Geometric Burrows-Wheeler Transform: Linking Range Searching and Text Indexing[*]

Yu-Feng Chien Wing-Kai Hon
National Tsing Hua University, Taiwan
{cyf,wkhon}@cs.nthu.edu.tw

Rahul Shah
Louisiana State University, LA, USA
rahul@csc.lsu.edu

Jeffrey Scott Vitter
Purdue University, IN, USA
jsv@cs.purdue.edu

Abstract

We introduce a new variant of the popular Burrows-Wheeler transform (BWT) called Geometric Burrows-Wheeler Transform (GBWT). Unlike BWT, which merely permutes the text, GBWT converts the text into a set of points in 2-dimensional geometry. Using this transform, we can answer to many open questions in compressed text indexing: (1) Can compressed data structures be designed in external memory with similar performance as the uncompressed counterparts? (2) Can compressed data structures be designed for position restricted pattern matching [16]? We also introduce a reverse transform, called `Points2Text`*, which converts a set of points into text. This transform allows us to derive the* first *known lower bounds in compressed text indexing. We show strong equivalence between data structural problems in geometric range searching and text pattern matching. This provides a way to derive new results in compressed text indexing by translating the results from range searching.*

1 Introduction

Given a text T of length n over an alphabet set Σ, *is there a data structure which takes $O(n \log \Sigma)$ bits of space and can answer pattern matching queries efficiently over T?*—is a central question in the field of succinct data structures and text indexing. This was answered positively by Grossi and Vitter [11] and Ferragina and Manzini [8]; subsequently a new field of compressed text indexing was established (see [14] for an excellent survey). This was a positive improvement over suffix trees and suffix arrays which used $O(n \log n)$ space. A central technique in this field has been to use the Burrows-Wheeler Transform (BWT) [4].

Although many positive results are known, many questions still remain unanswered. What is the best possible space versus query complexity; are there any fundamental lower bounds? What other more complex pattern searching functions

[*] This work is supported in part by Taiwan NSC Grant 96-2221-E-007-082-MY3 (W. Hon) and US NSF Grant CCF–0621457 (R. Shah and J. S. Vitter).

can be supported in the compressed space? Can one design such indexes in external memory I/O model? It seemed difficult to get insight into these problems using current BWT based approaches.

We take a fresh look at the problem of compressed text indexing and develop an alternative approach. Using this approach, we answer many of these questions, either positively or negatively. Similar to the Burrows-Wheeler transform (BWT), which transforms a text into another text, we define the Geometric Burrows-Wheeler Transform (GBWT), which transforms a text into a set of points. It is equivalent to taking Burrows-Wheeler transform on *blocked* text and reversing the characters. Unlike BWT, which needs the text characters to be in a particular order, GBWT can maintain position information (for each character) explicitly within $O(n \log |\Sigma|)$ bits and hence it is more amenable for other models like external memory. Since the BWT permutes the text, the contiguous characters in the text can end up being far apart in the permutation. This is the key bottleneck for pattern matching in external memory which relies on the locality between the contiguous text characters. In traditional BWT based approach, every character match of a pattern could possibly come from a different disk block. Thus, in the external memory model, it was never possible to achieve an additive term like $|P|/B$ in query I/Os, where B is a memory block size (measured in words). For our GBWT, since we explicitly have position information, the $|P|/B$ additive term can be achieved.

We also define a reverse transform called `Points2Text`, which transforms a set of points into text. Each individual point is converted into a string of characters and these strings are concatenated. This allows many known lower bounds known for the geometric problems to be applicable to compressed text searching problem. Both GBWT and `Points2Text` preserve space up to a constant factor. These transformations allow results in orthogonal range searching and compressed text indexing to be interchangeably used for each other. Since orthogonal range searching is a very extensively studied field [1], many results (lower bounds and upper bounds) can now be translated to the field of compressed text indexing.

Problems. We start with describing the key problems/queries we consider. For the pattern matching query Q_{match}, the input is a pattern P, and the query returns the set (or the cardinality of the set) $Q_{match}(T) = \{i \mid T[i..(i + |P| - 1)] = P\}$. An index should support these queries in $O(|P| + \text{polylog}(n) + |Q_{match}(T)|)$ time. The problem of designing the data structure for this taking only $O(n \log |\Sigma|)$ bits is called $\mathcal{CSA}$ (compressed suffix array) problem.

