A Compact Representation of Indoor Trajectories[*]

Antonio Fariña
Pablo Gutiérrez-Asorey
Susana Ladra
Miguel R. Penabad
Tirso V. Rodeiro
Universidade da Coruña, CITIC, Database Lab. Elviña, 15071. A Coruña, Spain.

March 9, 2024

Abstract

We present a system that combines indoor positioning with a compression algorithm for trajectories in the context of a nursing home. Our aim is to gather and effectively represent the location of the residents and caregivers along time, while allowing for efficient access to those data.

We briefly show the system architecture that enables the automatic tracking of user's movements and consequently the gathering of their locations. Then, we present indRep, our compact representation to handle positioning data using grammar-based compression, and provide two basic operations that enable pseudo-random access to the data. Finally, we include experiments that show that indRep is competitive with well-know general-purpose compressors in terms of compression effectiveness and also provides fast access to the compressed data. We expect both features would enable exploitation functionalities even in computers with rather low computational resources.

1 The introduction:

The current global pandemic scenario has resulted in a greater need for automatic systems in various contexts, and particularly in health-care related ones. We have considered the case of a local nursing home for the elderly as a prime example of this. We address the control of movements and activities within the facilities, proposing a system to automate the processes of information gathering, focusing on storing and exploiting said information efficiently.

While it is common for a facility like a nursing home to employ some sort of manually operated system to track staff activities, such as Helpnex [4], caregivers must use their personal card on a special reader to manually register the beginning and ending of an activity (e.g. changing the diapers of a resident). This rudimentary process presents high chances for errors.

To solve this issue, we present an automatic indoor trajectory collector infrastructure. While this is not new [7], our novel approach uses a grammar-based lossless compact structure, called *indRep*, demonstrating promising results, both in storage space and query times. This work establishes the basis for an automatic system capable of gathering, storing, and exploiting the positioning information on an indoor context. We will show that our representation effectively exploits the redundancy in the indoor trajectories, obtaining a large reduction in storage space with fast access to the data: our proposal is able to retrieve the full trajectory of a person during a day in just tens of microseconds while using just a small fraction of the space used by the raw data.

[*]Draft submitted for IEEE Pervasive Computing (2022).

Note that by providing a compressed representation of the original data, not only we reduce the space footprint but also enable using higher levels of the memory hierarchy when those data are accessed. This could lead to performance improvements which would be more noticeable when a large collection of trajectories can fit into main memory thanks to compression and therefore the need for disk accesses can be avoided.

In terms of practical applications, all the information collected by our proposal is useful for assessing the quality of care. For example, determining if a resident is being neglected. Also, exploiting trajectory data would enable knowing the contacts among people in the residence, which is of special interest in the current pandemic scenario.

2 RELATED WORK

We explore some relevant representations for trajectories and present RePair, the basis of our compressed representation.

2.1 Representation of trajectories

State-of-the-art works may be classified into those which aim at indexing objects in space [6] and those which focus on representing the actual trajectories of each moving object [7]. The former are fast at obtaining who was in a given area at a given time, the latter are able to retrieve full trajectories quickly.

Another classification may be the usage of indoor or outdoor trajectories; each have different necessities, and they differ in both movement modeling and technologies used [6]. One of the main divergences is that indoor trajectories do not usually use coordinates to represent positions but a set of labeled cells instead. Each cell is defined as the smallest organizational unit of indoor space (rooms, corridors, etc.) [5]. This cellular space enables inherent valuable semantic properties to the tracked routes [7] (e.g. if a resident spends 40 minutes in the dining room we can assume he/she was having lunch/dinner).

One characteristic all kinds of trajectory-related proposals have in common is the concern about their high rates of space consumption [11]. Sometimes, mobile objects follow rather regular movement patterns (e.g. a bus following a given route). Using grammar compression seems a promising way to deal with the inherent repetitiveness of the trajectories. A recent work [1] used RePair [9], a well-known grammar-based compressor, to store trajectories from ships while still supporting spatio-temporal queries efficiently.

2.2 The RePair algorithm

Given a sequence of symbols S, RePair repeatedly replaces repeating pairs of symbols from S by just one symbol, hence shortening S without losing any information. It operates as follows:

(i) It takes the most frequent pair of symbols $\alpha\beta$ within S.

