Compact Trip Representation over Networks *

Nieves R. Brisaboa[1], Antonio Fariña[1], Daniil Galaktionov[1], and M. Andrea Rodríguez[2]

[1] Database Laboratory, University of A Coruña, Spain
[2] Department of Computer Science, University of Concepción, Chile

Abstract. We present a new *Compact Trip Representation* (CTR) that allows us to manage users' trips (moving objects) over networks. These could be public transportation networks (buses, subway, trains, and so on) where nodes are stations or stops, or road networks where nodes are intersections. CTR represents the sequences of nodes and time instants in users' trips. The spatial component is handled with a data structure based on the well-known Compressed Suffix Array (CSA), which provides both a compact representation and interesting indexing capabilities. We also represent the temporal component of the trips, that is, the time instants when users visit nodes in their trips. We create a sequence with these time instants, which are then self-indexed with a balanced Wavelet Matrix (WM). This gives us the ability to solve range-interval queries efficiently. We show how CTR can solve relevant spatial and spatio-temporal queries over large sets of trajectories. Finally, we also provide experimental results to show the space requirements and query efficiency of CTR.

1 Introduction

Current technology allows us to capture data about the usage of transportation networks whose analysis could have an important impact on improving the quality of services. Data about the origin and destination of passengers of train services can be directly captured when selling tickets. Using more sophisticated technology, the movement of people or vehicles over networks of streets or roads can be collected from the mobile phone signals. Even more, nowadays many cities (from London to Santiago of Chile) provide smartcards to the users of their public transportation network. These smartcards (that can be recharged with money) allow users to pay the entrance to subways and buses. Even though there typically exists only a card reader in the entrance to the network (i.e.,

* Funded in part by European Union's Horizon 2020 research and innovation programme under the Marie Sklodowska-Curie grant agreement No 690941 (project BIRDS). N. Brisaboa, A. Fariña, and D. Galaktionov are funded by MINECO (PGE, CDTI, and FEDER) [TIN2013-46238-C4-3-R, TIN2013-47090-C3-3-P, TIN2015-69951-R, IDI-20141259, ITC-20151305, ITC-20151247]; by ICT COST Action IC1302; and by Xunta de Galicia (co-funded with FEDER) [GRC2013/053]. A. Rodríguez is funded by Fondecyt 1140428 and the Complex Engineering Systems Institute (CONICYT: FBO 16)

there is no control at exits or in middle stops), it is possible to know how people actually use the public transportation by collecting the entrance and estimating the destination (e.g., as the entry point for the return trip) and the traversed stops (the shorter path among stops used as the entrance and exit) [11]. In all scenarios, the massive data about trips makes the problem of storing and efficiently accessing data about trips a challenging computational problem.

This paper presents a compact and self-indexed data structure to represent trips over networks, which could be public transportation networks where nodes are stations or stops, or road networks where nodes are intersection points.

Although there exist proposals of data structures for moving objects, they have addressed typical spatio-temporal queries such as time slice or time interval queries that retrieve trajectories or objects that were in a spatial region at a time instant or during a time interval. They were not designed to answer queries that are based on counting occurrences such as the number of trips starting or ending at some time instant in specific stops (nodes) or the top-k most used stops of a network during a given time interval, which are more meaningful queries for public-transportation or traffic administrators. Our proposal (CTR) is oriented to efficiently answering these types of queries, and it differs from previous approaches in the use of compact self-indexed data structures to represent the big amount of trips in compact space. It is important to emphasize that our goal is to provide an indexed representation for a static collection of trips in order to allow an efficient batch processing of such data.

CTR combines two well-known data structures. The first one, initially designed for the representation of strings, is Sadakane's Compressed Suffix Array (CSA) [18]. The second one is the Wavelet Matrix (WM) [1]. To make the use of the CSA possible in this domain, we define a trip or trajectory of a moving object over a network as the temporally-ordered sequence of the nodes the trip traverses. An integer *id* is assigned to each node such that a trip is a string of nodes' *id*s. Then a CSA, over the concatenation of these strings (trips) is built with some adaptations for this context. In addition, we discretize the time in periods of fixed duration (i.e. timeline split into 5-minute instants) and each time segment is identified by an integer *id*. In this way, it is possible to store the times when trips reach each node by associating the corresponding time *id* with each node in each trip. The sequence of times for all the nodes within a trip is self-indexed with a WM to efficiently answer spatio-temporal queries.