An extension to pattern matching is the problem of position-restricted pattern matching [16]. This can be used as a building block for many other complex text retrieval queries [12]. Here, the text T is given and the input of a query Q_{pr_match} consists of a pattern P along with positions i and j. The query returns the set (or the cardinality of the set) $Q_{pr\_match}(T) = Q_{match}(T) \cap [i, j]$.

In orthogonal range searching, we are given a set of n points by their x and y coordinates: $S = \{(x_1, y_1), (x_2, y_2), .., (x_n, y_n)\}$. The query Q_{range} specifies a rectangle $(x_\ell, x_r, y_\ell, y_r)$. The answer to query is given by $Q_{range}(S) = \{(x_i, y_i) \in S \mid x_\ell \leq x_i \leq$

$x_r,\ y_\ell \leq y_i \leq y_r\}$. Two specific versions of this query have been considered: counting and reporting. We shall also consider similar queries in dimension 3. We call the problems of designing data structures on S for efficient orthogonal range queries the $\mathcal{RS2D}$ problem for the 2-D case and the $\mathcal{RS3D}$ problem for the 3-D case.

Our Results. Based on our transforms, we show equivalence between the complexities of Q_{match} and Q_{range} in 2-dimensions, and between the complexities of Q_{pr_match} and Q_{range} in 3-dimensions. We design data structures based on this principle. Following is the summary of our results:

1. We propose two transforms GBWT and `Points2Text`; these transforms are simple, quickly computable, invertible, and (asymptotically) space-preserving. With these, we show that text pattern matching and orthogonal range searching are closely related.
2. We show that using GBWT one can achieve an $O(n \log |\Sigma|)$-bit text index answering queries in $O(|P| + (\log_\Sigma n)(\log n) + occ \log n)$ time. This gives an alternative result to the current internal memory text index.
3. We show that lower bounds for orthogonal range searching in the pointer machine (and other models) can be applied to text indexing, thus giving the first lower bound results.
4. We develop the first external memory data structure taking $O(n \log |\Sigma|)$ bits that can answer pattern matching queries in $O(|P|/B + (\log_{|\Sigma|} n)(\log_B n) + occ \log_B n)$ I/Os.
5. We develop a data structure taking $O(n \log |\Sigma|)$ bits that can answer position-restricted pattern matching queries in $O(|P| + \text{polylog}(n) + occ)$ time if $P > (\log^{2+\epsilon} n)/(\log |\Sigma|)$. This is an improvement in space from the current data structures [16] taking $O(n \log n)$ bits of space.

Comparison with Related Work. Range query structures have been used for many pattern matching queries, however rarely for achieving compressed space [11]. Kärkkäinen and Ukkonen [13] also explored the use of $\mathcal{RS2D}$ indexes for string matching and proposed the sparse suffix arrays; nevertheless, their query complexity was exponential in the worst case.

The problem of compressed text indexing ($\mathcal{CSA}$) in external memory has been a well-known open problem [18]. The question of developing compressed data structure for position-restricted queries was left open [12, 16]. No good lower bounds have been known except for the one by Demaine and López-Ortiz [6], which only goes to say that the data structures need to be at least the size of the input text.

2 The Geometric BWT

Let T be a text of n characters, denoted by $T[1..n]$. The substrings of T in the form $T[i..n]$ (for $i = 1, 2, \ldots, n$) are called the *suffixes* of T. The *suffix tree* is a compact trie storing the n suffixes of T. The *suffix array* $SA[1..n]$ is an array of n integers, where $SA[i]$ stores the starting position of the ith smallest suffix in the lexicographical order. If a pattern P appears in T, P is the prefix of some suffix of T, say $T[i..n]$; in this case, we say P occurs at position i. It is shown [17] that one can find ℓ and r such that $SA[\ell], SA[\ell+1], \ldots, SA[r]$ stores all positions where P occurs. We call $[\ell, r]$ the *SA range* of P in T.

