds to the suffix of everything after p.

Now, we note that there is a reduction from d-dimensional 3SUM Indexing over a universe of size u to 3SUM Indexing over a universe of size $\mathcal{O}(u^d)$. Our reduction works by doing the following transformation on sets A and B: define $A' = \{a_1 + a_2 \cdot u + a_3 \cdot u^2 + \ldots a_d \cdot u^{d-1} : a \in A\}$; and define B' in the same way. For a query c, define the query $c' = c_1 + c_2 \cdot u + c_3 \cdot u^2 \ldots c_d \cdot u^{d-1}$. Thus, by spacing out using u-factors, we ensure that any match to a query c to the sets A and B corresponds exactly to a match to the corresponding query c' to the sets A' and B'. With this reduction, the set size remains unchanged, that is, $|A| = |A'|$ and $|B| = |B'|$, however, we are now indexing with a universe of size $u' = O(u^d)$.

Thus, finally, for Jumbled Indexing, let n be the length of the string S. Our reductions would result in a universe of size $u' = n^\sigma$. For constant σ this value u' is polynomial in n. Therefore, by Corollary 4, we obtain a data structure for Jumbled Indexing over constant sized alphabets with $\tilde{\mathcal{O}}(n^{2-\delta})$ space and $\tilde{\mathcal{O}}(n^{3\delta} \cdot (\mathrm{occ} + 1))$ query time. ◀

7 Final Remarks

Our solutions show new and interesting relations between Gapped String Indexing, Shifted Set Intersection, and 3SUM Indexing; in particular, we contribute new trade-offs for 3SUM Indexing w. Reporting. Chan showed that the 3SUM problem has direct applications in computational geometry [13]; it would be interesting to see if our data structure yields improved bounds for the data structure versions of these geometric problems.

References

1 Peyman Afshani, Ingo van Duijn, Rasmus Killmann, and Jesper Sindahl Nielsen. A lower bound for jumbled indexing. In *Proceedings of the 2020 ACM-SIAM Symposium on Discrete Algorithms, SODA 2020, Salt Lake City, UT, USA, January 5-8, 2020*, pages 592–606, 2020. doi:10.1137/1.9781611975994.36.

2 Amihood Amir, Timothy M. Chan, Moshe Lewenstein, and Noa Lewenstein. On hardness of jumbled indexing. In *Automata, Languages, and Programming - 41st International Colloquium, ICALP 2014, Copenhagen, Denmark, July 8-11, 2014, Proceedings, Part I*, pages 114–125, 2014. doi:10.1007/978-3-662-43948-7_10.

3 Boris Aronov, Jean Cardinal, Justin Dallant, and John Iacono. A general technique for searching in implicit sets via function inversion. In *2024 Symposium on Simplicity in Algorithms (SOSA)*, pages 215–223, 2024. doi:10.1137/1.9781611977936.20.

4 Johannes Bader, Simon Gog, and Matthias Petri. Practical variable length gap pattern matching. In *Experimental Algorithms - 15th International Symposium, SEA 2016, St. Petersburg, Russia, June 5-8, 2016, Proceedings*, pages 1–16, 2016. doi:10.1007/978-3-319-38851-9_1.

5 Philip Bille and Inge Li Gørtz. Substring range reporting. *Algorithmica*, 69(2):384–396, 2014. doi:10.1007/S00453-012-9733-4.

6 Philip Bille, Inge Li Gørtz, Moshe Lewenstein, Solon P. Pissis, Eva Rotenberg, and Teresa Anna Steiner. Gapped string indexing in subquadratic space and sublinear query time, 2022. doi:10.48550/ARXIV.2211.16860.

7 Philip Bille, Inge Li Gørtz, Max Rishøj Pedersen, and Teresa Anna Steiner. Gapped indexing for consecutive occurrences. *Algorithmica*, 85(4):879–901, 2023. doi:10.1007/S00453-022-01051-6.

