2018 IEEE 34th International Conference on Data Engineering

CiNCT: Compression and Retrieval for Massive Vehicular Trajectories via Relative Movement Labeling

Satoshi Koide #*1, Yukihiro Tadokoro #2, Chuan Xiao *3, Yoshiharu Ishikawa *4

# *Toyota Central R&D Labs., Inc., Nagakute, Aichi, Japan*
* *Nagoya University, Nagoya, Aichi, Japan*

1 `koide@mosk.tytlabs.co.jp` 2 `tadokoro@mosk.tytlabs.co.jp`
3 `chuanx@nagoya-u.jp` 4 `ishikawa@i.nagoya-u.ac.jp`

***Abstract*—In this paper, we present a compressed data structure for moving object trajectories in a road network, which are represented as sequences of road edges. Unlike existing compression methods for trajectories in a network, our method supports pattern matching and decompression from an arbitrary position while retaining high compressibility with theoretical guarantees. Specifically, our method is based on FM-index, a fast and compact data structure for pattern matching. To further enhance the compression performance, we incorporate the *sparsity* of road networks. In particular, we present the novel concepts of *relative movement labeling* and *PseudoRank*, each contributing to significant reduction in data size and query processing time. Our theoretical analysis and experimental studies reveal the advantages of our proposed method as compared to existing trajectory compression methods and FM-index variants.**

I. Introduction

In recent years, a vast amount of trajectory data from moving objects, such as automobiles, has become available. According to Han et al. [1], the total amount of GPS trajectories generated by automobiles in the U.S. alone exceeded 53 TB in 2011. With recent increased interests in the use of such large datasets in wide range of data-driven applications, fundamental data manipulations such as retrieval and compression are once again becoming crucial. In this paper, we focus on moving object trajectories in (road) networks, called *network-constrained trajectories* (NCTs), one of the most important types of trajectories with many practical applications. Traveled paths of NCTs can be represented as symbol sequences of road segment IDs. Although this representation is more compact than GPS coordinates, it is still insufficient for the vast datasets that are now available. Therefore, compressed representations of NCTs have been studied thus far [1–5].

If trajectories are simply compressed without an augmented data structure, it is difficult to use them in real applications. Therefore, compression methods that allow several operations without decompressing the entire dataset are necessary, and such methods have been the focus of recent studies. For example, such studies include the in-memory data structures proposed in [3] and [4], as well as an in-memory/on-disk hybrid structure proposed in [6]. In our present paper, we propose a method that realizes a high-level compression while retaining a high utility of the data. As the background and motivation for our method, we first review the existing compressed data structures for NCTs and their functions below.

NCTs consist of spatial paths and corresponding timestamps. Therefore, we must consider compression of these paths and timestamps separately. For spatial paths, lossless compression methods based on *shortest-path encoding* have been studied in [1], [2], and [4]. Here, to compress the data, these methods remove partial shortest paths in an NCT because these paths can be recovered from the road network itself. One drawback of this approach is that it cannot guarantee the information-theoretic upper bound of the compressed data size. A recent lossless path compressor introduced in [1] called *minimum entropy labeling* (MEL) guarantees a theoretic bound and also achieves practically higher compressibility than the shortest-path encoding methods. As for the timestamps, all methods noted above compress them independently from the spatial path compression. In this paper, we do not discuss the compression of timestamps directly, but we emphasize here that our method can be easily combined with such temporal compression methods (see Sec. IV-D and Sec. VII for details).

In general, it is difficult to define high utility of compressed NCTs, because their utility depends on the application. In this paper, we focus on two functions, i.e., pattern matching without decompressing the entire dataset, and extracting subpaths from an arbitrary position. Intuitively, pattern matching operations that find trajectories along a given path would have wide applications in NCT processing. In fact, the existing methods mentioned above (i.e., [3], [4], and [6]) closely relate to pattern matching; however, to the best of our knowledge, there are no NCT compressors that guarantee theoretical bound for the compressed size while supporting fast pattern matching.

Given the above background, our research question is, *how can we realize high compressibility while enabling pattern matching for NCTs?* To address this question, we focus on *suffix arrays* [7], data structures related to pattern matching. Although the data structures for NCTs proposed in [3] and [6] also employ suffix arrays, they do not focus on a compression method, instead use existing general-purpose compressed suffix arrays that are typically used for genomic sequences. Unfortunately, these existing methods are inefficient because NCTs consist of a large alphabet (i.e., road segment IDs in a potentially large road network) whereas genomic sequences include only four characters (i.e., A, C, G, and T).

NCTs have another noteworthy feature, i.e., *they can only move along physically connected road segments*. This feature

2375-026X/18/$31.00 ©2018 IEEE
DOI 10.1109/ICDE.2018.00102

is quite different from general sequences, as illustrated in Fig. 1. In Fig. 1(a), we show four example NCTs in a small network with six road segments (A–F). The corresponding graph shown in Fig. 1(b) represents symbol transitions for these four NCTs. Here, each vertex corresponds to a symbol (i.e., a road segment), and directed edges exist between two vertices if the corresponding two symbols can appear successively. For example, in Fig. 1(b), vertex A is connected with vertexes B and D because we can only move to road segment B or D from A. For NCTs, this *empirical transition graph* (ET-graph) becomes a sparse graph, reflecting the physical topology of road networks. This sparsity cannot be obtained for general sequences, which leads to a denser ET-graph, as illustrated in Fig. 1(c).

Our proposed method, *Compressed-index for NCTs* (*CiNCT*), significantly improves the compression and pattern matching operations when applied to sequences with such sparse ET-graphs. Our method is based on *FM-index* [8], a compressed data structure for suffix arrays, which we describe further in Section II. Note that it is challenging to incorporate such sparsity into FM-index while retaining its theoretical advantages because FM-index is compressed at the bit-level. Therefore, in the remainder of our paper, we introduce some novel techniques and provide theoretical analysis that explains why our method yields substantial improvement in practice.

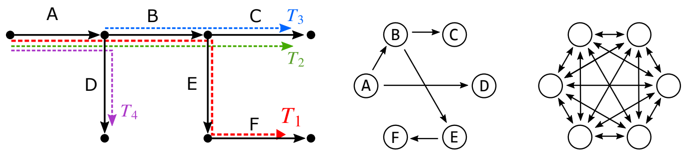
Contributions: To develop a data structure for NCTs that simultaneously achieves a high compression ratio and high utility, we propose CiNCT, as a novel method to compress suffix arrays for sequences on a sparse graph. We summarize our contributions as follows.

- We propose *relative movement labeling* (RML), which converts sequences on a sparse graph to low-entropy sequences. We theoretically prove its optimality and show that RML provides a more compact representation of NCTs than that of the MEL method [1].
- We incorporate RML into FM-index by introducing a new concept called *PseudoRank*, which leads to significant improvements in both size and query processing speed (i.e., the speed of pattern matching and sub-path extraction) as compared to existing FM-index variants. We also explain theoretically why this occurs.
- Using several real NCT datasets, we show that our method outperforms the state-of-the-art methods that do not consider graph sparsity.

Outline: The remainder of our paper is organized as follows: preliminaries (Section II), proposed data structure (Section III), proposed algorithms (Section IV), theoretical analysis (Section V), experiments (Section VI), related work (Section VII), and conclusion (Section VIII).

II. Preliminaries

In this section, we introduce the data models and pattern matching query. For readers not familiar with string processing and indexing, we also describe the basic concepts regarding FM-index. This is a data structure in which the *Burrows–Wheeler transform* (Sec. II-A2) of a target string



(a) Network-constrained trajectories (NCTs) (b) ET-graph for NCTs (c) ET-graph for general strings

Fig. 1. (a) Network-constrained trajectories (NCTs), and both (b) sparse and (c) dense symbol transition graphs (ET-graphs).

TABLE I
Notation

Symbol	Description	Defined in
$w, w' \in E$	Road segments (characters)	—
T, T_{bwt}	Trajectory string and its BWT	Def. 2, Fig. 2
$S[i,j)$	Substring of S from i to $j-1$	§ II-A1
Σ, σ	Alphabet set and its size	§ II-A1
$R(P)=[sp,ep)$	Suffix range of a pattern P	§ II-A2
$C[w]$	The number of w' in T s.t. $w'<w$	§ II-A3
$H_0(S)$, $H_k(S)$	0th and kth order empirical entropy	Eq. (3), (4)
G_T, E_T	ET-graph and its edge set	§ III-B (Def. 4)
ϕ	Relative movement labeling func.	§ III-B1
$Z_{w'w}$	Correction term	Eq. (7)

is stored in a compressed data structure called *wavelet tree* (Sec. II-A4). Specifically, we introduce Huffman-shaped wavelet tree (HWT) and the related complexities. In Sec. II-B, further compression technique called *compression boosting* is introduced with the related compressed bit vector called *RRR*. With the compression boosting, FM-index can be compressed to the *kth order empirical entropy* (Eq. (4)). At the end of this section, we discuss the issues of these techniques. The relationship between these techniques are shown as the dotted line in Fig. 5. Table I shows the notation used in this paper.

A. Definitions

1) Data models: First, we define NCTs as follows.

Definition 1: A *network-constrained trajectory* (NCT) on a directed graph (V, E) is defined as a sequence of physically connected road segments, i.e., $e_1e_2\cdots e_n$ $(e_i \in E)$.

For example, we have $e_1 = \mathsf{A}$, $e_2 = \mathsf{B}$, $e_3 = \mathsf{E}$, and $e_4 = \mathsf{F}$ for $T_1 = \mathsf{ABEF}$ illustrated in Fig. 1 (a). To build an FM-index for a set of documents, they are usually concatenated into one long string [6]. Similarly, we define a *trajectory string* that concatenates the NCTs.