Lemma 1 *We can index T in $O(n \log n)$ bits such that for any input pattern P, we can find the SA range $[\ell, r]$ of P in T using $O(|P|)$ time. Also, each SA value can be reported in $O(1)$ time.*

The *Burrows-Wheeler transform* of T is a text $BWT[1..n]$ such that $BWT[i] = T[SA[i]-1]$.[‡] So, $BWT[i]$ is the character preceding the ith smallest suffix.

The Geometric-BWT. Given a text T drawn from $\{1, 2, \ldots, |\Sigma|\}$ and a blocking factor d, GBWT(T, d) is a set S of n/d points (x_i, y_i).[‡‡] Let $T'[1..n/d]$ be the text formed by blocking every d characters of T to form a single meta-character. Thus, the suffix of T' at starting position i corresponds to the suffix of T starting at position $(i-1)d+1$. Let $SA'[1..n/d]$ be the suffix array of T'. For each character c appearing in T', its binary representation, denoted by bin(c), has $d \log |\Sigma|$ bits. Let $\overleftarrow{c}$ be the character such that bin($\overleftarrow{c}$) is the reverse bit-string of bin(c) of $d \log |\Sigma|$ bits, and we call $\overleftarrow{c}$ the *reverse character* of c.

The GBWT(T, d) is simply the set of n/d points $S = \{(\overleftarrow{T'[SA'[i]-1]}, i) \mid 1 \leq i \leq n/d\}$. Note that when the points in S are sorted (in increasing order) in the y-coordinates, the corresponding x-coordinates will be similar to the BWT of T', except that each character is replaced by its reverse character. The GBWT of T can be constructed in the same time as the BWT of T'. Given GBWT, T can be recovered in $O(n)$ time. Also, GBWT is space-preserving within a constant factor.

Definition of `Points2Text`. Let $S = \{(x_1, y_1), (x_2, y_2), \ldots, (x_n, y_n)\}$ be a set of n points in $\mathbb{N}^2$, such that the x-, or y- coordinate of each point is represented naturally in binary using $h = O(\log n)$ bits. Fix an alphabet $\{\mathtt{0}, \mathtt{1}, \#, \star\}$ (i.e., each character is encoded in two bits). Let $\langle x \rangle$ denote the string of h characters formed by translating each bit (0 or 1) in the representation of x into the corresponding character (`0` or `1`). To represent the n points of S, we construct a text T with alphabet $\{\mathtt{0}, \mathtt{1}, \#, \star\}$, known as the the `Points2Text` *transform* of S, as follows.

$$T = \langle \overleftarrow{x_1} \rangle \# \langle y_1 \rangle \star \langle \overleftarrow{x_2} \rangle \# \langle y_2 \rangle \star \cdots \langle \overleftarrow{x_n} \rangle \# \langle y_n \rangle.$$

The `Points2Text` of S can be constructed and inverted in $O(n)$ time in RAM. In addition, `Points2Text` is space-preserving within a constant factor.

[‡] In the special case where $SA[i] = 1$, we set $BWT[i] = T[n]$.

[‡‡] For simplicity, we assume n is a multiple of d. Otherwise, T is first padded with enough special character \$ at the end to make the length a multiple of d.

3 External Memory Succinct Text Index

We first show an alternative succinct text index in internal memory using GBWT. Then, we show that this index can be easily converted into an external memory index.

Let T be a text and T' be the meta-text formed by blocking every $d = \delta \log_{|\Sigma|} n$ characters of T into a single meta-character, with $\delta = 1/4$.[§] To obtain an $O(n \log |\Sigma|)$-bit text index, we first construct a data structure Δ consisting the suffix tree and suffix array of T', so that it occupies only $O((n/d)\log(n/d)) = O(n \log |\Sigma|)$ bits. With Δ, we can already support pattern searching, though in a very restricted form. Precisely, it can only report those occurrences of P in T which occur at positions of the form $id + 1$ (Note that $|P|$ does not need to be a multiple of d here.)