We experimentally tested our proposal using two sets of synthetic data representing trips over two different real public transportation systems. Our results are promising because the representation uses only around 30% of its original size and answers spatial and spatio-temporal queries in microseconds. No experimental comparisons with classical spatial or spatio-temporal index structures are possible, because none of them were designed to answer the types of queries in this work. Our approach can be considered as a proof of concept that opens new application domains for the use of CSA and WM, creating a new strategy for exploiting trajectories represented in a self-indexed way.

The organization of this paper is as follows. Section 2 reviews previous works on trip representations. It also makes reference to the CSA and WM, upon which we develop our proposal. Section 3.1 shows how CTR represents the spatial component and Section 3.2 the temporal component of trips. Section 4 presents the relevant queries that are solved by CTR and Section 5 gives our experimental results. Finally, conclusions and future work are discussed in Section 6.

2 Previous Work

Trajectory indexing. Many data structures have been proposed to support efficient query capabilities on collections of trajectories. We refer to [13, Chapter 4] for a comprehensive and up-to-date survey on data management techniques for trajectories of moving objects. We can broadly classify these data structures into two groups: those that index trajectories in free space and those that index trajectories constrained to a network. The 3D R-tree (an extension of the classical R-tree spatial index [7]), the TB-tree [14], and MV3R-tree [19] are examples of the former, whereas the FNR-tree [4], the MON-tree [2], and PARINET [15] are examples of the latter. While the former type of structures could also apply over networks, the second type exploits the constraints imposed by the topology of the network to optimize the data structure. From them, PARINET is the most efficient alternative [15]. It partitions trajectories into segments from an underlaying road network, and then adds one temporal B^+-tree to index the trajectory segments from each road. Those indexes permit us to filter out candidate trajectory segments matching time constraints at query time.

All previous data structures were designed to answer spatio-temporal queries, where the space and time are the main filtering criteria. Examples of such queries are: *retrieve trajectories that crossed a region within a time interval*, *retrieve trajectories that intersect*, or *retrieve the k-best connected trajectories* (i.e., the most similar trajectories in terms of a distance function). Yet, they could not easily support queries such as *number of trips starting in X and ending at Y*.

The application of data compression techniques has been explored in the context of massive data about trajectories. The work by Meratnia and de By [10] adapts a classical simplification algorithm by Douglas and Peucker to reduce the number of points in a curve and, in consequence, the space use to represent trajectories. Ptomaias *et al.* [16] use concepts, such as speed and orientation, to improve compression. Both techniques work for trajectories in free space.

In [17, 8, 5], they focus mainly on how to represent trajectories constrained to networks, and in how to gather the location of one or more given moving objects from those trajectories. Yet, these works are also out of our scope as they would poorly support queries oriented to exploit the data about the network usage such as those oriented to aggregate the number of trips with a specific spatio-temporal pattern (e.g. Count the trips starting at stop X and ending at stop Y in working days between 7:00 and 9:00).

In [9], authors use a representation of trajectories where for each edge in a trajectory both the starting and ending times are kept, and present an index

called *NETTRA*. They used a relational database where those data are stored in a table and indexes are created in order to support a particular type of queries called *Strict Path Queries*. Although our CTR could also deal with those types of queries, this database-oriented representation is out of our scope as they do not consider space constraints (they do not compress data nor do they consider the size of the indexes used).

Underlying Compact Structures of CTR. Our proposal is based on two well-known compact structures: a Compressed Suffix Array (CSA) [18] and a Wavelet Matrix (WM) [1]. We used the variant of CSA from [3], where authors adapted CSA to deal with large (integer-based) alphabets and created the *integer-based CSA* (iCSA). They also showed that the best compression of Ψ was obtained when combining differential encoding of runs with Huffman and run-length encoding.

WM is a data structure originated from the Wavelet Tree [6], but requires less space and permits to make an efficient occurrence count of a continuous range of values [1] (see Section 3.2 for details). WM provides, as the Wavelet Tree a self-indexed representation of symbols based on the rearrangement of their bits in different bit maps at different levels. WM allows us to perform efficient operations over the sequence, among other operations: $access(i)$ returns the symbol at the position i, $rank_\alpha(i)$ counts the number of occurrences of a symbol α up to position i; and $select_\alpha(j)$ gives the position of the j-th α. Those operations are implemented using the classical bit operations *rank* and *select* on the underlying bitmaps and they need $O(\log \sigma)$ time, being σ the number of encoded symbols.

3 Compact Trip Representation (CTR)

Trips on networks are temporally-ordered sequences of nodes (referred to as the spatial component) tagged with timestamps (referred to as the temporal component). We show how the proposed *Compact Trip Representation* (CTR) combines a Compact Suffix Array (CSA) to represent the spatial component and a Wavelet Matrix (WM) to represent the temporal one.