Definition 2 (Trajectory string): Let $\mathcal{T} := \{T_k\}_{k=1}^N$ be a set of NCTs to be indexed. A *trajectory string* is defined as $T := T_1^{\text{rev}}\$T_2^{\text{rev}}\$\cdots T_N^{\text{rev}}\$\#$, where T_k^{rev} is the reversal of string T_k, and \$ and # are special symbols that represent NCT boundaries and the end of the string, respectively.

For the four NCTs in Fig. 1 (a), the trajectory string is

$$T = \underbrace{\mathsf{FEBA}}_{T_1^{\text{rev}}}\$\underbrace{\mathsf{CBA}}_{T_2^{\text{rev}}}\$\underbrace{\mathsf{CB}}_{T_3^{\text{rev}}}\$\underbrace{\mathsf{DA}}_{T_4^{\text{rev}}}\$\#. \tag{1}$$

In the later sections, we use this example for explanation. In this paper, a string S has 0-based subscripts and $|S|$ denotes its length. $S[i]$ and $S[i,j)$ are the i-th element and the substring from i to $j-1$, respectively. The alphabet set is defined as $\Sigma := E \cup \{\$, \#\}$, and σ denotes its size. To define the BWT below, we assume a lexicographical order on the road

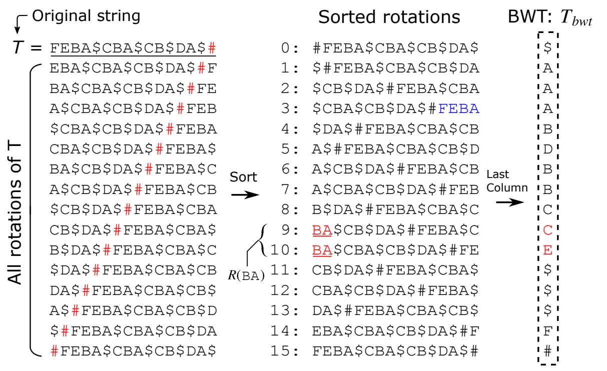


Fig. 2. The BWT of T is defined to be the last column of the sorted rotations of T. This example is based on the trajectory string T in Eq. (1).

Algorithm 1: Finding the suffix range $R(P) = [sp, ep)$ for a given query P of length m based on T_{bwt} (*SearchFM*)

```
  Input: BWT string: $T_{bwt}$, Query string: $P$,
         Integer array: $C$
  Output: Range of $T_{bwt}$ that matches to $P$
1 $w \leftarrow P[m-1]$; $sp \leftarrow C[w]$; $ep \leftarrow C[w+1]$
2 for $i \leftarrow 2$ to $m$ do
3     $w \leftarrow P[m-i]$
4     $sp \leftarrow C[w] + rank_w(T_{bwt}, sp)$
5     $ep \leftarrow C[w] + rank_w(T_{bwt}, ep)$
6     if $sp \geq ep$ then return NotFound
7 return $[sp, ep)$
```

segments E (any ordering can be used for our purpose). The lexicographical order is assumed to be $\# < \$ < w$ ($\forall w \in E$).

2) Pattern matching and BWT: The *Burrows–Wheeler transform* (BWT) [9] is closely related to pattern matching and is used in FM-index. It is a reversible transform of T, defined to be the last column of the lexicographically sorted rotations of T (Fig. 2). For trajectory string Eq. (1), we have

$$T_{bwt} = \texttt{\$AAABDBBCCE\$\$\$F\#}. \tag{2}$$

For a given pattern (string) P, we can define a unique range $R(P) = [sp, ep)$ for which the prefixes of the corresponding sorted rotations are equal to P. We call this range the *suffix range* of P. For example, if $P = \mathsf{BA}$, we have $R(P) = [9, 11)$ (see the underlined prefixes in Lines 9 and 10 in Fig. 2).

Definition 3 (Pattern matching): Finding $R(P)$ for a given P is called *pattern matching*, or *suffix range query*.

For NCTs, pattern matching for a trajectory string T finds a suffix range of a given spatial path P. It is known that the suffix range is useful in spatio-temporal query processing for NCTs (see Section VII). This is why we focus on this query.

In this paper, we also focus on another query, *sub-path extraction query*, that recovers a sub-path from an arbitrary position in BWT T_{bwt}. We describe this query in Section IV-C.

3) FM-index and an algorithm to find suffix ranges: The FM-index [8] is a data structure that compresses a large string and indexes it at the same time. Specifically, the FM-index of a string T is a data structure in which the BWT of T is stored in a *wavelet tree*. Suffix range queries can be processed rapidly by FM-index. In the following, we overview how it works. It is known that Algorithm 1 can find the suffix range $R(P)$ for any P based on T_{bwt}. The *rank function*, $rank_w(T_{bwt}, i)$, returns the number of occurrences of a symbol $w \in \Sigma$ in a substring $T_{bwt}[0, i)$. For example, we have $rank_{\mathsf{B}}(T_{bwt}, 5) = 1$ because

$$T_{bwt} = \overbrace{\mathsf{\$AAA\underline{B}}}^{T_{bwt}[0,5)}\mathsf{DBBCCE\$\$\$F\#}.$$

Moreover, $C[w]$ is the number of symbols in T_{bwt} that are lexicographically smaller than w. For example, we have $C[\mathsf{A}] = 5$ and $C[\mathsf{B}] = 8$ by simple counting. The range $[C[w], C[w+1])$ defines the suffix range $R(w)$ (e.g., $R(\mathsf{A}) = [5, 8)$; see that A appears as the prefixes in $[5, 8)$ in Fig. 2).

To understand how Algorithm 1 works, let us consider a query $P = \mathsf{BA}$. In Line 1, we have $w = \mathsf{A}$, $sp = 5$, and $ep = 8$. Consider the first (and last) iteration with $i = 2$. We have $sp = C[\mathsf{B}] + 1 = 9$ and $ep = C[\mathsf{B}] + 3 = 11$ because $rank_{\mathsf{B}}(T_{bwt}, sp) = 1$ and $rank_{\mathsf{B}}(T_{bwt}, ep) = 3$ by definition. Therefore $[sp, ep) = [9, 11)$ is returned at Line 7, which is equivalent to $R(\mathsf{BA})$ given in Fig. 2.

We can say that fast calculation of $rank_w$ enables the fast execution of Algorithm 1 because all the operations except for $rank_w(T_{bwt}, i)$ are merely either substitutions or summations. However, naïve calculation of $rank_w$ with cumulative counting incurs an unacceptable $O(|T_{bwt}|)$ time.

4) Wavelet tree: A *wavelet tree* [10] storing T_{bwt} enables fast calculation of $rank_w(T_{bwt}, i)$; its time complexity does not depend on the data size $|T_{bwt}|$. Figure 3 illustrates a wavelet tree for the string $S = T_{bwt}$ in Eq. (2). The bit representation of each symbol is predefined (e.g., Huffman coding based on the frequency of each symbol in T_{bwt}). Each node v in the tree stores a bit vector B_v. For the root node v_0, B_{v_0} stores the most significant bit (MSB) of each symbol in S. At the second level, the symbols are divided into two parts based on the bit value at the first level, while keeping the ordering. Each bit vector stores the second MSB. Repeating such partitioning recursively, we obtain the wavelet tree. In fact, B_v is stored in a *succinct dictionary* [11], [12], which is a bit vector that supports a bit-wise rank (i.e., $rank_0(B_v, j)$ and $rank_1(B_v, j)$) in $O(1)$ time.

There are several types of wavelet tree with different compression characteristics that are determined by tree shape and the type of succinct dictionary. In CiNCT, we use a *Huffman-shaped wavelet tree* (HWT) [13], whose tree shape is that of the Huffman tree of S. It is known that an HWT can compress a string S of length n to at most $n(1 + H_0(S)) + o(n)$ bits. Here, $H_0(S)$ is the *0th order empirical entropy* [14]:

$$H_0(S) = \sum\nolimits_{w \in \Sigma} \frac{n_w}{n} \lg \frac{n}{n_w}, \tag{3}$$

where n_w is the number of occurrences of w in S.

To calculate $rank_w(S, j)$, the wavelet tree calculates the bit-wise rank value at each node $v_0, v_1, \cdots, v_k$ between the root and the leaf corresponding to the bit representation

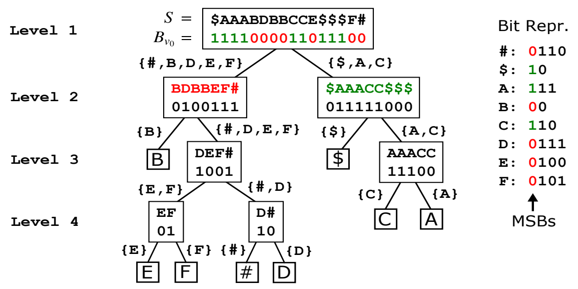


Fig. 3. Wavelet tree: a bit representation of each symbol in a string S is stored in a binary tree (this example is the HWT of the BWT of the trajectory string Eq. (2)). Note that only bit vectors are stored in each node.

$w = b_0 b_1 \cdots b_k$ (see [10] for details). This indicates that bit-wise rank operations required to obtain $rank_w(S, j)$ is equal to k (i.e., the length of the bit representation of w). This fact leads to the following result [13].

Theorem 1 (Rank on HWT): If $rank_w(S, j)$ is executed on uniformly random w over $S[0, n)$, it runs in $O(1 + H_0(S))$ time on average.

This result implies that a string with small entropy $H_0(S)$ achieves not only a small size but also a fast rank operation, which plays an important role in our theoretical analysis.

B. Compressed variants of FM-index

Here, we consider further compression of FM-index. Let us consider a sub-path of length 3 in a real NCT dataset: $e_{t-2}\, e_{t-1}\, e_t$. It is unlikely that two right turns occur in a row because most vehicles go toward their destinations. Considering such high-order correlations among symbols, we can boost the compression. As noted before, the prefix BA $\in \Sigma^2$ appears in $[9, 11)$ (Fig. 2). The other prefixes $W \in \Sigma^2$ have their corresponding ranges.