To extend the power of Δ, we obtain the points GBWT(T, d), sort them in the y-coordinates, and get the modified Burrows-Wheeler transform BWT_{mod} of T' by listing the corresponding x-coordinates. That is, $BWT_{mod}[i] = \overleftarrow{BWT[i]}$. After that, we construct the wavelet tree [10, 12, 16] (See Lemma 3 in Appendix) of BWT_{mod}. As each value of BWT_{mod} is character in T', it is represented in $d \log |\Sigma|$, so the wavelet tree takes $O((n/d)d \log |\Sigma|) = O(n \log |\Sigma|)$ bits.

We now show how to use the wavelet tree of BWT_{mod} to extend the searching power of Δ. Note that we can alternatively use any $\mathcal{RS2D}$ data structure on GBWT(T, d). We describe this in terms of wavelet tree because it is easy to later derive high-order entropy compressed index. In particular, we find all those occurrences of P in T with starting position *inside* a character in T'. That is, those occurring at positions i (in T) with $i \bmod d = k$, where k may not be 1. We call such an occurrence an *offset-k* occurrence. Here, we require that P is longer than $\pi = d - k + 1$, so that its offset-k occurrence does not start and end inside the same character in T'.

Let $\widehat{P}$ denote the prefix of P of length π (i.e., with $\pi \log |\Sigma|$ bits) and $\tilde{P}$ denote the suffix of P formed by taking $\widehat{P}$ away from P. We define two characters c_{min} and c_{max} as follows: Reverse the bit-string of $\widehat{P}$ and then append $(d - \pi)\log |\Sigma|$ bits of 0s to it, we obtain the $d \log |\Sigma|$ bit-string of the character c_{min}; if we append $(d - \pi)\log |\Sigma|$ bits of 1s instead, we obtain the bit-string of c_{max}.

To find all offset-k occurrences of P in T, it is sufficient to find all positions i' in T' such that $\tilde{P}$ occurs at i' in T',[‡‡] with the binary encoding of $\widehat{P}$ matching the suffix of the binary encoding of $T'[i' - 1]$. The latter happens if and only if $c_{min} \le \overleftarrow{T'[i' - 1]} \le c_{max}$. Based on this, the set of i's can be found as follows:

1. Search Δ to obtain the SA range $[\ell, r]$ of $\tilde{P}$, so that $SA'[\ell..r]$ are all occurrences of $\tilde{P}$ in T'.
2. Construct c_{min} and c_{max} based on $\widehat{P}$.
3. Search the wavelet tree to find all y's in $[\ell, r]$ with $c_{min} \le BWT_{mod}[y] \le c_{max}$.
4. Find $SA'[y]$ for all the y's in Step 3 (using the suffix array Δ built on T'), which are the offset-k occurrences of P.

[§]For simplicity, we assume that d is an integer. If not, we can slightly modify the data structures without affecting the overall complexity.

[‡‡]Precisely, $\tilde{P}$ occurs at $(i' - 1)d + 1$ in T.

We apply the above to find offset-k occurrences of P for $k = 2, 3, \ldots, d$, thus giving:

Lemma 2 *Based on Δ and the wavelet tree of BWT_{mod}, all occurrences of P with starting and ending positions inside different characters of T' can be found in $O(|P| + (\log n)(\log_{|\Sigma|} n) + occ \log n)$ time.*

It remains to show how to find those occurrences of P that start and end in the *same* character of T'. We claim that this can be solved in $O(|P| + occ)$ time, by using an $o(n)$-bit auxiliary data structure based on the standard four-russians technique. First, for each character c in T', if c appears at least $\sqrt{n}$ times, we say c is a *frequent* character. Otherwise, c is an *infrequent* character.

For each frequent character c, we maintain a list of positions in T' where it appears. Because each occurs at least $\sqrt{n}$ times, there are at most $O(\sqrt{n})$ of them. We now treat each of the *distinct* frequent characters as a string of d original characters, and construct a *generalized* suffix tree on these strings.¶ Then, on any input pattern P, we can find all distinct frequent characters containing P (and the exact position(s) where P is located inside each frequent character). Also, the corresponding pointers to their lists of positions can be returned. Precisely, once we have searched the generalized suffix tree and found that P appears at position α in a particular frequent character c, we can conclude that P appears at position $d\gamma_1 + \alpha, d\gamma_2 + \alpha, \ldots,$ in T, where γ_i denotes the ith occurrence of c in T'. Thus, the time to output all occurrences of P appearing in all frequent characters is $O(|P| + occ)$ time. The space is dominated by the generalized suffix tree, which requires $O(\sqrt{n} \times d \times \log n) = o(n)$ bits.