3.1 Representing the spatial component of CTR with a CSA

In CTR, integer IDs identify stops of the network. The first step to build the CSA is to sort the trips. They are sorted by the first stop, then by the last stop, then by the start time of the trip, and finally by the second, third, and successive stops. For example, we have a dataset $\mathcal{T}$ with the following set of trips: $\{\langle 2,3,10,6\rangle$, $\langle 2,3,10,4,7\rangle$, $\langle 1,2,3\rangle$ $\langle 3,10,5\rangle$, $\langle 1,2,3\rangle$ $\langle 9,8,7\rangle\}$. Let us assume that these trips start at time instants $10, 2, 0, 9, 5, 12$, respectively. Following lexicographic order, the trip $\langle 2,3,10,4,7\rangle$ should be before the trip $\langle 2,3,10,6\rangle$. However, because after the first stop, we consider the last stop, the trip $\langle 2,3,10,6\rangle$ goes before the trip $\langle 2,3,10,4,7\rangle$. In addition, the two trips $\langle 1,2,3\rangle$ are sorted by their starting

time instants (0 and 5 respectively). This sorting of the trips will allow us to answer a useful query very efficiently (i.e., trips starting at X and ending at Y).

We concatenate the sorted trips and construct an array S where trips are separated with a symbol \$. We also add an additional ending \$. Figure 1 shows the array S for the running example. Despite the standard suffix array construction in the CSA that compares two suffixes by their lexicographical order until the end of S, we introduced a modification so that two suffixes are now compared considering their trips as a cycle.

Figure 1 depicts the structures Ψ and D used by the CTR over the trips in the dataset $\mathcal{T}$. There is also the vocabulary V containing all the stops in their lexicographic order, as well as the \$ symbol. We include the sequence S, the suffix array A, and Ψ' only for clarity (they are not needed in the CTR). Ψ' contains the first entries of Ψ from a regular CSA, just to explain the difference of how we build Ψ. For example, $A[8] = 1$ points to the first stop of the first trip $S[1]$. $\Psi[8] = 10$ and $A[10] = 2$ points to the second stop. $\Psi[10] = 14$ and $A[14] = 3$ points to the third stop. $\Psi[14] = 2$ and $A[2] = 4$ points to the ending \$ of the first trip. Therefore, in the standard CSA, $\Psi'[2] = 9$ and $A[9] = 5$ points to the first stop of the second trip. However, in CTR, $\Psi[2] = 8$ and $A[8] = 1$ points to the first stop of the first trip. Thus, subsequent applications of Ψ will allow us to cyclically traverse the stops of the trip. Finally, note that aligned with sequence S, we could keep the times associated with the stops in each trip with the structures I and $Icode$, which are explained in the following subsection.

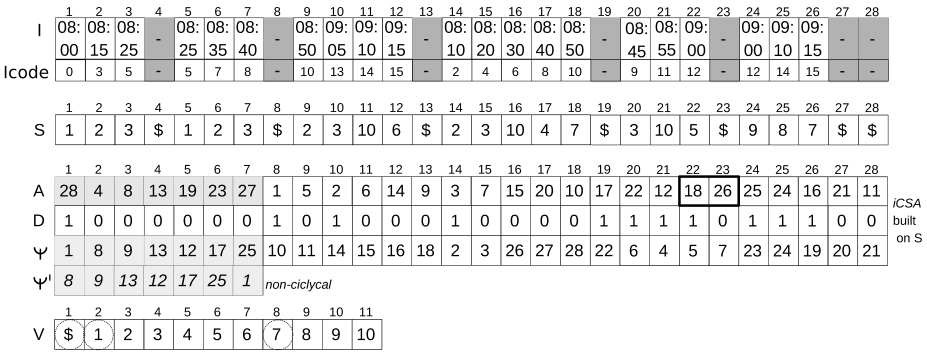

Fig. 1. Structures involved in the creation of a CTR.

The definition of *a suffix* proposed above explains why $A[22] = 18$ is placed before $A[23] = 26$. Note that the suffix starting at $S[18]$ is "$7 \cdot \$ \cdot 2 \cdot 3 \ldots$" and that suffix at $S[26]$ is "$7 \cdot \$ \cdot 9 \cdot \ldots$". Therefore, it holds that $A[22] \prec A[23]$. However, considering the traditional definition of a *suffix*, these suffixes would be "$7 \cdot \$ \cdot 3 \cdots$" and "$7 \cdot \$ \cdot \$ \cdots$" respectively, and $A[22] \prec A[23]$ would not hold.