8 Philip Bille, Inge Li Gørtz, Hjalte Wedel Vildhøj, and Søren Vind. String indexing for patterns with wildcards. *Theory Comput. Syst.*, 55(1):41–60, 2014. doi:10.1007/S00224-013-9498-4.

9 Philip Bille, Inge Li Gørtz, Hjalte Wedel Vildhøj, and David Kofoed Wind. String matching with variable length gaps. *Theor. Comput. Sci.*, 443:25–34, 2012. doi:10.1016/J.TCS.2012.03.029.

10 Philip Bille and Mikkel Thorup. Regular expression matching with multi-strings and intervals. In *Proceedings of the Twenty-First Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2010, Austin, Texas, USA, January 17-19, 2010*, pages 1297–1308, 2010. doi:10.1137/1.9781611973075.104.

11 Philipp Bucher and Amos Bairoch. 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, August 14-17, 1994, Stanford University, Stanford, California, USA*, pages 53–61, 1994. URL: http://www.aaai.org/Library/ISMB/1994/ismb94-007.php.

12 Manuel Cáceres, Simon J. Puglisi, and Bella Zhukova. Fast indexes for gapped pattern matching. In *SOFSEM 2020: Theory and Practice of Computer Science - 46th International Conference on Current Trends in Theory and Practice of Informatics, SOFSEM 2020, Limassol, Cyprus, January 20-24, 2020, Proceedings*, pages 493–504, 2020. doi:10.1007/978-3-030-38919-2_40.

13 Timothy M. Chan. More logarithmic-factor speedups for 3SUM, (median, +)-convolution, and some geometric 3SUM-hard problems. *ACM Trans. Algorithms*, 16(1):7:1–7:23, 2020. doi:10.1145/3363541.

14 Timothy M. Chan and Moshe Lewenstein. Clustered integer 3sum via additive combinatorics. In *Proceedings of the Forty-Seventh Annual ACM on Symposium on Theory of Computing, STOC 2015, Portland, OR, USA, June 14-17, 2015*, pages 31–40, 2015. doi:10.1145/2746539.2746568.

15 Shucheng Chi, Ran Duan, Tianle Xie, and Tianyi Zhang. Faster min-plus product for monotone instances. In *STOC '22: 54th Annual ACM SIGACT Symposium on Theory of Computing, Rome, Italy, June 20 - 24, 2022*, pages 1529–1542, 2022. doi:10.1145/3519935.3520057.

16 Ferdinando Cicalese, Gabriele Fici, and Zsuzsanna Lipták. Searching for jumbled patterns in strings. In *Proceedings of the Prague Stringology Conference 2009, Prague, Czech Republic, August 31 - September 2, 2009*, pages 105–117, 2009. URL: http://www.stringology.org/event/2009/p10.html.

17 Richard Cole, Lee-Ad Gottlieb, and Moshe Lewenstein. Dictionary matching and indexing with errors and don't cares. In *Proceedings of the 36th Annual ACM Symposium on Theory of Computing, Chicago, IL, USA, June 13-16, 2004*, pages 91–100, 2004. doi:10.1145/1007352.1007374.

18 Richard Cole, Tsvi Kopelowitz, and Moshe Lewenstein. Suffix trays and suffix trists: Structures for faster text indexing. *Algorithmica*, 72(2):450–466, 2015. doi:10.1007/S00453-013-9860-6.

19 Maxime Crochemore, Christophe Hancart, and Thierry Lecroq. *Algorithms on strings*. Cambridge University Press, 2007.

20 Martin Farach. Optimal suffix tree construction with large alphabets. In *38th Annual Symposium on Foundations of Computer Science, FOCS '97, Miami Beach, Florida, USA, October 19-22, 1997*, pages 137–143, 1997. doi:10.1109/SFCS.1997.646102.

21 Amos Fiat and Moni Naor. Rigorous time/space trade-offs for inverting functions. *SIAM J. Comput.*, 29(3):790–803, 1999. doi:10.1137/S0097539795280512.