(ii) It replaces $\alpha\beta$ by a new symbol A along S and adds a new rule $A \rightarrow \alpha, \beta$ to RePair's grammar R.

(iii) The previous two steps are repeated recursively until there are no repeated pairs in S. The final sequence (C), can contain both original symbols (terminals) and new symbols (non-terminals).

A simple solution to speed up access to the original data is to provide synchronization between C and S, so that, for some (regular) positions x of S we keep the actual position in C of the phrase that will permit to retrieve the source symbols from $S[x]$ on.

In addition, other common solution to efficiently process the compressed data is to add extra information to the grammar rules R. In [1], they compressed the relative movements of objects in a grid along time with RePair, and used extra information to efficiently compute the aggregated relative movement associated to each rule. In our proposal, we present an enhanced Repair-based technique that provides pseudo-random access capabilities and also includes additional information attached to the grammar rules to boost decompression speed.

3 A SYSTEM ARCHITECTURE FOR THE DETECTION OF TRAJECTORIES

The concern of our project was to develop a system capable of automatically detecting activities within the nursing home. Since activities in an enclosed space are often related to a time and place, we track the movements of all caregivers and residents and store them as trajectories. From these trajectories, we can infer activity information. For example, if a resident is in the dinning room at the lunch time, we can deduce that he/she is "*having lunch.*"

We propose a system architecture using Bluetooth Low-Energy (BLE) technology, which succeeded at indoor positioning detection in different contexts [3, 12]. An Indoor Positioning System (IPS) governs the system. This is a network composed of different wireless devices for retrieving the positioning information of individuals. In our case, those devices are the *beacons* or transmitters carried by people, and the *receivers* located in every room. The positioning information is represented using unique location identifiers associated to the receivers. If a beacon is detected by more than one receiver, the stored location corresponds to the receiver that gets the most powerful signal.

Thus, our system obtains the trajectory information for any given person by concatenating the identifiers of the locations they traversed along time. This trajectory information will be the sole input for our compact representation, meaning that, as long as we can obtain a sequence of location identifiers, the specific technologies used to gather them are replaceable.

4 OUR COMPRESSED REPRESENTATION FOR TRAJECTORIES

Assuming that the location for each person is known along regular intervals of time, this section details how our compressed representation (*indRep*) to store those trajectories is built. Then, we also show the main queries we have implemented.

4.1 *indRep* construction

indRep relies primarily on RePair compression, however, we also improve upon the RePair capabilities by creating additional structures to both support partial daily-based decompression and some basic information retrieval capabilities. These structures would additionally speed up the full decompression process by skipping the need for expanding the non-terminals that generate a sub-sequence containing only repetitions of a unique symbol (i.e. representing the same location along several time instants).

We will explore in detail the different steps taken to create these structures. Figure 1 shows an example and the different components of the proposed representation.

First, people's locations along d days are gathered and concatenated to generate a unique sequence S, which is compressed with RePair. Daily boundaries should be considered to avoid choosing pairs that involve data from two different days. This enables knowing the position in C associated with the initial location of each person and day.

In practice, the data corresponding to caregivers and residents is handled separately in two *indRep* structures. For simplicity, this section focus only on the representation of the residents' locations, as the construction of the data structure for the caregivers is analogous. In the example of Figure 1, S is composed of the trajectories of the four residents.