The number of distinct infrequent characters is at most $2^{\delta \log n} = n^{1/4}$, as each character is represented in $\delta \log n$ bits. In total, the number of infrequent characters (counting repeats) is at most $n^{1/4} \times \sqrt{n}$. We treat each of them as a string of d original characters and construct a generalized suffix tree for these strings. Then, on any input pattern P, we can find all occurrences of P appearing inside all (possibly *non-distinct*) infrequent characters in $O(|P| + occ)$ time. The space of this suffix tree is $O(n^{1/4} \times \sqrt{n} \times d \times \log n) = o(n)$ bits.

In conclusion, the desired occurrences of P can be found in $O(|P| + occ)$ time, and the space of the overall data structure is $o(n)$-bits. Combining this with Lemma 2, we have the theorem below:

Theorem 1 *We can index a text T in $O(n \log |\Sigma|)$ bits such that we can find all occurrences of a pattern P in T in $O(|P| + (\log n)(\log_{|\Sigma|} n) + occ \log n)$ time.*

Extension to External Memory Index. Let B be the size of a disk page. Our first result simply replaces each data structure used in Section 3 by its external memory counter-part. That is, we replace Δ of T' by the string B-tree [7] of T' and the wavelet tree of BWT_{mod} by the external memory wavelet tree of BWT_{mod} (See Appendix); for the suffix trees inside the data structures that search short patterns, they are replaced by string B-trees as well. Immediately, we obtain:

¶Given a set of strings $S_1, S_2, \ldots, S_k$, the generalized suffix tree is a compact trie storing all suffixes of all S_i's. Searching of a pattern is done simultaneously in all S_i's.

Theorem 2 *We can index a text T in $O(n \log |\Sigma|)$ bits such that we can find all occurrences of a pattern P in T in $O(|P|/B + \log_{|\Sigma|} n \log_B n + occ \log_B n)$ I/Os.*

The additive term $occ \log_B n$ in the query I/O is not optimal. Here, we show that this factor can be achieved when patterns are sufficiently large. First, we use a new blocking factor $d = (\log^2 n)/((\log |\Sigma|)(\log \log_B n))$ and block the text T accordingly.[‖] We maintain the string B-tree of the blocked text T'. Instead of using the external memory wavelet tree to index BWT_{mod}, we use the four-sided query index I of Arge et al. [3] to store the points $(i, BWT_{mod}[i])$. It is easy to check that the index I performs the desired query supported by the wavelet tree of BWT_{mod}. This gives:

Theorem 3 *We can index a text T in $O(n \log |\Sigma|)$ bits such that whenever P is longer than $d = (\log^2 n)/((\log |\Sigma|)(\log \log_B n))$, we can find all occurrences of P in T in $O(|P|/B + d \log_B n + occ/B)$ I/Os, where occ is the number of occurrences.*

3.1 Lower bounds using `Points2Text`

We first demonstrate the reduction from 2-dimensional range query with n points in $[1, n] \times [1, n]$ (so that each point is represented in $h = \Theta(\log n)$ bits) to text searching. Based on this, we can obtain the lower bound result for pattern matching with succinct text index. The general case for reducing range query in $\mathbb{R}^2$, assuming each point is represented in h' bits, can be handled easily by storing a sorted array of x values and a sorted array of y values, using $O(nh')$-bit extra space and $O(\log n)$ extra time for coordinate translation.

Let $S = \{(x_1, y_1), (x_2, y_2), \ldots, (x_n, y_n)\}$ be a set of n points in $[1, n] \times [1, n]$, and T be the text T in the `Points2Text` of S. On an input ranges $[x_{left}, x_{right}]$ and $[y_{bottom}, y_{top}]$, it is easy to see that finding all points in S that fall inside the ranges can be done by issuing the corresponding $(x_{right} - x_{left}) \times (y_{top} - y_{bottom})$ pattern searching queries in T. In fact, we can limit the number of pattern queries to $O(\log^2 n)$ by the following observation.