22 Johannes Fischer and Pawel Gawrychowski. Alphabet-dependent string searching with wexponential search trees. In *Combinatorial Pattern Matching - 26th Annual Symposium, CPM 2015, Ischia Island, Italy, June 29 - July 1, 2015, Proceedings*, pages 160–171, 2015. doi:10.1007/978-3-319-19929-0_14.

23 Michael L. Fredman, János Komlós, and Endre Szemerédi. Storing a sparse table with 0(1) worst case access time. *J. ACM*, 31(3):538–544, 1984. doi:10.1145/828.1884.

24 Kimmo Fredriksson and Szymon Grabowski. Efficient algorithms for pattern matching with general gaps, character classes, and transposition invariance. *Inf. Retr.*, 11(4):335–357, 2008. doi:10.1007/S10791-008-9054-Z.

25 Pawel Gawrychowski, Moshe Lewenstein, and Patrick K. Nicholson. Weighted ancestors in suffix trees. In *Algorithms - ESA 2014 - 22th Annual European Symposium, Wroclaw, Poland, September 8-10, 2014. Proceedings*, pages 455–466, 2014. doi:10.1007/978-3-662-44777-2_38.

26 Isaac Goldstein, Tsvi Kopelowitz, Moshe Lewenstein, and Ely Porat. Conditional lower bounds for space/time tradeoffs. In *Algorithms and Data Structures - 15th International Symposium, WADS 2017, St. John's, NL, Canada, July 31 - August 2, 2017, Proceedings*, pages 421–436, 2017. doi:10.1007/978-3-319-62127-2_36.

27 Alexander Golovnev, Siyao Guo, Thibaut Horel, Sunoo Park, and Vinod Vaikuntanathan. 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, June 22-26, 2020*, pages 294–307, 2020. doi:10.1145/3357713.3384342.

28 Dan Gusfield. *Algorithms on Strings, Trees, and Sequences - Computer Science and Computational Biology*. Cambridge University Press, 1997. doi:10.1017/cbo9780511574931.

29 Tuukka Haapasalo, Panu Silvasti, Seppo Sippu, and Eljas Soisalon-Soininen. Online dictionary matching with variable-length gaps. In *Experimental Algorithms - 10th International Symposium, SEA 2011, Kolimpari, Chania, Crete, Greece, May 5-7, 2011. Proceedings*, pages 76–87, 2011. doi:10.1007/978-3-642-20662-7_7.

30 Kay Hofmann, Philipp Bucher, Laurent Falquet, and Amos Bairoch. The PROSITE database, its status in 1999. *Nucleic Acids Res.*, 27(1):215–219, 1999. doi:10.1093/NAR/27.1.215.

31 Costas S. Iliopoulos and M. Sohel Rahman. Indexing factors with gaps. *Algorithmica*, 55(1):60–70, 2009. doi:10.1007/S00453-007-9141-3.

32 Toru Kasai, Gunho Lee, Hiroki Arimura, Setsuo Arikawa, and Kunsoo Park. Linear-time longest-common-prefix computation in suffix arrays and its applications. In *Combinatorial Pattern Matching, 12th Annual Symposium, CPM 2001 Jerusalem, Israel, July 1-4, 2001 Proceedings*, pages 181–192, 2001. doi:10.1007/3-540-48194-X_17.

33 Donald E. Knuth, James H. Morris Jr., and Vaughan R. Pratt. Fast pattern matching in strings. *SIAM J. Comput.*, 6(2):323–350, 1977. doi:10.1137/0206024.

34 Tomasz Kociumaka, Jakub Radoszewski, and Wojciech Rytter. Efficient indexes for jumbled pattern matching with constant-sized alphabet. *Algorithmica*, 77(4):1194–1215, 2017. doi:10.1007/S00453-016-0140-0.