Let us divide T_{bwt} based on such prefixes W (called *contexts* of length two) as in Fig. 4. These context blocks represent the next segment e_t given the context $W = e_{t-1}\, e_{t-2}$. We have a chance of compression because the frequency of symbols in each context is biased as discussed above.

1) Compression boosting (CB): The above idea can be generalized to any length of context. Let us divide T_{bwt} into l blocks of context $W \in \Sigma^k$ of length k: $T_{bwt} = L_1 L_2 \cdots L_l$ ($l \le \sigma^k$). Storing each L_j in a 0-th order entropy compressor such as an HWT, we can compress T_{bwt} to $nH_k(T) + o(n)$. Here, H_k is k-th order empirical entropy [14]:

$$H_k(T) := \sum\nolimits_{W \in \Sigma^k} \frac{n_W}{n} H_0(T_W), \tag{4}$$

where T_W is the concatenation of all symbols in T that precede the context W. To support a fast rank operation on those divided blocks, we need to precompute and store the rank results at each location of l blocks for all $w \in \Sigma$.

Taking larger k seems to be desirable because $H_k(T) \ge H_{k+1}(T)$ for all $k \ge 0$ [14]. However, partitioning into many blocks leads to the following problems in practice:

P1) Blocks of variable length lead to inefficient random access to T_{bwt}.

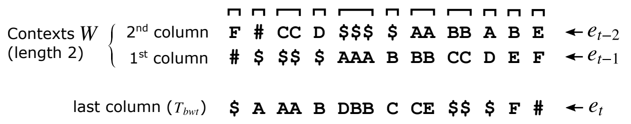

Contexts W (length 2)
2nd column F # CC D $$$ $ AA BB A B E ← e_{t-2}
1st column # $ $$ $ AAA B BB CC D E F ← e_{t-1}
last column (T_{bwt}) $ A AA B DBB C CE $$ $ F # ← e_t

Fig. 4. Compression boosting of FM-index: T_{bwt} is divided into *contexts* and each partition is compressed separately.

P2) Index size increases because of the overhead of block-wise storage (e.g., pointers in Huffman trees).

P3) We have to save $l\sigma$ integers for the rank results. This is unrealistic for huge σ even if $k = 1$ ($l = \sigma$).

2) Variants of CB: There are some CB variants that avoid the above problems. Fixed-block boosting [15] adopts blocks of a fixed size. Although this solves P1 (and P2 partially), problem P3 remains for huge σ. *Implicit compression boosting* (ICB) [16] avoids such explicit block partitioning by using a compressed succinct dictionary called an *RRR* [12] in the wavelet tree of T_{bwt}. This solves P1 and P3. In this paper, we consider two types of ICBs, ICB-Huff and ICB-WM. The former is ICB with an HWT, while the latter is ICB with a *wavelet matrix* [17], which is an efficient alternative to a wavelet tree. As discussed in our theoretical analysis, ICBs still suffer from large overheads when applied to a string with large alphabet, such as a trajectory string of NCTs.

III. Proposed data structure

A. Overview

For NCTs, the alphabet size σ can be millions because it is the number of road segments in a road network. As discussed in the previous section, this makes the compression of trajectory strings inefficient, because the redundant bits in wavelet trees increase as σ increases. To avoid this, we convert trajectory strings into strings with a small alphabet via *relative movement labeling* (RML), which is based on the sparsity of road networks. Figure 5 and the following give an overview of how to construct the proposed data structure, CiNCT.

1) Convert a set of NCTs into a trajectory string T.
2) Calculate the BWT of T and obtain T_{bwt}.
3) Construct an *ET-graph* G_T and a *relative movement labeling* (RML) function ϕ based on T (Section III-B)
4) Label T_{bwt} based on the RML function ϕ and obtain the *labeled BWT* $\phi(T_{bwt})$ (Section III-C).
5) Store $\phi(T_{bwt})$ in an HWT with RRR and obtain the proposed index structure (Section III-C).

As steps 1 and 2 are straightforward, we describe the details of steps 3–5 in the following sections. We emphasize that the NCTs are labeled *after* the BWT (step 4), otherwise we would be unable to implement the suffix range query. Due to this labeling step, we need to develop an algorithm that differs from Algorithm 1. Such an algorithm is described in Section IV. The theoretical consequences of CiNCT are described in Section V. Here, we focus on the index structure.

Note that CiNCT basically deals with static data. We can treat growing data by periodic reconstruction or by constructing an index for new data at certain time intervals.

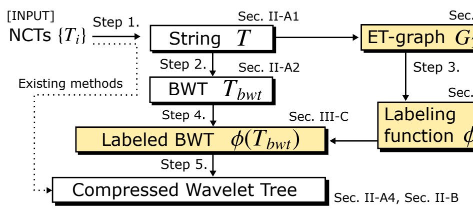
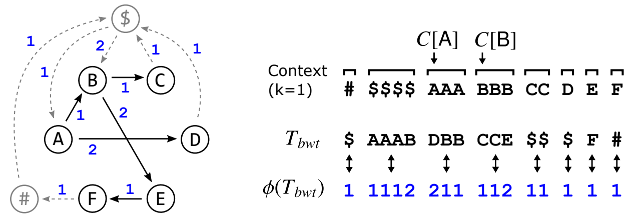


Fig. 5. Overview of CiNCT (index construction). Note that the existing methods (ICBs; dotted line) do not have the labeling step (yellow boxes).



(a) ET-graph and RML (b) Labeling T_{bwt} with RML

Fig. 6. (a) ET-graph of our example Eq. (1): each node represents a road segment, and an edge exists if the corresponding transition occurs in T. The integer on each edge is the corresponding label. (b) Relative movement labeling: $\phi(T_{bwt})$ produces a string with a lower entropy than that of T_{bwt}.

B. Relative movement labeling (RML)

The RML converts trajectory strings into strings with small alphabet based on the following fact: *NCTs can only move between physically connected road segments*. First, we describe its idea based on the example in Fig. 1 (a). If a vehicle is on a road segment $w' = \mathsf{A}$, the next segment w has to be B or D. Hence, we label them `1` and `2`, respectively. Generally, if there are k connected road segments from a certain segment, we can label them with `1`, $\cdots$, `k`. The sequences converted with this *relative movement labeling* (RML) are expected to have small alphabet because k is smaller than the maximum out-degree of the road network. To define RML formally, let us define an *empirical transition graph* (ET-graph).

Definition 4 (ET-graph): Let T be a string defined on an alphabet Σ. An *ET-graph* G_T of T is a directed graph that satisfies: *1)* the vertex set is Σ; *2)* a directed edge $(w', w) \in \Sigma \times \Sigma$ exists iff there exists a substring ww' in T. The edge set is denoted by E_T.

In other words, an edge exists iff a direct transition between w' and w exists in T. An ET-graph is a sparse graph because it has a similar topology to the original road network. Figure 6 (a) illustrates the ET-graph of the trajectory string T given in Eq. (1). Note that ET-graphs include the special symbols $ and #.

1) Definition of RML: RML can be defined as an integer assigned on each edge of the ET-graph (see Fig. 6 (a)). For example, the transition $\mathsf{A} \to \mathsf{B}$ is labeled `1`. We denote such labeling as $\phi(\mathsf{B}|\mathsf{A}) = \mathtt{1}$. The transition $\mathsf{A} \to \mathsf{D}$ must have a different label, otherwise we cannot distinguish them.

In general, for transition $w' \to w$, we denote such a labeling function by $\phi(w|w')$. To make the labeling *distinct* based on the previous symbol w', the RML function ϕ must satisfy the following requirement:

- *Requirement*: *The RML function $\phi(\cdot|w')$ must be a one-to-one map for any w'.*

Now, we discuss how to construct the RML function ϕ that satisfies the requirement above. Let us consider the *out-vertex set* of w', defined as $N_{out}(w') = \{w|(w', w) \in E_T\}$, that determines the set of vertexes directly accessible from w'. Based on the ET-graph and out-vertex set, we define $\phi(\cdot|w')$ as follows. Given w', assign a different small integer $c_{ww'}$ to each $w \in N_{out}(w')$ and define $\phi(w|w') := c_{ww'}$. It is clear that $\phi(\cdot|w')$ is a one-to-one map. If $w \notin N_{out}(w')$, we cannot define $\phi(w|w')$. However, this is not a problem because $w \notin N_{out}(w')$ indicates that the string ww' is not found in T, which tells us the result of pattern matching is null. This point is important for our search algorithm.

2) Finding an optimal RML: The RML ϕ described above does not define a unique labeling function because we have not yet specified a concrete way to assign the small integers $c_{ww'}$. Here, we propose a strategy based on a bigram count $n_{ww'}$ (i.e., the frequency of ww' in T). The elements in $N_{out}(w')$ are sorted in descending order of bigrams $n_{ww'}$. The vertex w with the largest bigram count is given the smallest label, `1`. The second-most frequent vertex is labeled `2`, the third-most frequent vertex is labeled `3`, and so on.

The labels in Fig. 6 (a) are determined in this way. For example, since we have $n_{\mathsf{BA}} > n_{\mathsf{DA}}$ ($n_{\mathsf{BA}} = 2$ and $n_{\mathsf{DA}} = 1$), the edge from A to B has the smallest label `1`: $\phi(\mathsf{B}|\mathsf{A}) = \mathtt{1}$.

In the next section, we show how to label T_{bwt} using this bigram-based RML function ϕ. One might wonder whether there exists a better labeling strategy. We prove, however, the optimality of the labeling that leads to strong conclusions: *our RML achieves the smallest size and the fastest search*. See Section V-A for details.