Observation 1 *For any k and any i, we call the range $[k2^i, (k+1)2^i - 1]$ a* complete range*, which is denoted by $R_{k,i}$. That is, the range contains all 2^i numbers whose quotient, when divided by 2^i, is k. For any range $[\ell, r]$ with $1 \leq \ell \leq r \leq 2^h$, it can be partitioned into at most $2h$ complete ranges in $O(h)$ time.*

Now, suppose that $R_{k,i}$ is a complete range in $[x_{left}, x_{right}]$ and $R_{k',i'}$ is a complete range in $[y_{bottom}, y_{top}]$. Then, each point in S with x-coordinate falling in $R_{k,i}$ and y-coordinate falling in $R_{k',i'}$ corresponds to exactly a substring of T in the form of:

$$\text{Last } h - i \text{ characters of } \langle \overleftarrow{k2^i} \rangle, \text{ then } \#, \text{ then first } h - i' \text{ characters of } \langle k'2^{i'} \rangle.$$

Thus, we need to issue only $4h^2 = \Theta(\log^2 n)$ pattern queries in T. So we have:

[‖]Note that choosing larger d allows more sparsification, but it is not possible to design the four-russians data structure for small patterns in such cases.

Theorem 4 *Given a set S of n points in $\mathbb{R}^2$, each represented in h' bits, we can construct two sorted arrays for storing the x-coordinates and y-coordinates, and a text T of length $O(n \log n)$ with characters chosen from an alphabet of size 4; then, a four-sided range query on S can be answered by $O(\log^2 n)$ pattern match queries on T, each query searching a pattern of $O(\log n)$ characters.*

Chazelle [5] showed that in a pointer machine, an index supporting d-dimensional range searching in $O(\text{polylog}(n) + occ)$ query time requires $O(n(\log n / \log\log n)^{d-1})$ words of storage. Combining this ($d = 2$) with Theorem 4, we have:

Theorem 5 *In pointer machine, an index on $T[1..n]$ supporting pattern matching in $O(\text{polylog}(n) + occ)$ time requires $\Omega((n \log n)/(\log\log n))$ bits in the worst case.*

We can also apply Theorem 4 to obtain a lower bound in the external memory model. Subramanian and Ramaswamy [19] showed that any external memory data structure that can answer 2-D orthogonal range query in $O((\log_B n)^c + occ/B)$ I/Os, for any c, must use at least $\Omega((n \log n)/(\log\log_B n))$ words. So, we have:

Theorem 6 *An external memory index on $T[1..n]$ supporting pattern matching in $O((\log_B n)^c / \log^2 n + occ/B)$ I/Os needs $\Omega((n \log n)/\log\log_B n)$ bits in the worst case.*

4 Succinct Index for Position-Restricted Query

Given a text T and a pattern P, and two positions i and j, the *position-restricted query* finds all occurrences of P in T whose starting positions are between i and j. Our index is defined as follows: We use a similar blocking technique as in Lemma 2. Let $T[1..n]$ be a text with characters drawn from $\{1, 2, \ldots, |\Sigma|\}$. Let $d = \log^{2+\epsilon} n / \log |\Sigma|$ for some fixed $\epsilon > 0$ and $T'[1..n/d]$ be a meta-text formed by blocking every d characters in T into a meta-character. Let SA' denote the suffix array of T'. Also, we re-use the notations $\overleftarrow{c}$, $\widehat{P}$, $\tilde{P}$, c_{min}, and c_{max} in Section 3 analogously.

We construct a data structure Δ consisting of the suffix tree and suffix array of T'. Now, we obtain the set of points S in 3-dimensions. For this, we use augmented-GBWT which also adds z-coordinate as $SA'[i]$ to the x and y coordinates obtained by GBWT. Thus, S consists of points of the form $(\overleftarrow{T'[SA'[i]-1]}, i, SA'[i])$. Now, we construct an index I for S such that $\mathcal{RS3D}$ can be answered in $O(\log n + k)$ time [2]. The sizes of both data structures are $O(n \log |\Sigma|)$ bits.