35 Tsvi Kopelowitz and Robert Krauthgamer. Color-distance oracles and snippets. In *27th Annual Symposium on Combinatorial Pattern Matching, CPM 2016, June 27-29, 2016, Tel Aviv, Israel*, pages 24:1–24:10, 2016. doi:10.4230/LIPICS.CPM.2016.24.

36 Tsvi Kopelowitz and Ely Porat. The strong 3SUM-INDEXING conjecture is false. *CoRR*, abs/1907.11206, 2019. arXiv:1907.11206.

37 Paul R. Kroeger. *Analyzing Grammar: An Introduction*. Cambridge University Press, Cambridge, 2005.

38 Moshe Lewenstein. Indexing with gaps. In *String Processing and Information Retrieval, 18th International Symposium, SPIRE 2011, Pisa, Italy, October 17-21, 2011. Proceedings*, pages 135–143, 2011. doi:10.1007/978-3-642-24583-1_14.

39 Moshe Lewenstein, J. Ian Munro, Venkatesh Raman, and Sharma V. Thankachan. Less space: Indexing for queries with wildcards. *Theor. Comput. Sci.*, 557:120–127, 2014. doi:10.1016/j.tcs.2014.09.003.

40 Moshe Lewenstein, Yakov Nekrich, and Jeffrey Scott Vitter. Space-efficient string indexing for wildcard pattern matching. In *31st International Symposium on Theoretical Aspects of Computer Science (STACS 2014), STACS 2014, March 5-8, 2014, Lyon, France*, pages 506–517, 2014. doi:10.4230/LIPICS.STACS.2014.506.

41 Udi Manber and Eugene W. Myers. Suffix arrays: A new method for on-line string searches. *SIAM J. Comput.*, 22(5):935–948, 1993. doi:10.1137/0222058.

42 Christopher D. Manning and Hinrich Schütze. *Foundations of Statistical Natural Language Processing*. MIT Press, Cambridge, MA, 1999.

43 Gerhard Mehldau and Gene Myers. A system for pattern matching applications on biosequences. *Bioinformatics*, 9(3):299–314, 1993. doi:10.1093/bioinformatics/9.3.299.

44 Gary Miner, Dursun Delen, John Elder, Andrew Fast, Thomas Hill, and Robert A. Nisbet. *Practical Text Mining and Statistical Analysis for Non-structured Text Data Applications*. Academic Press, Boston, 2012.

45 Michele Morgante, Alberto Policriti, Nicola Vitacolonna, and Andrea Zuccolo. Structured motifs search. *J. Comput. Biol.*, 12(8):1065–1082, 2005. doi:10.1089/cmb.2005.12.1065.

46 Eugene W. Myers. Approximate matching of network expressions with spacers. *J. Comput. Biol.*, 3(1):33–51, 1996. doi:10.1089/cmb.1996.3.33.

47 Gonzalo Navarro and Yakov Nekrich. Time-optimal top-k document retrieval. *SIAM J. Comput.*, 46(1):80–113, 2017. doi:10.1137/140998949.

48 Gonzalo Navarro and Mathieu Raffinot. Fast and simple character classes and bounded gaps pattern matching, with applications to protein searching. *J. Comput. Biol.*, 10(6):903–923, 2003. doi:10.1089/106652703322756140.

49 Pierre Peterlongo, Julien Allali, and Marie-France Sagot. Indexing gapped-factors using a tree. *Int. J. Found. Comput. Sci.*, 19(1):71–87, 2008. doi:10.1142/S0129054108005541.

50 Solon P. Pissis. MoTeX-II: structured MoTif eXtraction from large-scale datasets. *BMC Bioinform.*, 15:235, 2014. doi:10.1186/1471-2105-15-235.

51 Ken Thompson. Regular expression search algorithm. *Commun. ACM*, 11(6):419–422, 1968. doi:10.1145/363347.363387.

52 Peter Weiner. Linear pattern matching algorithms. In *14th Annual Symposium on Switching and Automata Theory, Iowa City, Iowa, USA, October 15-17, 1973*, pages 1–11, 1973. doi:10.1109/SWAT.1973.13.