Note also that, in the shaded range $\Psi[1, 7]$, the first entry is related to terminator \$, whereas the next six entries correspond to the \$ symbols that mark the end of each trip in S (sorted by the starting stop, then by the ending stop, then by their initial time, and finally by the second, third and following up

stops). This property makes it very simple to find starting stops. For example, the ending \$ of the 4^{th} trip is at the 5^{th} position (because the first \$ corresponds to the final \$ at $S[28]$). Therefore, its starting stop can be obtained by $\Psi[5] = 12$ and $rank_1(D, 12) = 3$; that is, the starting stop is the 3^{th} entry in the vocabulary. The next stop of that trip would be obtained by $\Psi[12] = 16$ and $rank_1(D, 16) = 4$, and so on.

We expect to obtain good compressibility in CTR due to the structure of the network, and the fact that trips that start in a given stop or simply those going through that stop will probably share the same sequence of "next" stops. This will lead us to obtain many *runs* in Ψ [12], and consequently, good compression.

3.2 Representing the temporal component of CTR with a WM

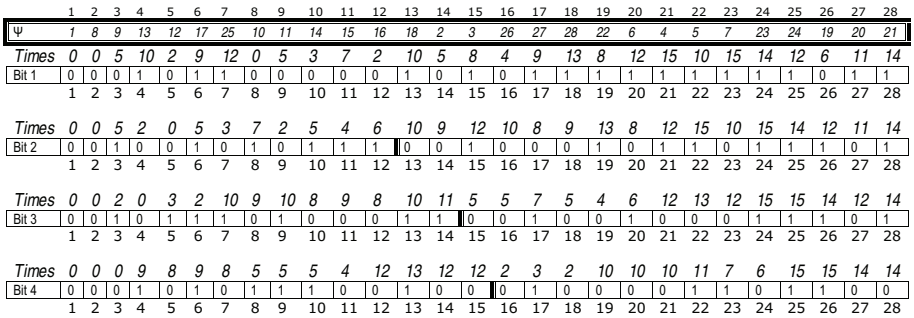

Fig. 2. WM representation for the times associated with the trips in Figure 1.

To exploit usage patterns of a network, we need to represent and query the time component of trips, which indicates when a moving object reaches each node along its trip. To represent this time component, we discretize time and assign an integer code to each resulting time interval. The size of the time interval is a parameter that can be adjusted to fit the required precision in each domain. For example, in a public transportation network, if we had data about five years of trips, a possibility would be to divide that five-years period into 10-minutes intervals, or in cyclical annual periods resulting in a vocabulary of roughly $365 \times 24 \times 60/10 = 52,560$ different codes. However, in public transportation networks queries such as *"Number of trips using the stop X on May 10 between 9:15 and 10:00"* may be not as useful as queries such as *"Number of trips using stop X on Sundays between 9:15 and 10:00"*. For this reason, CTR can adapt how the time component is encoded depending on the queries that the system must answer.

In Figure 1, sequence I contains the time associated with each stop in a trip, and $Icode$ a possible encoding of times. In CTR we use a similar encoding to that in $Icode$, yet aligned to Ψ rather than to S.

Those entries in $Icodes$ are given a fixed-length binary code and are represented with a balanced Wavelet Matrix (WM) [1]. That is, for any stop in a trip at the position i in Ψ, its timestamp t_i can be recovered by accessing the WM

at position i. Recall [1] that a WM is a grid of $n \times m$ bits. In our case n is the number of entries in the CSA and $m = \log \sigma_t$ are the bits needed to represent the different σ_t codes for the time instants of interest.

Besides the typical $access(i)$, $rank_\alpha(i)$ and $select_\alpha(i)$, the WM provides a *count* operation that CTR heavily relies on. $count(x1, x2, y1, y2)$ returns the number of occurrences of symbols between y1 and y2 in the range of positions $[x1, x2]$ from the encoded sequence in $O(m)$ time. While its implementation details can be found in [1], we include an example of how to solve $count(20, 28, 10, 15)$ over the sequence shown in Figure 2. The algorithm starts from the upper level ($Bit1$) of the WM and iterates downwards, refining the searching range. In $Bit1$ we are only interested in positions from $[20, 28]$ that have a 1, because none of the symbols between 10 and 15 starts with a 0. Also, since $rank_0(Bit1, 28) = 12$, in $Bit2$ we will have to search in the positions between $12 + rank_1(Bit1, 20) = 21$ and $12 + rank_1(Bit1, 28) = 28$. Now, while the second bit for 10 and 11 is 0, it is 1 for the symbols between 12 and 15. Because of this, we need to perform both $rank_0$ and $rank_1$ on the limits of $[21, 28]$ in $Bit2$[3], and split the search in two subranges for $Bit3$: $[10, 11]$ using $rank_0$ and $[23, 28]$ using $rank_1$. As the second subrange may only contain symbols from 12 to 15 (11xx), further refinement is not needed. In the case of the range $[10, 11]$, it could contain symbols from 8 to 11, depending on their third bits, so we need to perform $rank_1$ over its limits in $Bit3$, which leads to $[21, 22]$ in $Bit4$. The number of 10 and 11 symbols is the size of this last range.