When P is longer than d, any offset-k occurrence of P with starting position between i and j in the original text T must have $\tilde{P}$ occurring at some position x in T', $\widehat{P}$ matching the "suffix" of $T'[x-1]$, and x between $i' = \lceil (i + |\widehat{P}|)/d \rceil$ and $j' = \lceil (j + |\widehat{P}|)/d \rceil$. Thus, all offset-$k$ occurrences can be found as follows:

1. Search $\tilde{P}$ in Δ to obtain the SA range $[\ell, r]$ of $\tilde{P}$ in T'.
2. Construct c_{min} and c_{max} based on $\widehat{P}$. Compute i' and j'.
3. Search I for all (x, y, z) such that $z \in [i', j']$, $x \in [c_{min}, c_{max}]$, and $y \in [\ell, r]$.
4. The z values of all points obtained in Step 3 correspond to offset-k occurrences of P. (Precisely, P appears at positions $(z-1)d + k$ in T for all z.)

The position-restricted occurrences of P can thus be obtained by finding all offset-k occurrences of P in the above process, for $k = 1, 2, \ldots, d$. The total time to obtain all SA ranges for d times is $O(|P|)$. The total time to search I for d times is $O(d \log^2 n + occ)$, where occ is the number of position-restricted occurrences of P. Combining the two terms gives the following theorem.

Theorem 7 *For a fixed $\epsilon > 0$, we can index T in $O(n \log |\Sigma|)$ bits such that for any input pattern P longer than $(\log^{2+\epsilon} n)/(\log |\Sigma|)$ and any input positions i and j, we can support the position-restricted query in $O(|P| + (\log^{4+\epsilon} n)/(\log |\Sigma|) + occ)$ time, where occ is the number of occurrences.*

Lower Bound. Here, we reduce the 3-dimension range query about n points in $[1, n] \times [1, n] \times [1, n]$ to position-restricted query. The general case for reducing range query in $\mathbb{R}^3$, when each point is represented in h' bits, can be handled easily with $O(nh')$-bit extra space and $O(\log n)$ extra time for coordinate translation.

Let $S = \{(x_i, y_i, z_i) \mid 1 \leq i \leq z\}$ be a set of n points in $[1, n] \times [1, n] \times [1, n]$. We perform `Points2Text` transform on S to obtain a text T and an array Z, where we assume $z_1 \leq z_2 \leq \ldots z_n$ and $Z[i] = z_i$. Recall that T is in the form

$$T = \langle \overleftarrow{x_1} \rangle \# \langle y_1 \rangle \star \langle \overleftarrow{x_2} \rangle \# \langle y_2 \rangle \star \cdots \langle \overleftarrow{x_n} \rangle \# \langle y_n \rangle.$$

Let $e = 2h+2$ denote the length of the string $\langle \overleftarrow{x_i} \rangle \# \langle y_i \rangle \star$. On input ranges $[x_{left}, x_{right}]$, $[y_{bottom}, y_{top}]$, and $[z_{front}, z_{back}]$, let i denote the minimum k with $Z[k] \geq z_{front}$ and j denote the maximum k with $Z[k] \leq z_{back}$. Then, finding all points in S that fall inside the ranges can be done by searching the substring representing (x, y) in T for all $x \in [x_{left}, x_{right}]$ and $y \in [y_{bottom}, y_{top}]$ with positions restricted by $(i-1)e+1$ and $(j-1)e+1$. Again by Observation 1, we can limit the number of position-restricted pattern queries to $O(\log^2 n)$. This gives the following theorem.

Theorem 8 *Given a set S of n points in $\mathbb{R}^2$, each represented in h' bits, we can construct a sorted array for each of the x-, y-, and z- coordinates, and a text T of length $O(n \log n)$ with characters chosen from an alphabet of size 4; then, a 3-D orthogonal range query on S can be answered by $O(\log^2 n)$ position-restricted pattern searching queries on T, with each query searching a pattern of $O(\log n)$ characters.*

Then, we can combine Chazelle's lower bound (with $d = 3$) and obtain:

Theorem 9 *In pointer machine, an index on $T[1..n]$ supporting position-restricted query in $O(\text{polylog}(n) + occ)$ time needs $\Omega(n(\log n / \log \log n)^2)$ bits in the worst case.*

References

[1] P. K. Agarwal and J. Erickson. Geometric Range Searching and Its Relatives. *Advances in Discrete and Computational Geometry*, 23:1–56, 1999.