53 Dan E. Willard. Log-logarithmic worst-case range queries are possible in space $\Theta(N)$. *Inf. Process. Lett.*, 17(2):81–84, 1983. doi:10.1016/0020-0190(83)90075-3.

54 Jens Willkomm, Martin Schäler, and Klemens Böhm. Accurate cardinality estimation of co-occurring words using suffix trees. In *Database Systems for Advanced Applications - 26th International Conference, DASFAA 2021, Taipei, Taiwan, April 11-14, 2021, Proceedings, Part II*, pages 721–737, 2021. doi:10.1007/978-3-030-73197-7_50.

A A Linear-Space Solution for Gapped Indexing

The linear-space solution uses a standard linear-time pattern matching algorithm (e.g., the Knuth-Morris-Pratt algorithm [33]), which, given two strings S and P, it outputs all positions in S where P occurs, in sorted order. We run this algorithm two times: on P_1 and S; and on P_2 and S. The two sorted lists of positions are scanned from left to right using a two-finger approach: We start with the first position i_1 where P_1 occurs in S, and scan the list of P_2 occurrences until we find the first one that is at least $i_1 + \alpha$. Then we scan and output all

pairs until we arrive at a position of P_2 which is larger than $i_1 + \beta$. Then we advance to the next position i_2 of P_1, and from the last position we considered in the list for P_2, we scan first backward and then forward to output all positions in $[i_2 + \alpha, i_2 + \beta]$. Every position we scan is charged to an occurrence we output, thus we achieve a running time of $\mathcal{O}(n + \text{occ})$.

B A Near-Quadratic-Space Solution for Gapped Indexing

Let us start with a data structure of $\mathcal{O}(n^3)$ space and $\mathcal{O}(|P_1| + |P_2| + \text{occ})$ query time: We store, for each pair (v, w) of nodes in the suffix tree of S, the set of distances that appear between the set of occurrences defined by v and w. To answer a query, we search for P_1 and P_2 in the suffix tree to locate the corresponding pair of nodes and then report the distances within the gap range $[\alpha, \beta]$. Since the number of distinct distances is at most $n - 1$ this leads to a solution using $\mathcal{O}(n^3)$ space and $\mathcal{O}(|P_1| + |P_2| + \text{occ})$ query time, where occ is the size of the output. A more sophisticated approach can reduce the space to $\tilde{\mathcal{O}}(n^2)$ at the cost of increasing the query time by polylogarithmic factors. Namely, the idea is to consider subarrays of the suffix array of S [41] corresponding to the dyadic intervals on $[1, n]$, and store, for every possible distance d, all pairs of subarrays (A, B) containing elements at distance d. Further, we store a list of all (a, b), $a \in A$ and $b \in B$ such that $b - a = d$. Since the total size of the subarrays is $\mathcal{O}(n \log n)$, the total number of pairs of elements is $\mathcal{O}(n^2 \log^2 n)$, thus this solution can be stored using $\tilde{\mathcal{O}}(n^2)$ space. We also store the suffix tree of S, which takes $\Theta(n)$ extra space. To answer a query, we search for P_1 and P_2 in the suffix tree to find their suffix array intervals: their starting positions in S sorted lexicographically with respect to the corresponding suffixes. We cover each interval in dyadic intervals and use the precomputed solution for any pair consisting of a subarray in the cover of P_1's interval and a subarray in the cover of P_2's interval. This results in query time $\mathcal{O}(|P_1| + |P_2| + \log^2 n \cdot (\text{occ} + 1))$.

C Smallest Shift

When studying the problem of deciding whether a given shift incurs an intersection between sets, a natural next question is the optimization (minimization) variant of the problem: *what is the smallest shift that yields an intersection?* We formally define this problem next.

Smallest Shift
Preprocess: A collection of k sets $S_1, \ldots, S_k$ of total size $\sum_i |S_i| = N$ of integers from a universe $U = \{1, 2, \ldots, u\}$.
Query: Given i, j, output the smallest s such that there exists $a \in S_i$ and $b \in S_j$ with $a + s = b$.