If we wanted to return the positions of the results in the original sequence, we could do that with a simple algorithm, using *select* of bits over bitmaps, that iterates upwards from the level where each result is found until the first level where its position in the original sequence can be retrieved.

Summarizing, CTR takes the advantage of the WM to count and report the occurrences of a continuous range of values. The starting positions in the CSA belonging to the \$ symbols have no time by themselves, but it is useful to answer some queries to store the starting time instants of the corresponding trip in these positions too.

The time intervals could be mapped to a variable-length code, instead of a fixed length codes, where the most frequent intervals would be represented by less bits and, therefore, requiring less levels in a Wavelet Tree. In the future we will explore this possibility.

4 Query processing

We distinguish two types of queries to be answered by the CTR: spatial and spatio-temporal queries. We briefly sketch the algorithms to process these queries.

Spatial queries. The following queries can be solved by only using the CSA that represents the spatial component of trips.

[3] $rank_1(Bit2, i) = i - rank_0(Bit2, i)$, and vice versa

- *Number of trips starting at stop X.* Because Ψ was cyclically built in such a way that every \$ symbol is followed by the first stop of its trip, this query is solved by performing the binary search of the pattern $\$X$ over the section of Ψ corresponding to \$. The size of the resulting range gives the number of trips starting at X.
- *Number of trips ending at stop X.* In a similar way to the previous query, this one can be answered with a binary search for pattern $X\$$ over the section of Ψ corresponding to stop X.
- *Number of trips starting at X and ending at Y.* Combining both ideas from above, this query is solved directly by searching for the $Y\$X$ pattern.
- *Number of trips using stop X.* Instead of performing a binary search over Ψ, we operate on bitmap D. Assuming that X is at position p in the vocabulary V of CTR, its total frequency is obtained by $occs_X \leftarrow select_1(D, p+1) - select_1(D, p)$. If p is the last entry in V, we set $occs_X \leftarrow n + 1 - select_1(D, p)$.
- *Top-k most used stops.* We provide two possible solutions for these queries: sequential and binary-partition approaches.
  - To return the k most used stops using a sequential approach, we can apply $select_1$ operation sequentially for every stop from 1 to δ, returning the k stops with highest frequency. We use a min-heap that is initialized with the first k stops, and for every stop s from $k+1$ to δ, we compare its frequency with the frequency of the minimum stop in the heap. In case the new one is higher, the root of the heap is replaced and moved down to comply with the heap ordering. At the end of the process, the heap will contain the top-k most used stops, which can be sorted with the heapsort algorithm if needed. Note that this approach always performs δ $select_1$ operations on D.
  - A binary-partition approach to solve queries about the top-k most used stops takes advantage of the skewed distribution of the stops that trips visit. Working over D and V, D is recursively split into segments of D after each iteration. Each partition must, if possible, leave the same number of different stops in each side of the partition. The segments created after the partitioning step are pushed into a priority queue Q, storing the initial and the final positions of the segment in D, and also the initial and final corresponding entries in V. The priority of each segment in Q is directly its size. The priority queue Q is initialized with a segment covering the whole D (without its initial range of δ \$ symbols). When a segment extracted from the queue Q represents the instance of only one stop, that stop is returned as a result of the top-k algorithm. The algorithm stops when the first k stops are found.
    For example, when searching for the top-1 most used stops in the running example, Q is initialized with the segment $[8, 28]$, corresponding to stops from 1 to 10 (positions from 2 to 11 in V). Note that the entries of D from 1 to 7 and $V[1]$ represent the \$ symbol. These are not stops and must be skipped. Then $[8, 28]$ is split producing the segments $[8, 20]$ for stops 1 to 5 and $[21, 28]$ for stops 6 to 10. After three more iterations, we extract the segment $[14, 18]$ for the single stop 3, concluding that the top-1 most used stop is 3 with a frequency equal to 5.