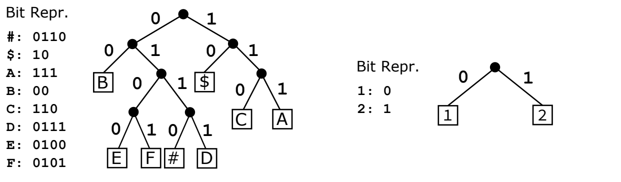
C. Data structure

Here, we describe how to obtain $\phi(T_{bwt})$ and the final index (steps 4 and 5 in Section III-A).

1) Labeling BWT (step 4): Based on the RML function ϕ obtained in the previous section, the BWT T_{bwt} is converted to $\phi(T_{bwt})$ in the following manner. For example, let us focus on the third block of T_{bwt}, DBB, in Fig. 6 (b). This block corresponds to the context of A, which indicates that the previous symbol of these DBB is A. Hence, DBB is labeled as `211` because $\phi(\mathsf{B}|\mathsf{A}) = \mathtt{1}$ and $\phi(\mathsf{D}|\mathsf{A}) = \mathtt{2}$ in Fig. 6 (a). All the other blocks also can be labeled in the same manner.

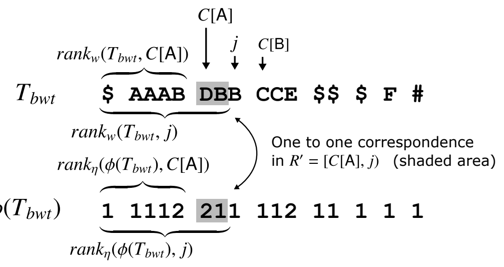
This labeling strategy generates a low-entropy sequence $\phi(T_{bwt})$ as shown in Fig. 6 (b), because the distribution of the resulting symbols is biased toward smaller integers (i.e., `1` is the largest fraction). For this example, we have $H_0(T_{bwt}) = 2.8$ and $H_0(\phi(T_{bwt})) = 0.7$ (unit: bits/symbol).

2) Storing to a compressed wavelet tree (step 5): In this step, we store the labeled BWT $\phi(T_{bwt})$ to an HWT. For bit vectors in an HWT, we adopt a practical version of the compressed succinct dictionary called RRR [18]. This is a straightforward step. Figure 7 depicts the comparison of Huffman trees of T_{bwt} and $\phi(T_{bwt})$ for the example in Fig. 6 (b). The Huffman tree of $\phi(T_{bwt})$ is obviously simpler than that of T_{bwt}. Because these tree shapes are the same as those



(a) T_{bwt} (b) $\phi(T_{bwt})$: CiNCT

Fig. 7. Huffman trees of T_{bwt} and $\phi(T_{bwt})$: each leaf corresponds to a symbol. The tree generated by CiNCT (b) is simpler than that from the existing technique (a).



C[A]
j
C[B]
$rank_w(T_{bwt}, C[\mathsf{A}])$
T_{bwt}
$ AAAB DBB CCE $$ $ F #
$rank_w(T_{bwt}, j)$
$rank_\eta(\phi(T_{bwt}), C[\mathsf{A}])$
One to one correspondence in $R' = [C[\mathsf{A}], j)$ (shaded area)
$\phi(T_{bwt})$
1 1112 211 112 11 1 1 1
$rank_\eta(\phi(T_{bwt}), j)$

Fig. 8. Basis of the balancing equation (Eq. (5)) for *PseudoRank*

of HWTs, this simplification explains intuitively why CiNCT is small and fast. For more details, see Section V.

An RRR bit vector has one parameter b, that controls the size of the internal blocks. For larger b, we obtain better compression but slower search ($rank$ calculation) in general, and *vice versa*. This b is the only parameter in CiNCT. However, in Section VI, we show that this parameter has only a small influence on the index size and the search time.

3) Storing ET-graph: We use an adjacency list to represent the ET-graph G_T. The value $\phi(w|w')$ is assigned to the edge $(w', w) \in E_T$. Thus $\phi(w|w')$ is obtained in $O(\delta)$ time by a linear search over $N_{out}(w')$. We also assign $C[w]$ to each vertex w in G_T. *Correction terms* $Z_{w'w}$, introduced in Section IV-A, are also attached to each edge. Since G_T is sparse, the space needed to store G_T is negligible when $|T|$ gets large. Note that ET-graphs can be implemented using *succinct graphs* (e.g., [19]). In this paper, however, we do not use them because their impact on the data size was small in our preliminary experiments.

IV. Proposed query processing algorithms

Here, we describe another key concept of this paper, *PseudoRank*, then show algorithms for two types of queries, suffix range queries and sub-path extraction queries.

A. PseudoRank

As noted in Sec. II-A3, fast calculation of $rank_w(T_{bwt}, j)$ is needed for Algorithm 1. The original FM-index stores T_{bwt} in a wavelet tree to calculate ranks quickly. In our case, however, we do not have the original T_{bwt} but only have the labeled $\phi(T_{bwt})$. Can we obtain the rank values for the original BWT by using only the labeled BWT? Seemingly, this is difficult because different symbols are mapped to the same label (e.g., both A and C are converted to 1 as illustrated in Fig. 6 (b)).

The key idea in CiNCT is to simulate the rank operation over T_{bwt}. Figure 8 illustrates this idea. Let us consider the range $R(\mathsf{A}) = [C[\mathsf{A}], C[\mathsf{B}])$ and $j \in R(\mathsf{A})$. Because the substring $T_{bwt}[C[\mathsf{A}], C[\mathsf{B}]) = \mathsf{DBB}$ is labeled as 211 by using the one-to-one map $\phi(\cdot|\mathsf{A})$ as described in Section III-C, the following two counts are equivalent for $\forall j \in R(\mathsf{A})$:

- the number of occurrences of D within the range $R' := [C[\mathsf{A}], j)$ in T_{bwt} (the shaded region in Fig. 8), and
- the number of occurrences of 2 within R' in $\phi(T_{bwt})$.

This balancing relationship holds in general. Let us consider a context w'. For all j such that $C[w'] \le j \le C[w'+1]$, let us consider a range $R' := [C[w'], j)$. For a symbol $w \in N_{out}(w')$, the number of occurrences w within R' in T_{bwt} and that of the label $\eta := \phi(w|w')$ within R' in $\phi(T_{bwt})$ are the same because of the one-to-one requirement for $\phi(\cdot|w')$. This leads to the following balancing equation:

$$\begin{aligned} &rank_w(T_{bwt}, j) - rank_w(T_{bwt}, C[w']) \\ &\quad = rank_\eta(\phi(T_{bwt}), j) - rank_\eta(\phi(T_{bwt}), C[w']). \end{aligned} \tag{5}$$

Rearranging this equation, we have the following theorem, which allows us to simulate the rank operation.

Theorem 2 (Pseudo-rank): If $w \in N_{out}(w')$ and $C[w'] \le j \le C[w'+1]$, then we have

$$rank_w(T_{bwt}, j) = rank_\eta(\phi(T_{bwt}), j) - Z_{w'w}, \tag{6}$$

where $\eta := \phi(w|w')$ and

$$Z_{w'w} := rank_\eta(\phi(T_{bwt}), C[w']) - rank_w(T_{bwt}, C[w']). \tag{7}$$

We emphasize that the correction term $Z_{w'w}$ does not depend on j, implying that the number of correction terms needed is equal to $|E_T|$. Importantly, this property allows us to precompute and store the correction terms (as noted in Section III-C, they are attached to each edge $(w', w) \in E_T$).

This theorem produces Algorithm 2, which calculates the rank values using only $\phi(T_{bwt})$. We also emphasize that *PseudoRank* does not allow us to calculate rank values for all pairs of (w, j). However, this limitation is not a problem for our search algorithm, as shown in the next subsection.

Algorithm 2: Emulating $rank_w(T_{bwt}, j)$ by using only $\phi(T_{bwt})$ ($PseudoRank(\phi(T_{bwt}), j, w, w', Z_{w'w})$)

```
Input: Labeled BWT string of length $n$: $\phi(T_{bwt})$,
       Location of rank $j$,   Correction term $Z_{w'w}$,
       Target symbol $w$,   Previous symbol $w'$
Output: The value of $rank_w(T_{bwt}, j)$
1 if $w \in N_{out}(w')$ and $C[w'] \le j \le C[w'+1]$ then
2     $\eta \leftarrow \phi(w|w')$                    // RML
3     return $rank_\eta(\phi(T_{bwt}), j) - Z_{w'w}$
4 return NotFound
```

B. Suffix range query with CiNCT

With the *PseudoRank*, we can simulate $rank_w(T_{bwt}, j)$ using only the wavelet tree of $\phi(T_{bwt})$ and the correction term $Z_{w'w}$ (Eq. (7)). Replacing the rank operations in Algorithm 1 with *PseudoRank*, we obtain our search algorithm (Algorithm 3), whose correctness is shown below.

Algorithm 3: Finding the suffix range $[sp, ep)$ for a given query P of length m based on $\phi(T_{bwt})$ (*LabeledSearchFM*)

```
Input: Labeled BWT string of length $n$: $\phi(T_{bwt})$,
       Query of length $m$: $P[0, m)$,
       Correction terms: $\{Z_{w'w}\}$
Output: Range of $T_{bwt}$ that matches to $P$
1  $w \leftarrow P[m-1]$; $sp \leftarrow C[w]$; $ep \leftarrow C[w+1]$
2  for $i \leftarrow 2$ to $m$ do
3      $w' \leftarrow w$   // Save the previous symbol
4      $w \leftarrow P[m-i]$
5      if $w \notin N_{out}(w')$ then
6          return NotFound
7      $sp \leftarrow C[w] + PseudoRank(\phi(T_{bwt}), sp, w, w', Z_{w'w})$
8      $ep \leftarrow C[w] + PseudoRank(\phi(T_{bwt}), ep, w, w', Z_{w'w})$
9      if $sp \geq ep$ then
10         return NotFound
11 return $[sp, ep)$
```

1) Correctness of the algorithm: To guarantee that Algorithm 3 is equivalent to Algorithm 1, we have to check the following two conditions on *PseudoRank* (Theorem 2) are satisfied immediately before Line 7: (c1) $w \in N_{out}(w')$; (c2) $C[w'] \leq sp \leq C[w'+1]$ and $C[w'] \leq ep \leq C[w'+1]$.