[2] S. Alstrup, G. S. Brodal, and T. Rauhe. New Data Structures for Orthogonal Range Searching. In *FOCS*, pages 198–207, 2000.

[3] L. Arge, G. S. Brodal, R. Fagerberg, and M. Laustsen. Cache-Oblivious Planar Orthogonal Range Searching and Counting. In *SOCG*, pages 160–169, 2005.

[4] M. Burrows and D. J. Wheeler. A Block-Sorting Lossless Data Compression Algorithm. Technical Report 124, Digital Equipment Corporation, Paolo Alto, CA, USA, 1994.

[5] B. Chazelle. Lower Bounds for Orthogonal Range Searching, I: The Reporting Case. *Journal of the ACM*, 37:200–212, 1990.

[6] E. D. Demaine and A. López-Ortiz. A Linear Lower Bound on Index Size for Text Retrieval. In *Proceedings of Symposium on Discrete Algorithms*, pages 289–294, 2001.

[7] P. Ferragina and R. Grossi. The String B-tree: A New Data Structure for String Searching in External Memory and Its Application. *JACM*, 46(2):236–280, 1999.

[8] P. Ferragina and G. Manzini. Indexing Compressed Text. *JACM*, 52(4):552–581, 2005.

[9] P. Ferragina and R. Venturini. A Simple Storage Scheme for Strings Achieving Entropy Bounds. In *Proceedings of Symposium on Discrete Algorithms*, pages 690–696, 2007.

[10] R. Grossi, A. Gupta, and J. S. Vitter. High-Order Entropy-Compressed Text Indexes. In *Proceedings of Symposium on Discrete Algorithms*, pages 841–850, 2003.

[11] R. Grossi and J. S. Vitter. Compressed Suffix Arrays and Suffix Trees with Applications to Text Indexing and String Matching. *SIAM J. Comput.*, 35(2):378–407, 2005.

[12] W.-K. Hon, R. Shah, and J. S. Vitter. Ordered Pattern Matching: Towards Full-Text Retrieval. Technical Report TR-06-008, Purdue University, March 2006.

[13] J. Kärkkäinen and E. Ukkonen. Sparse Suffix Trees. In *Proceedings of International Conference on Computing and Combinatorics*, pages 219–230, 1996.

[14] G. Navarro and V. Mäkinen. Compressed Full-Text Indexes. *ACM CSUR*, 39(1), 2007.

[15] V. Mäkinen and G. Navarro. Dynamic Entropy-Compressed Sequences and Full-Text Indexes. To appear in *ACM TALG*.

[16] V. Mäkinen and G. Navarro. Position-Restricted Substring Searching. In *LATIN*, pages 703–714, 2006.

[17] U. Manber and G. Myers. Suffix Arrays: A New Method for On-Line String Searches. *SIAM Journal on Computing*, 22(5):935–948, 1993.

[18] S. J. Puglisi, W. F. Smyth, and A. Turpin. Inverted Files Versus Suffix Arrays for Locating Patterns in Primary Memory. In *SPIRE*, pages 122–133, 2006.

[19] S. Subramanian and S. Ramaswamy. The P-range Tree: A New Data Structure for Range Searching in Secondary Memory. In *SODA*, pages 378–387, 1995.

A Appendix

Simple Wavelet Tree as a Data Structure for $\mathcal{RS2D}$. Given an array $A[1..m]$ of m integers drawn from $[1,n]$, a simple implementation of the wavelet tree [10, 16] supports on an input range $[\ell,r]$ and an input value y, finding all z's in $[\ell,r]$ such that $A[z]=y$ in $O((occ+1)\log n)$ time, where occ denotes the number of z's in the output. In fact, the wavelet tree can be generalized easily so that we achieve the following results [12].

Lemma 3 *We can index A in $O(m\log n)$ bits such that on an input range $[\ell,r]$ and an input bound $[x,y]$, we can output all $z\in[\ell,r]$ such that $x\le A[z]\le y$ in $O((1+occ)\log n)$ time (or, $O((1+occ)\log_B n)$ I/Os in external memory), where occ denotes the size of the output.*