Spatio-temporal queries. These queries combine both the CSA and WM. The idea is to restrict spatial queries to a time interval $[t_1, t_2]$. An example of this type of query is to return the *number of trips starting at stop X between t_1 and t_2*, which we solve by relying on the *count* operation of the WM. The following are the spatio-temporal queries solved by the CTR:

- *Number of trips starting at stop X during the time interval $[t_1, t_2]$.* Remember that in the WM we also have timestamps associated with the area of \$-symbols in Ψ; each \$ has associated the time of the first stop of its trip and, therefore, we can use the WM in that area of Ψ. Using the range in Ψ obtained by searching the $\$X$ pattern, as done in a regular spatial query, a *count* operation is performed over these positions in the WM searching for the limits of the interval. That is, we count the number of entries in the obtained range that have a timestamp in the WM inside $[t_1, t_2]$.
- *Number of trips ending at stop X during the time interval $[t_1, t_2]$.* As before, we use a *count* operation in the WM, restricted to the range in Ψ that corresponds to the pattern $X\$$ found in the spatial query.
- *Number of trips using stop X during the time interval $[t_1, t_2]$.* As in the spatial query, the range in Ψ is obtained with two $select_1$ on D. Then, a *count* operation is done over the WM to find the occurrences inside the time interval $[t_1, t_2]$.
- *Number of trips starting at X and ending at Y occurring during time interval $[t_1, t_2]$.* We consider two different semantics. A query with *strong semantics* will obtain trips that start and end inside $[t_1, t_2]$. Whereas, a query with *weak semantics* will obtain trips whose time intervals overlap $[t_1, t_2]$ and, therefore, they could actually start before t_1 or end after t_2.
  We can binary search Ψ for the pattern $Y\$X$, hence obtaining the corresponding continuous range of positions in the section of Ψ devoted to Y. We know that the range for $Y\$X$ in Ψ has pointers to the section \$ in Ψ. But, note that taking into account the considerations in the sorting of trips when building the CSA, this section $\$XY$ is a continuous range of the same size than the range $Y\$X$, and it also preserves the same order of the trips.
  Note that, the range $Y\$X$ of Ψ has associated the final time of each trip in the WM, whereas the range $\$XY$ has associated the timestamps of the starting time of each trip in increasing order (due to how we sorted the trips). Therefore, we can use these ranges, respectively, to check time constraints related to the ending stop (Y) and to the starting stop ($\$X$) of each trip.
  - *Strong semantics.* Since time instants within the range $\$XY$ are in increasing order, we can use the WM to obtain a continuous subrange (inside $\$XY$) of trips starting during the interval $[t_1, t_2]$. That subrange has a matching subrange inside of the range $Y\$X$ corresponding to the final stop of those trips (in the same order). We can again use the WM to count the number of those trips with valid ending times. That is, we can perform a *count* operation in the WM over the subrange of $Y\$X$ corresponding to the subrange of $\$XY$ with valid starting times.
  - *Weak semantics.* In this case we need to consider all the trips in the range $\$XY$ starting within $[t_1, t_2]$, as well as the ones starting before t_1 but ending after t_1.

5 Experimental evaluation

In this section we provide experimental results to show how CTR handles a large collection of trips. We discuss both the space requirements of our representation and also show its performance at query time. Although due to legal issues we could not provide experiments over real trips gathered from transport companies, we managed to use real data of the Madrid's public transportation network[4] (in the GTFS[5] format) to generate two datasets of synthetic trips:

- **Subway trips.** This combines the subway network with the Spanish commuter rail system called "Cercanías". In total, there are 313 different stations organized in 23 lines. They are open to the public from 6:00 AM to 2:00 AM, thus trips were always generated within 20 hours a day.
- **Bus trips.** It uses 4648 bus stops, organized in 206 lines. Some of these lines are from special night services, so we generated trips using 24 hours a day.

Trip generation process choses a starting stop and an ending stop, and uses the network description to generate every stop that the trip must traverse. We generated 50 million trips in both datasets, whose lengths vary from 2 to 31 stops following a binomial distribution with a mean length of 11.81 stops. Based on the GTFS data, we also generated realistic timestamps along each stop, and built the WM-based time index in CTR discretizing these timestamps into 5-minute intervals. We distinguished four kinds of days in a week: regular working days, Fridays/holiday eves, Saturdays, and Sundays/holidays; and two kinds of weeks for high and low season representations. In total, a time interval may belong to eight types of day.

Below, we show the space/time tradeoff for both datasets obtained by three settings of CTR. We tune its Ψ sample rate parameter to values 16, 64, and 256, respectively. All tests were run on a machine with an Intel(R) Core(TM) i5-4440@3.1GHz CPU, and 8GB DDR3 RAM. The operating system was Ubuntu 15.04 and the compiler gcc 4.9.2 (options -O3).