As noted previously, no substring ww' appears in T if $w \notin N_{out}(w')$; hence, NotFound is returned if $w \notin N_{out}(w')$ at Line 6. Therefore, (c1) $w \in N_{out}(w')$ holds immediately before Line 7. For (c2), before Line 7, sp satisfies

$$sp = C[w'] + rank_{w'}(T_{bwt}, sp'), \tag{8}$$

where sp' is the previous value. By the *rank* definition, $0 \leq rank_{w'}(T_{bwt}, j) \leq C[w'+1] - C[w'] (0 \leq \forall j < |T|)$ holds, where $C[w'+1] - C[w']$ means the number of occurrences of w' in T. Combining this inequality with Eq. (8), we obtain $C[w'] \leq sp \leq C[w'+1]$. We can prove the condition for ep in a similar manner.

C. Extracting a sub-path with CiNCT

Here, we describe another important query, *sub-path extraction query*. For example, let us focus on the *third* sorted rotation in Fig. 2. Its suffix of length four is FEBA (colored in blue). This corresponds to the example NCT T_1^r in Eq. (1). In this way, the sub-path extraction queries recover a sub-path of length l from an arbitrary position j in BWT T_{bwt} ($j = 3$ for the example above). This query is useful if we need to obtain certain NCTs stored in BWT string, or we need to recover the entire trajectory string. Formally, *extract*(j, l) returns $T[i-l, i)$ where $i = SA[j]$ (SA is the suffix array of T). The subscript j is often referred to as an *inverse suffix array* ($j = ISA[i]$).

Algorithm 4 shows how to obtain *extract*(j, l) using only $\phi(T_{bwt})$ and the ET-graph. This is obtained by mimicking *LF-mapping* [8] with PseudoRank. Line 1 performs a binary search to find the last character $T[i] = w'$ such that $C[w'] \leq j < C[w'+1]$. Line 4 first accesses the j-th character of $\phi(T_{bwt})$ (i.e., the labeled $T_{bwt}[j]$), then decodes the $T_{bwt}[j] = T[i-k-1] = w$ using the ET-graph. Line 5 is similar to Line 7 in Algorithm 3, which jumps to the next position on T_{bwt} (LF-mapping simulated by PseudoRank).

Algorithm 4: Extracting a sub-path $T[i-l, i)$ for given $j = ISA[i]$ and $l > 0$ (*extract*)

```
Input: Labeled BWT: $\phi(T_{bwt})$, Position on $T_{bwt}$: $j$,
Extraction length: $l$, Correction terms: $\{Z_{w'w}\}$
Output: A substring $S := T[i-l, i)$
1  $w' \leftarrow BinarySearch(j, \{C[w']\})$          // T[i]
2  for $k \leftarrow 1$ to $l$ do
3      $\eta \leftarrow \phi(T_{bwt})[j]$; $w \leftarrow decode(\eta|w')$; $S[l-k] \leftarrow w$
4      $j \leftarrow C[w] + PseudoRank(\phi(T_{bwt}), j, w, w', Z_{w'w})$
5      $w' \leftarrow w$        // Save previous symbol
6  return $S$
```

D. Linking time information to CiNCT

Here, we describe how to link trajectory IDs and timestamps to CiNCT. In document retrieval, Sadakane [20] proposed a method to list document (trajetory) IDs that match a pattern P, using additional $4|T| + o(|T|)$ bits. Once we obtain the trajectory IDs, we can access to each corresponding timestamps compressed by existing methods [1], [4], [5] (see Sec. VII-2).

Other linking methods are found in [3], [6]. Essentially, these methods link a suffix array for the trajectory string and the timestamp information using an *inverse suffix array* of the trajectory string T. Based on this idea, SNT-index [6] links FM-index and B$^+$-tree for timestamps and trajectory IDs. This enables an advanced query called *strict path query*, which finds the trajectory IDs that traveled along a given sub-path P during a time interval I (see Sec. VII-1).

V. Theoretical analysis

Here, we explain theoretically why CiNCT is compact and fast. We first show the optimality of RML, that is, the labeled BWT $\phi(T_{bwt})$ achieves the smallest entropy. Then, we explain that such a small entropy contributes high compressibility and fast query processing. We also show that RML is better than other labeling method called MEL, recently proposed in [1].

A. Optimality of RML

The 0th order entropy H_0 given in Eq. (3) plays important roles in our analysis. First, we show the labeling strategy based on bigram counts $n_{ww'}$ proposed in Section III-B achieves the minimum value of H_0 among all possible labelings.

Theorem 3 (Optimality): Let ϕ^* be the RML based on the bigram ordering strategy and ϕ be any possible RML that satisfies the requirement in Section III-B. Then, we have

$$H_0(\phi^*(T_{bwt})) \leq H_0(\phi(T_{bwt})). \tag{9}$$

Proof: Due to space limitations, we can only provide a sketch of our proof here (see our long version [21] for details). Let ϕ^* be a labeling function that achieves the minimum entropy. Without loss of generality, a label $k \in \mathbb{N}$

is the kth most frequent label in $\phi^*(T_{bwt})$. Assume that the optimal labeling ϕ^* is not completely sorted in the bigram counts $n_{ww'}$. Considering a new labeling $\tilde{\phi}$ that fixes one of such disordered parts, a simple calculation reveals that the new sequence labeled with $\tilde{\phi}$ achieves smaller entropy than $H_0(\phi^*(T_{bwt}))$. This contradicts the optimality of ϕ^*. ■

As a special case of this theorem, we obtain an *unlabeled case* result, i.e., $H_0(\phi^*(T_{bwt})) \leq H_0(T_{bwt})$, by putting as $\phi = id$ (identity labeling). Importantly, we see that

$$H_0(\phi^*(T_{bwt})) \ll H_0(T_{bwt}) \tag{10}$$

holds for real NCT datasets in our experiments (Table III).

B. Compressed size

1) Evaluating space overheads: The data structure of CiNCT consists of two parts: the labeled BWT $\phi(T_{bwt})$ and the ET-graph G_T. As noted in Section III-C, the size of G_T is negligible when $|T|$ is large. Here, we compare the sizes of T_{bwt} and $\phi(T_{bwt})$ stored in HWTs with RRR. Note that these corresponds ICB-Huff and CiNCT, respectively. The main advantage of CiNCT comes from the lower space overhead due to RRR, as explained below. For a given bit vector B, it is known that the practical RRR with the parameter b (see Sec. III-C2) uses at most

$$|B|H_0(B) + |B| \cdot h(b) \tag{11}$$

bits where $h(b) = \frac{\lg(b+1)}{b}$ [18]. We call the second term the *RRR-overhead*. For $b = 63$, we have an overhead of $h(b) = (\lg 64)/63 \simeq 0.095$ bits per bit.

For a given string S, it is known that the average code length with Huffman coding is at most $(1 + H_0(S))$ bits [13]. Hence, the total length of bit vectors in the HWT is $\sum_v |B_v| \simeq |S|(1+H_0(S))$. Summing the RRR-overheads over all internal nodes v in the HWT, we obtain total bits of the overhead:

$$\sum\nolimits_v |B_v| \cdot h(b) \simeq |S|(1 + H_0(S)) \cdot h(b). \tag{12}$$

The right-hand side implies that the RRR-overhead of a sequence S is small if its entropy $H_0(S)$ is small. Therefore, Eq. (10) indicates that the space overhead for CiNCT is much smaller than that for ICB-Huff.

2) High-order compression: Here, we analyze the remaining first (and dominant) term in Eq. (11). Summing this term over all internal nodes v in the HWT, we find that the total bits needed for this term achieves the k-th order entropy Eq. (4) for all $k > 0$, as shown in the following Theorem 4. This theorem implies that our method guarantees high compressibility in an information theoretic sense. Note that this kind of entropic bound has not been guaranteed by the existing shortest-path based NCT compressors.

Theorem 4: For all $k > 0$, the total bits required to store $\phi(T_{bwt})$ in an HWT with RRR, apart from the overhead Eq.(12), are $|T|H_k(T) + O(l\sigma b)$, where $l \leq \sigma^k$ is the number of distinct contexts $W \in \Sigma^k$ in T.

This is proved in a similar way to the existing compression boosters [16], but we omit the proof here due to the space limitations. See our long version for the proof [21]. ■

C. Processing time of suffix range queries

To evaluate whether Algorithm 3 is faster than Algorithm 1, we focus on the time complexity of the rank operation. As stated in Theorem 1 (Sec. II-A4), $rank_w(S, j)$ runs in $O(1 + H_0(S))$ time.[1] Hence, the relationship $H_0(\phi(T_{bwt})) \ll H_0(T_{bwt})$ (Eq. (10)) again explains why CiNCT is faster than ICB-Huff. Of course, Algorithm 3 incurs an additional cost in calculating $\phi(w|w')$, but this is not serious for a sparse G_T.

Moreover, we have the following theorem implying that the search time does not depend on the road network size σ but depends only on the maximum out-degree δ of the road network (which is usually less than four).

Theorem 5 (σ-independence): Let $P \in E^*$ be any query path (\$ is not included). Algorithm 3 runs in $O(|P| \cdot \delta b)$ time.