Note that the problem is symmetric in i and j in the sense that finding the smallest positive shift transferring S_i to intersect S_j is equivalent to finding the largest negative shift transferring S_j to intersect S_i. Note also that by constructing a predecessor/successor data structure [53] at preprocessing time, we may answer the query in $\tilde{\mathcal{O}}(|S_i|)$ time: simply perform a successor (or predecessor) query in S_j for each element $a \in S_i$, and report the smallest element-successor distance.

Thus, we are safely able to handle Smallest Shift queries in $\mathcal{O}(\sqrt{N} \log \log N)$ time in all cases where either S_i or, because of symmetry, S_j, is of size at most $\sqrt{N}$. Remaining is the case where both sets are of size larger than $\sqrt{N}$. Here, however, we note that at most $\mathcal{O}(\sqrt{N})$ such sets can exist, so we can tabulate all answers in $\mathcal{O}(\sqrt{N} \cdot \sqrt{N}) = O(N)$ space.

In other words, we have the following solution:

Preprocessing. Store for each i whether $|S_i| \leq \sqrt{N}$. Let l denote the number of sets $L_1, \ldots, L_l$ of size larger than $\sqrt{N}$. For each set S_i with $|S_i| > \sqrt{N}$ store its index in the list of large sets $L_1, \ldots, L_l$.

- For all i, with $|S_i| \leq \sqrt{N}$, compute and store predecessor $\text{pre}(S_i)$ and successor $\text{suc}(S_i)$ data structures for the set S_i.
- Store an $l \times l$ table in which the (i,j)th entry stores the smallest shift s such that there exists $a \in L_i$ and $b \in L_j$ with $a + s = b$.

Query i, j.

- If $|S_i| \leq \sqrt{N}$, perform $|S_i|$ successor queries in $\text{suc}(S_j)$, and return the smallest difference.
- If $|S_j| \leq \sqrt{N}$, similarly, perform $|S_i|$ predecessor queries in $\text{pre}(S_i)$.
- If $|S_i| > \sqrt{N}$ and $|S_j| > \sqrt{N}$, use the precomputed information to output the smallest shift.

▶ **Proposition 27.** *The above data structure for* Smallest Shift *uses $\mathcal{O}(N)$ space and has $\tilde{\mathcal{O}}(\sqrt{N})$ query time. Moreover, the data structure can be constructed in $\mathcal{O}(N\sqrt{N})$ time.*

Proof. We construct predecessor and successor queries for all sets in $\sum_i |S_i| \log\log |S_i| = O(N \log\log N)$ total time and $\sum_i |S_i| = O(N)$ space, with $\mathcal{O}(\log\log N)$ query time [53], surmounting to an $\mathcal{O}(\sqrt{N}\log\log N)$ query time when either S_i or S_j is smaller than $\sqrt{N}$.

For constructing the $l \times l$ table, we note that $l \leq \sqrt{N}$. Since the (i,j)th entry of the table can be computed in time $\mathcal{O}(|L_i| + |L_j|)$ by a merge-like traversal of the sorted respective sets, the total preprocessing time becomes:

$$\sum_i \sum_{j \neq i} (|L_i| + |L_j|) \leq \sum_i (\sqrt{N}|L_i| + \sum_j |L_j|) \leq \sum_i (\sqrt{N}|L_i| + N)$$
$$= \sqrt{N} \sum_i |L_i| + \sum_i N = \mathcal{O}(\sqrt{N}N).$$

Here, we are using that $\sum_i |S_i| = N$ and that there are $l \leq \sqrt{N}$ large sets. This $\mathcal{O}(N\sqrt{N})$ term dominates the $\sum_i \tilde{\mathcal{O}}(|S_i|)$ terms needed to sort the sets and construct predecessor/successor data structures for each of them. ◀