We compare the space usage of the stops representation in the CTR with the space required by two baseline compressors: *gzip* and *bzip2*. To measure the compression, we assume, as a reference, a plain representation that uses the least amount of bits needed to represent every stop with a fixed width[6]. The sizes of these plain representations are 687.28 MiB for the subway trips dataset, and 992.59 MiB for the bus trips. Note that we ignore the space needed for the representation of time intervals, as WM does not offer any compression by itself, and needs 866.27 and 944.88 MiB for subway and bus trips, respectively.

Results regarding space usage are given in Table 1. Note that an iCSA built on English text [3] typically reached the compression of *gzip* (around 35% in compression ratio). As expected, the high compressibility of our sorted dataset

[4] Data from the EMT corporation https://www.emtmadrid.es/movilidad20/googlet.html

[5] GTFS is a well-known specification for representing an urban transportation network. See https://developers.google.com/transit/gtfs/reference?hl=en

[6] 9 bits/stop for subway trips, 13 bits/stop for bus trips

of trips permits CTR to improve those numbers with compression ratios under 30% in the most sparse setup, much better than *gzip*, and even than *bzip2*. Yet, CTR offers also indexing features that allow us to perform efficient searches.

To provide a rough comparison with a database solution similar to *NETTRA* [9] we included in a table a row containing each trip ID (represented with 4 bytes), stop ID (represented with 2 bytes), and time interval (represented with 2 bytes instead of a full `datetime`). The size of the whole table was around 4505 MiB, without taking any indexes into account. Therefore, such representation would use at least more than twice the space of CTR while it could not efficiently support the queries discussed in this paper.

Dataset	Ψ_{16}	Ψ_{64}	Ψ_{256}	gzip	bzip2
Subway	467.07 (67.96%)	249.14 (36.25%)	193.10 (28.10%)	401.72 (58.45%)	238.43 (34.69%)
Bus	499.84 (50.36%)	283.12 (28.52%)	227.42 (22.91%)	957.03 (96.42%)	389.74 (39.26%)

Table 1. Comparison on space usage for stops. Space in MiB.

To see the query performance of CTR, we generated 10,000 random queries of each type, and measured the average time required to solve them.

Table 2 shows the results of spatial queries. Almost any query can be solved in the order of ten μsecs and the heaviest Top-k within $m$secs per query in our experiments. As expected, the query *"ends at X"* performs slightly faster than *"starts at X"*, as the region in Ψ for any stop X is smaller than the region of \$, thus needing more time to search a pattern inside the later. It is also expected that the spatial *"uses X"* performs much faster than any other query as it does not operate over Ψ and its samples, using instead the $select_1$ operator over D. For the same reasons, both spatial Top-k algorithms are also independent from the Ψ sample rate parameter. However, it is interesting to point out that even when the binary partitioning algorithm is much faster for small values of k, its sequential counterpart overcomes it for large values of k. This is a reasonable phenomena considering that for large values of k, the number of $select_1$ operations that the binary partitioning algorithm needs to perform tends to be the same as in the sequential algorithm, but with the additional cost of maintaining a larger and more complex structure (a priority queue versus a binary heap).

CTR	Starts at X	Ends at X	Starts at X ends at Y	Uses X	Sequential Top 10	Binary Top 10	Sequential Top 1000	Binary Top 1000
Subway Ψ_{16}	6.03	4.53	11.24					
Subway Ψ_{64}	8.22	4.61	16.68	0.3902	50.42	39.36	62.79	75.09
Subway Ψ_{256}	18.78	5.69	38.82					
Bus Ψ_{16}	7.51	6.27	9.24					
Bus Ψ_{64}	9.58	6.52	15.72	0.7944	761.14	588.01	1031.84	1514.07
Bus Ψ_{256}	22.77	11.35	41.31					

Table 2. Time performance for spatial queries (in $\mu secs$/query).

Table 3 shows the results of spatio-temporal queries. Looking at the differences between spatial queries and their spatio-temporal counterparts, it can be

seen that computing a *count* query over the WM takes roughly around 3 μsec, so its time overhead is relatively small.

CTR	Starts at X	Ends at X	Starts at X ends at Y (strong)	Starts at X ends at Y (weak)	Uses X
Subway Ψ_{16}	8.34	7.44	22.42	18.95	
Subway Ψ_{64}	11.21	7.83	28.07	24.32	2.08
Subway Ψ_{256}	21.68	8.58	49.98	46.50	
Bus Ψ_{16}	10.41	9.50	12.25	12.12	
Bus Ψ_{64}	12.95	10.19	18.84	18.75	4.90
Bus Ψ_{256}	26.20	14.87	44.84	44.92	

Table 3. Time performance for spatio-temporal queries (in $\mu secs$/query).