Proof: For any $w, w' \in E$, we have $\eta := \phi(w|w') \leq \delta+2$. By the construction of RML, η is at least the $\delta + 2$-th most frequent symbol in $\phi(T_{bwt})$. Thus η is at most located at the $\delta + 2$ level of the Huffman tree. Hence, $rank_\eta(\phi(T_{bwt}), j)$ in Eq. (6) runs in $O(\delta b)$ time (remember the bit-wise rank operation in practical RRR [18] requires $O(b)$ time). Since *PseudoRank* is calculated at most $2|P| - 2$ times in Algorithm 3, this leads to the conclusion. ■

Other FM-indexes do not satisfy this property. Note that this time complexity also does not depend on the data size $|T|$.

D. Comparison of RML with MEL

Minimum entropy labeling (MEL) is a labeling scheme for NCTs recently proposed in [1], which works as a preprocessor for general compressors, such as the Huffman coding or the LZ coding (i.e., pattern matching was not considered). Similar to RML, MEL converts a sequence of road edges to a low entropy sequence of small integers as follows: $w_1w_2 \cdots w_n \to \psi(w_1)\psi(w_2) \cdots \psi(w_n)$ where $\psi : E \to \mathbb{N}$ is the MEL function. Different labels are assigned to road segments that shares a head node v (Fig. 9(b)). In contrast, our RML conversion is as follows: $w_1w_2 \cdots w_n \to \phi(w_1|\$)\phi(w_2|w_1) \cdots \phi(w_n|w_{n-1})$. Unlike RML, the MEL function ψ does not consider the previous symbol. Specifically, As in Fig. 9(b), MEL labels based on the *unigram* frequencies, n_{A} and n_{B}. Conversely, our RML, shown in Fig. 9(a), is based on the bigram frequencies, n_{XA}, n_{XB}, n_{YA}, and n_{YB}.

Given these differences, the advantage of RML can be intuitively explained as follows. Real trajectories tend to go straight rather than turn left or right, as shown in Fig. 9 (a). Because RML considers the previous road segment, it can take account the direction of the movement, whereas such information is lost in MEL. This implies that RML can capture a higher-order correlation compared to MEL. Although MEL also has the optimality of entropy, it cannot be better than RML. The experimental comparison is shown in Section VI-D. Mathematically, we have the following theorem.

Theorem 6: For any trajectory string T, RML achieves a smaller 0th order empirical entropy than MEL does.

[1]To be exact, this complexity is proportional to b because practical RRR [18] runs the bit-wise rank operation in $O(b)$ time.

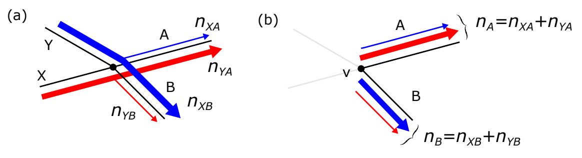


Fig. 9. Comparing two NCT labeling methods: (a) RML; and (b) MEL [1]

Proof: Considering the size of the feasible labeling space, we find that our labeling space $\{\phi(w|w')\}$ is a superset of that of MEL, $\{\psi(w)\}$. In other words, MEL can be emulated by an RML $\bar{\phi}$ that might not be the optimal ϕ^*. Therefore, the optimality of RML (Theorem 3) leads to the conclusion. ∎

VI. Experiments

A. Experimental setup

1) Implementation: All methods were implemented in C++ and compiled with g++ (version 4.8.4) with the `-O3` option. We used the `sdsl-lite` library (version 2.0.1) for (in-memory) wavelet trees (http://github.com/simongog/sdsl-lite/). The BWT was calculated using `sais.hxx` (http://sites.google.com/site/yuta256/sais/). Experiments were conducted on a workstation with the following specifications: Intel Core i7-K5930 3.5GHz CPU (64-bit, 12 cores, L1 64kB×12, L2 256kB×12, L3 15MB), DDR4 32GB RAM, Ubuntu Linux 14.04.

2) Competitors: Table II lists the competitors used in this paper. We used five FM-index variants: uncompressed (UFMI, FM-GMR) and compressed (ICB-WM, ICB-Huff, FM-AP-HYB). The block-size parameter b had to be specified for CiNCT, ICB-Huff, and ICB-WM. Unless otherwise noted, we use $b = 63$. FM-GMR [22] and FM-AP-HYB [23] are FM-index variants that are tailored for huge σ and that support $O(\log\log\sigma)$ rank operation (faster than the $O(\log\sigma)$ of UFMI); they are available in the `sdsl-lite` library. These were the fastest (FM-GMR) and the smallest (FM-AP-HYB) methods for huge σ in a recent benchmark [24].

There are many possibilities for compressing NCTs by combining simple techniques such as run-length encoding. However, we do not consider such techniques in this study because pattern matching is not supported in sublinear time. In our prior evaluation, the Boyer-Moore method (linear time search) was at least four orders of magnitude slower than CiNCT. In this study, we thus only consider RePair [25], a standard benchmark in stringology which showed the best compression ratio in the initial evaluation, and PRESS [26], a shortest-path-based NCT compressor, and MEL [1], state-of-the-art labeling-based NCT compressor. Note that the existing NCT indexing methods, such as MON-tree [27], are not in our competitors because they focus on neither suffix range queries nor compression (as will be noted in Section VII, these methods are *enhanced by* suffix range queries).

3) Measurement: The search time was averaged over 500 suffix range queries of length 20 randomly sampled from data. For evaluation of the data size, the ET-graph was included.

4) Datasets: The datasets used in this study are as follows:

- Singapore: NCTs of taxi cabs used in [26]. This dataset contains many transitions without physical connection.

TABLE II
Our proposed method and its competitors*

Method	Data	Description	C?†	Q?‡
CiNCT	$\phi(T_{bwt})$	HWT with RRR	✓	✓
UFMI	T_{bwt}	WM$^\diamond$ [17] with uncompressed bitmap [11]		✓
ICB-WM	T_{bwt}	WM with RRR [17]	✓	✓
ICB-Huff	T_{bwt}	HWT with RRR [16]	✓	✓
FM-GMR	T_{bwt}	FM-index for huge σ with $O(\log\log\sigma)$ rank [22]		✓
FM-AP-HYB	T_{bwt}	FM-index for huge σ with $O(\log\log\sigma)$ rank [23]	✓	✓
PRESS	T	The state-of-the-art trajectory compressor [26]	✓	
MEL	T	Min. entropy labeling [1]	✓	
Re-Pair††	T	A string compressor [25]	✓	

* For the first four methods, the type of WT is described / $^\diamond$ WM: wavelet matrix
† Uncompressed or compressed / ‡ Supports suffix range query or not
†† We used an implementation at https://www.dcc.uchile.cl/~gnavarro/software/

TABLE III
Statistics of each dataset

Dataset	$\|T\|$	$\lg\sigma$	$H_0(T)$	$H_0(\phi)^\dagger$	$H_1(T)$	$\bar{d}^{\,\ddagger}$
Singapore	53M	15.5	13.8	1.8	1.5	26.8
Singapore-2	75M	15.5	14.0	1.3	1.1	4.0
Roma	12M	15.5	13.0	0.9	0.7	2.4
MO-Gen	193M	17.4	13.0	2.8	2.5	8.8
Chess	20M	18.8	10.3	2.0	1.4	1.6

† $H_0(\phi)$ means $H_0(\phi(T_{bwt}))$
‡ $\bar{d}$ is the average out-degree of the ET-graph G_T.

- Singapore-2: Preprocessed Singapore dataset such that transitions between two road segments without a physical connection are interpolated with the shortest path.
- Roma: GPS trajectories of taxi cabs in Rome. NCT representations were obtained by HMM map-matching [28] (http://crawdad.org/roma/taxi/).
- MO-gen: NCTs generated by the moving object generator (http://iapg.jade-hs.de/personen/brinkhoff/generator/).
- Chess: All chess game records (Blitz, 2006–2015, 1.87M games, http://www.ficsgames.org). First 10 moves are converted into hash values of Forsyth-Edwards notation.

Although Chess is not a vehicular dataset, it is included to show the possibility that CiNCT is applicable to targets other than NCTs. Table III lists the statistics of the datasets.

B. Comparison with various FM-indexes

Evaluation results for data size and processing time of suffix range queries are shown in Fig. 10. We observe that CiNCT requires *less than 2 bits per symbol* to store NCTs, and pattern matching of length 20 is processed in *a few tens of microseconds*. We also observe that *CiNCT outperforms the competitors in terms of both data size and query processing time.* We explain these results in detail below.

1) Data size: Compared with ICB-Huff and ICB-WM, CiNCT reduces the data size by up to 78% and 57%, respectively. As explained in Section V, the space overhead decreases if $H_0(S)$ decreases. From Table III we can confirm that $H_0(\phi(T_{bwt})) \ll H_0(T_{bwt})$ holds for all datasets (note that $H_0(T) = H_0(T_{bwt})$). This explains why CiNCT shows this significant improvement.CiNCT even shows better com-

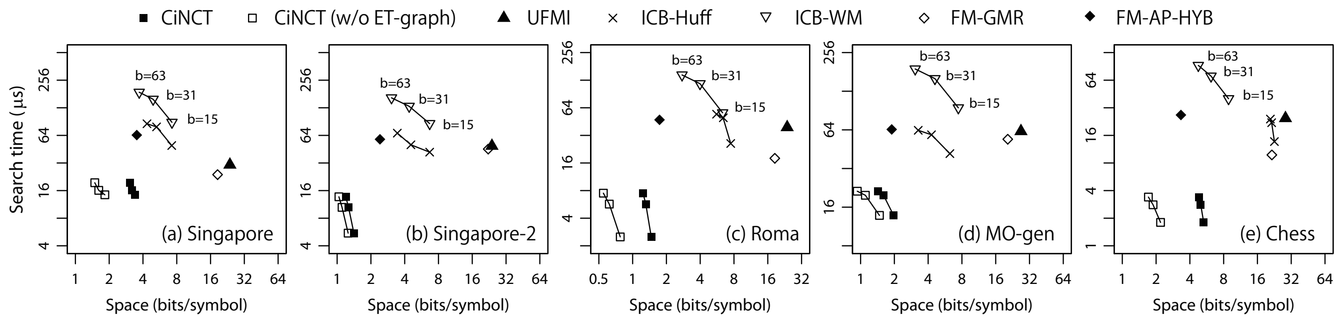


Fig. 10. Data size/search time (suffix range query): the proposed method shows the best performance. CiNCT (w/o ET-graph) is used to show the data size without the ET-graph (i.e., wavelet tree only). The block size used in the RRR bit vectors is parameterized as $b \in \{15, 31, 63\}$. Results for the other methods in Table II were omitted because their linear-time search was too slow.

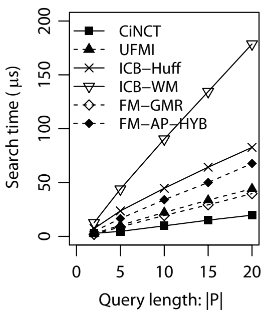


Fig. 11. $|P|$ vs. search time: (Singapore dataset)

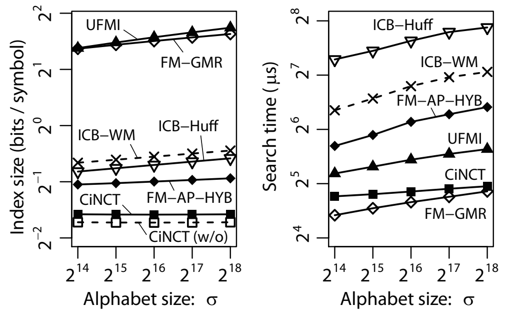


Fig. 12. CiNCT shows the best σ-dependence (Left: index size, right: search time / RandWalk dataset)

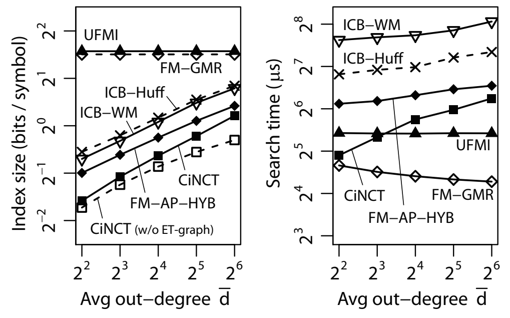


Fig. 13. Dependence on out-degree (Left: index size, right: search time / RandWalk dataset)

pression than the smallest variant FM-AP-HYB, which was designed for huge σ. The improvement in Singapore-2 is larger than that of Singapore. Because "gapped" transitions are interpolated in Singapore-2, the ET-graph gets sparser ($\bar{d} = 26.8 \rightarrow 4$ in Table III). This reduces the overhead regarding the ET-graph (this is confirmed through the difference of CiNCT and CiNCT (w/o ET-graph); w/o stands for without).

2) Processing time of suffix range queries: CiNCT is always much faster than ICB-Huff and ICB-WM; the speedups are up to 7 and 25 times, respectively. Surprisingly, CiNCT is even faster than those of the uncompressed indexes (UFMI and FM-GMR). Again, this speedup can be explained by the shallowness of the HWT of CiNCT. This decreases the number of bit-wise *rank* operations in the HWT (Section V-C).

3) Effect of block size b: As mentioned in Section III-C, as b becomes larger, the results show better compression but slower search. However, as shown in Fig. 10, *the sensitivity to the block size parameter b is very small for CiNCT*. This indicates that the proposed method is nearly *parameter-free*.

4) Effect of $|P|$: Figure 11 shows the processing time of suffix range queries against the query length $|P|$. For all methods, the processing time grows linearly, because the numbers of iterations in Algorithms 1 and 3 are $|P|$. We see that *CiNCT shows the slowest growth among all methods.*

C. Comparison with several compression methods

Table IV compares the compression ratio, defined as the uncompressed size (binary file of 32-bit integers) divided by the compressed size. We observe that CiNCT shows better compression than the existing methods. In particular, *our method is better than MEL, which showed the best compressibility in recent benchmark* [1]. This is explained as follows. First, as shown in Theorem 6 (and Table V in the next section), RML achieves smaller 0th order entropy than MEL (indicating a smaller average code length). Second, CiNCT is a higher order compressor (Theorem 4) whereas MEL is not. Note that the road network storage is not included in MEL evaluations whereas it is considered for CiNCT (as ET-graph).

TABLE IV
COMPRESSION RATIO (LARGER IS BETTER)

	Singapore	Singapore-2	Roma	Mo-Gen	Chess
CiNCT	**10.5**	**27.0**	**25.2**	**25.6**	10.3
MEL[†]	N/A	15.8	21.2	N/A	N/A
Re-Pair	8.4	11.4	20.6	20.6	**11.0**
ICB-WM	8.8	9.4	11.3	12.0	10.5
bzip2	5.3	5.6	13.6	5.3	7.1
PRESS[‡]	4.6	N/A	N/A	N/A	N/A
zip	2.5	2.5	5.0	2.6	3.9

[†] Huffman coding was used after labeling, as in [1]. We evaluated only for ungapped datasets because MEL assumes no gap (see Singapore-2 explanation in Sec.VI-A4).
[‡] Only the result for the Singapore dataset [26] is shown because no available implementation was found.

TABLE V
COMPARISON OF ENTROPY (RML AND MEL)

Dataset	RML (Proposed)	MEL [1]
Singapore2	**1.26**	1.93
Roma	**0.76**	0.99

D. Effect of labeling strategy

1) Comparison with MEL: According to our analysis in Section V-D, RML achieves lower entropy than MEL does. In Table V, we show a comparison of the entropy achieved

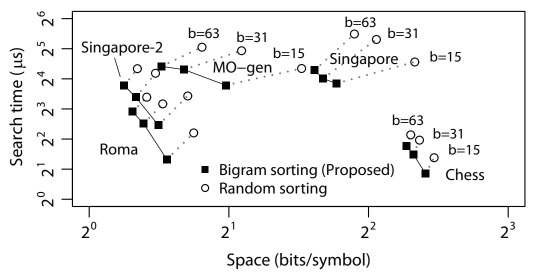


Fig. 14. Comparison of labeling strategies

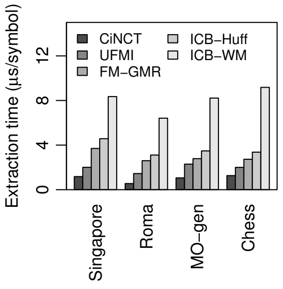


Fig. 15. Extraction time

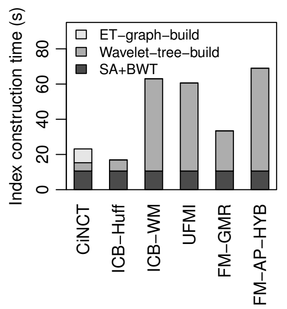


Fig. 16. Index construction time (Singapore dataset)

by RML and MEL for two "ungapped" NCT datasets, i.e., Singapore2 and Roma. We observe that *our RML obtained approximately 30% smaller entropy than that of MEL.*

2) Optimality: In Section III-B, we proposed a labeling strategy that assigns small integers $c_{ww'}$ sorted by the bigram counts $n_{ww'}$. The data size and search time under this strategy are expected to be better than those of any other possible labeling strategy, because we showed the optimality of our strategy (Theorem 3). Here, we compare our strategy with the *random sorting strategy*, which assigns randomly shuffled small integers $c_{ww'} \in \{1, \cdots, |N_{out}(w')|\}$. Figure 14 shows the comparison for the five datasets ($b \in \{15, 31, 63\}$). We observe that *the index size and the search time of the bigram sorting strategy are always better than those of random sorting strategy.* Compared to the random strategy, it reduces the data size by up to 32%, and the search time by up to 57%. These results indicate the importance of the bigram sorting strategy.

E. Effect of ET-graph size/shape

1) Effect of σ: In Theorem 5, we showed that the search time of CiNCT does not depend on the size σ of the road map. Here, we investigate the effect of σ using synthetic RandWalk dataset: random walks on a directed random graph. The average out-degree $\bar{d}$ of the graphs is fixed at four, and $|T|$ is set to 800σ. In Fig. 12, *CiNCT shows good scalability against σ, whereas the index sizes and the search times of the existing methods both increase.* The search time of CiNCT is almost constant, as predicted by Theorem 5. The other methods do not show this property. For example, both the index size and the search time of UFMI at $\sigma = 2^{18}$ are 30% larger compared to the $\sigma = 2^{14}$ case.

2) Effect of sparsity: Here, we investigate the effect of the average out-degree $\bar{d}$. Figure 13 shows the results for the RandWalk dataset used in Section VI-E. For comparison, we fixed $\sigma = 2^{16}$ and $|T| = 100$M, and changed $\bar{d}$ between 2^2 and 2^7. We observe that *the sparsity of the ET-graph is the key factor for CiNCT.* Although the compression performance of CiNCT is the best, the data size grows quickly. This is due to two factors: the increase of ET-graph size and the increase of the depth of HWT. However, this result shows that our method works for larger $\bar{d}$ than in the road network case, $\bar{d} \simeq 2^2$. This result opens the door to applications to datasets not mentioned in this paper (e.g., symbol-valued time series).

F. Sub-path extraction time

Here, we evaluate *extract* queries described in Section IV-C. We evaluated the extraction time for obtaining the entire T, that is, $l = |T|$ and $j = 0$. Figure 15 compares the extraction times for the four datasets. We observe that *CiNCT shows the fastest extraction among the competitors* (twice as fast as UFMI). Again, this can be explained by the fast *rank* calculation in CiNCT (*PseudoRank*), as discussed above. Note that we omitted the results for FM-AP-HYB because random access to T_{bwt} was not supported in the `sdsl-lite` library.