6 Conclusions and future work

As better tracking mechanisms will be installed, the problem of storing and querying trips to support network analysis will gain interest for network management administrations and even end-user applications. For instance, with enough data of vehicle trips from a significant amount of drivers over the network formed by the streets of a city, it would be possible to infer traffic rules by examining turns that nobody takes, their usual driving speed across the network, and other useful information.

We showed that CTR is a powerful structure to represent user trips. Using compact data structures to represent trips over a transportation network allows us not only to keep a much larger amount of data in main memory (compression ratio is around 30%), but also to efficiently perform spatial and spatio-temporal queries oriented to understand the real usage of the network.

We have presented CTR as a proof of concept development. It is flexible enough to allow new adaptations and functionality improvements we plan to do as future work, such as the analysis of line changes in switching stops (that would require storing the network topology) or providing compression for the time index. Also, future work considers providing new experiments over real data of trips.

References

1. F. Claude, G. Navarro, and A. Ordóñez . The wavelet matrix: An efficient wavelet tree for large alphabets. *Inf. Systems*, 47:15–32, 2015.
2. V. T. de Almeida and R.H. Güting. Indexing the Trajectories of Moving Objects in Networks. *GeoInformatica*, 9(1):33–60, 2005.
3. A. Fariña, N. Brisaboa, G. Navarro, F. Claude, A. Places, and E. Rodríguez. Word-based self-indexes for natural language text. *ACM TOIS*, 30(1):article 1, 2012.
4. Elias Frentzos. Indexing Objects Moving on Fixed Networks. In *Proc. 8th SSTD*, pages 289–305, 2003.
5. S. Funke, R. Schirrmeister, S. Skilevic, and S. Storandt. Compass-based navigation in street networks. In *Proc. 14th W2GIS*, LNCS 9080, pages 71–88, 2015.
6. R. Grossi, A. Gupta, and J.S. Vitter. High-order entropy-compressed text indexes. In *Proc. 14th SODA*, pages 841–850, 2003.

7. A. Guttman. R-trees: A dynamic index structure for spatial searching. In *Proc. SIGMOD*, pages 47–57, 1984.
8. G. Kellaris, N. Pelekis, and Y. Theodoridis. Map-matched Trajectory Compression. *Journal of Systems and Software*, 86(6):1566–1579, 2013.
9. Benjamin Krogh, Nikos Pelekis, Yannis Theodoridis, and Kristian Torp. Path-based queries on trajectory data. In *Proceedings of the 22nd ACM SIGSPATIAL International Conference on Advances in Geographic Information Systems*, pages 341–350. ACM, 2014.
10. N. Meratnia and Rolf A. de By. Spatiotemporal compression techniques for moving point objects. In *Proc. 9th EDBT 2004*, LNCS 2992, pages 765–782, 2004.
11. M. A. Munizaga and C. Palma. Estimation of a disaggregate multimodal public transport origin–destination matrix from passive smartcard data from santiago, chile. *Transportation Research Part C: Emerging Technologies*, 24:9 – 18, 2012.
12. G. Navarro and V. Mäkinen. Compressed full-text indexes. *ACM Computing Surveys*, 39(1):article 2, 2007.
13. N. Pelekis and Y. Theodoridis. *Mobility Data Management and Exploration*. Springer, 2014.
14. D. Pfoser, C. S. Jensen, and Y. Theodoridis. Novel Approaches in Query Processing for Moving Object Trajectories. In *Proc. VLDB*, pages 395–406, 2000.
15. I. S. Popa, K. Zeitouni, V. Oria, D. Barth, and S. Vial. Indexing In-network Trajectory Flows. *VLDB J.*, 20(5):643–669, 2011.
16. M. Potamias, K. Patroumpas, and T. K. Sellis. Sampling Trajectory Streams with Spatiotemporal Criteria. In *Proc 18th SSDBM*, pages 275–284, 2006.
17. K. Richter, F. Schmid, and P. Laube. Semantic Trajectory Compression: Representing Urban Movement in a Nutshell. *J. Spatial Information Science*, 4(1):3–30, 2012.
18. K. Sadakane. New text indexing functionalities of the compressed suffix arrays. *Journal of Algorithms*, 48(2):294–313, 2003.
19. Y. Tao and D. Papadias. MV3R-Tree: A Spatio-Temporal Access Method for Timestamp and Interval Queries. In *Proc. VLDB*, pages 431–440, 2001.