G. Index construction time

Figure 16 compares the index construction times of FM-indexes. *The construction time of CiNCT is comparable to that of ICB-Huff, and shorter than those of the other methods.* ET-graph-build in Fig. 16 includes all operations that are not needed for the other methods. Here, we can see the overhead for the construction of the ET-graph is not a serious problem. Note that all additional operations, including the construction of G_T from T, obtaining RML function ϕ, labeling T_{bwt}, and calculation of $Z_{w'w}$, can be executed in linear time $O(|T|)$, which implies the scalability of construction.

VII. Related work

1) Trajectory indexing: One important application of our method is trajectory indexing. Although there are numerous studies on this topic as shown in a survey [29], we present only the most relevant ones here. MON-tree [27] is one of the most famous methods to index NCTs. This method, as well as many of other NCT indexing methods, mainly focuses on spatio-temporal range queries. Krogh et al. [30] proposed a data structure similar to MON-tree designed for a different type of query called *strict path query* (Sec. IV-D). Koide et al. [6] showed that strict path queries can be efficiently processed using suffix range queries. This is a hybrid data structure that indexes timestamps and spatial paths using MON-tree-like B$^+$-trees and an additional FM-index, respectively. Brisaboa et al. [3] also employed a compressed suffix array to store spatial paths while a wavelet matrix was used for timestamps. The existing compressed suffix arrays used in these methods [3], [6] can be replaced with CiNCT.

2) Trajectory compression: As noted in Section I, shortest-path encodings have been used to compress spatial paths in several papers [1], [2], [4] and [26]. Although an NCT dataset is expected to have a small k-th order entropy (Eq. (4)), none of such shortest-path-based compressors have provided an information-theoretic evaluation of the compressed size. As an NCT compressor, our method first focuses on high-order entropy and gives an information-theoretic bound (Theorem 4).

One of the methods proposed in [1], MEL, is a different type of spatial path compressor that achieves better compression than shortest-path-based methods. As shown in Section V-D, RML achieves a smaller entropy than MEL does. In [5], graph partitioning was used to reduce the size of spatial paths.

To compress timestamps in NCTs, lossy compression methods are used in [1], [4] and [5], whereas lossless compression was used in [3]. In Sec. IV-D, we have shown a method to link these compressed timestamps to CiNCT through document listing queries using small additional storage [20].

3) FM-index: FM-index, a compressed representation of suffix arrays [7], was proposed by Ferragina and Manzini [8]. We have already described FM-index and the related topics in Section II. Although there are a number of FM-index variants (e.g., [15], [16], [22], [23]), these are essentially designed for general strings. In Section VI, we compared our method also with FM-indexes designed for a large alphabet [22], [23]. For large σ, these methods can process suffix range queries in $O(|P| \log \log \sigma)$ time, which is much faster than typical $O(|P| \log \sigma)$ time. Importantly, we employed the domain-specific knowledge of the target data (i.e., sparse transition in road networks) to enhance the compression and query processing. This point is the largest difference from the FM-index family designed for general strings.

Recently, the *Wheeler graph* was defined to provide a unified view on BWT-related methods [31]. Similar to the ET-graph, this is also an edge-labeled graph. The differences are; 1) In the Wheeler graph, all edges *entering* a given node must have the *same* label, while it is not necessary in the ET-graph. 2) In the ET-graph, the edges *leaving* a given node must have *different* labels. It is not necessary in the Wheeler graph.

VIII. Conclusion

In this paper, we proposed CiNCT, a novel compressed data structure for NCTs. We incorporated the sparsity of road networks into the FM-index by using our proposed RML and *PseudoRank* techniques. The resulting data structure supports pattern matching (i.e., suffix range queries) and sub-path extraction from an arbitrary position while still achieving high compressibility. Our experiments showed that CiNCT outperformed existing methods in terms of index size and search time (Fig. 10, Table IV, and Fig. 12). Our method was even faster than an uncompressed index. We also discussed theoretically why CiNCT is compact and fast. Further, we proved the optimality of RML, i.e., the smallest size and the fastest search are achieved. Our method has wide applications where pattern matching based on spatial paths is a key component. We expect that our method is applied in practical spatio-temporal systems in the future.

Acknowledgment: Y. Ishikawa and C. Xiao were partly supported by KAKENHI (16H01722).

References

[1] Y. Han, W. Sun, and B. Zheng, "COMPRESS: A comprehensive framework of trajectory compression in road networks," *ACM Trans. Database Syst.*, vol. 42, no. 2, pp. 11:1–11:49, May 2017.

[2] G. Kellaris, N. Pelekis, and Y. Theodoridis, "Map-matched trajectory compression," *J. Syst. Softw.*, vol. 86, no. 6, pp. 1566–1579, 2013.

[3] N. R. Brisaboa, A. Fariña, D. Galaktionov, and M. A. Rodríguez, "Compact trip representation over networks," in *Proc. SPIRE'16*, 2016, pp. 240–253.

[4] B. Krogh, C. S. Jensen, and K. Torp, "Efficient in-memory indexing of network-constrained trajectories," in *Proc. GIS '16*, no. 17, 2016.

[5] I. Sandu Popa, K. Zeitouni, V. Oria, and A. Kharrat, "Spatio-temporal compression of trajectories in road networks," *GeoInformatica*, vol. 19, no. 1, pp. 117–145, 2014.

[6] S. Koide, Y. Tadokoro, and T. Yoshimura, "SNT-index: Spatio-temporal index for vehicular trajectories on a road network based on substring matching," in *Proc. SIGSPATIAL UrbanGIS Workshop'15*, 2015.

[7] U. Manber and G. Myers, "Suffix arrays: a new method for on-line string searches," in *Proc. SODA'90*, 1990.

[8] P. Ferragina and G. Manzini, "Opportunistic data structures with applications," in *Proc. FOCS'00*, 2000.

[9] M. Burrows and D. J. Wheeler, "A block-sorting lossless data compression algorithm," in *Tech. Report 124, Digital Equipment Corporation*.

[10] R. Grossi, A. Gupta, and J. S. Vitter, "High-order entropy-compressed text indexes," in *Proc. SODA'03*, 2003, pp. 841–850.

[11] G. Jacobson, "Space efficient static trees and graphs," in *Proc. FOCS'89*, 1989, pp. 549–554.

[12] R. Raman, V. Raman, and S. Rao, "Succinct indexable dictionaries with applications to encoding k-ary trees and multisets," in *Proc. SODA'02*, 2002, pp. 233–242.

[13] G. Navarro, "Wavelet trees for all," in *Proc. CPM'12*, 2012, pp. 2–26.

[14] G. Manzini, "An analysis of the Burrows-Wheeler transform," *J. ACM*, vol. 48, no. 3, pp. 407–430, 2001.

[15] J. Kärkkäinen and S. J. Puglisi, "Fixed block compression boosting in FM-Indexes," in *Proc. SPIRE'12*, 2011, pp. 174–184.

[16] V. Mäkinen and G. Navarro, "Implicit compression boosting with applications to self-indexing," in *Proc. SPIRE'07*, 2007, pp. 229–241.

[17] F. Claude and G. Navarro, "The wavelet matrix," in *Proc. SPIRE'12*, 2012, pp. 167–179.

[18] G. Navarro and E. Providel, "Fast, small, simple rank / select on bitmaps," in *Proc. SEA'12*, 2012, pp. 295–306.

[19] G. Navarro, *Compact Data Structures: A Practical Approach.* Cambridge University Press, 2016.

[20] K. Sadakane, "Succinct data structures for flexible text retrieval systems," *Journal of Discrete Algorithms*, vol. 5, no. 1, pp. 12–22, 2007.

[21] S. Koide, Y. Tadokoro, C. Xiao, and Y. Ishikawa, "CiNCT: Compression and retrieval for massive vehicular trajectories via relative movement labeling," *CoRR*, 2017. [Online]. Available: https://arxiv.org/abs/1706.02885

[22] A. Golynski, J. I. Munro, and S. S. Rao, "Rank/select operations on large alphabets: a tool for text indexing," in *Proc. SODA'06*, 2006, pp. 368–373.

[23] J. Barbay, T. Gagie, G. Navarro, and Y. Nekrich, "Alphabet partitioning for compressed rank/select and applications," in *Proc. ISAAC'10*, 2010, pp. 315–326.

[24] S. Gog, A. Moffat, and M. Petri, "CSA++: fast pattern search for large alphabets," in *Proc. ALENEX'17*, 2017, pp. 73–82.

[25] N. Larsson and A. Moffat, "Offline dictionary-based compression," in *Proc. DCC '99*, 1999, pp. 296–305.

[26] R. Song, W. Sun, B. Zheng, and Y. Zheng, "PRESS: A novel framework of trajectory compression in road networks," *PVLDB*, vol. 7, no. 9, pp. 661–672, 2014.

[27] V. T. de Almeida and R. H. Güting, "Indexing the trajectories of moving objects in networks," *Geoinformatica*, vol. 9, no. 1, pp. 33–60, 2005.

[28] P. Newson and J. Krumm, "Hidden Markov map matching through noise and sparseness," in *Proc GIS'09*, 2009, pp. 336–343.

[29] L. Nguyen-Dinh, W. G. Aref, and M. F. Mokbel, "Spatio-temporal access methods: Part 2 (2003–2010)," *IEEE Data Eng. Bull.*, vol. 33, no. 2, pp. 46–55, 2010.

[30] B. Krogh, N. Pelekis, Y. Theodoridis, and K. Torp, "Path-based queries on trajectory data," in *Proc. GIS'14*, 2014, pp. 341–350.

[31] T. Gagie, G. Manzini, and J. Sirén, "Wheeler graphs: A framework for BWT-based data structures," *Theoretical Computer Science*, vol. 698, pp. 67–78, 2017.