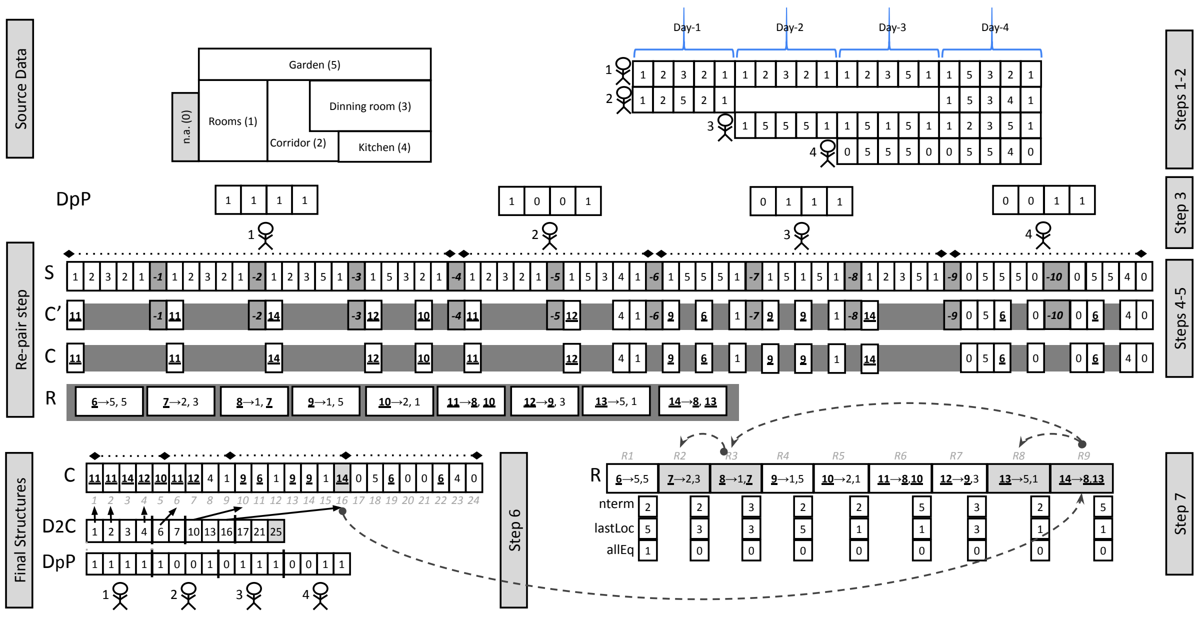

Figure 1: Structures involved in the creation of the *indRep* representation for a nursing home with $r = 4$ residents and $n_l = 5$ locations. Our example considers the data obtained during four days ($d = 4$), using $T = 5$ time intervals within each day.

The construction of the *indRep* representation involves the following steps:

Step 1 Time is discretized into fixed-size intervals, such that each day contains T time intervals. Therefore, there are at most $d \cdot T$ time intervals, where d is the number of days tracked. In the top part of Figure 1 it can be seen how all days are divided into 5 intervals.

Step 2 One location, consisting in a room identifier, is kept for each person at each time interval. If the location is unknown for some period due to a tracking error or a non-scheduled temporal absence from the premises, such location is tagged as "n/a." This is the case of some of the locations of the resident with the identifier 4 in Figure 1.

This is handled separately from the case where a person is not within the facilities during a whole day, hence avoiding to represent the corresponding locations. This case is discussed in *Step 3*.

Considering we have n_l possible locations, an integer identifier $l_i \in \{1, 2, ..., n_l\}$ is assigned to them. In addition, $l_0 = 0$ is reserved for unknown "n/a" locations. Note than in the example depicted in Figure 1 we have $n_l = 5$ locations.

Step 3 Assuming r residents, the trajectory belonging to a given resident i ($1 \le i \le r$) along time constitutes a sequence S_i that contains $T \cdot d_i$ values, being $d_i (1 \le d_i \le d)$ the number of days when resident i stayed within the nursing home. Given that not every resident is present every day at the nursing home, to keep track of the actual days that a resident was present, a bitvector $DpP_i[1..d]$ (up to d *Days-per-Person*) is used, where a 1 is set at position $DpP_i[j]$ to indicate resident i was tracked during the j-th day (0 otherwise). There must be d_i *ones* set in DpP_i. Note that $DpP_2[1..4] = \langle 1001 \rangle$ indicates that location data exists for the resident 2 only during *day-1* and *day-4*.

Step 4 The previous step is repeated for each resident to create a unique sequence $S[1..n]$ by concatenating the sequences S_i ($1 \le i \le r$) corresponding to the locations of each resident i.

Also, the bitvectors DpP_i are concatenated too to create a unique bitvector $DpP[1..d \cdot r]$. They are illustrated in the middle-top part of Figure 1.

Let us define $N_1 = \sum_{i=1}^{r} d_i$ (i.e. number of *ones* in DpP). Note that *indRep* is tracking T locations per each of the N_1 total days with presence of the residents, so the length of S is $n = N_1 \cdot T$.

Step 5 S is compressed using RePair. In order to facilitate future accesses to the data, we need to avoid the re-repairing process to create pairs involving data from two consecutive days. To achieve this, a simple trick consists in adding unique values $u_i = -1 \cdot i$ after appending the T locations corresponding to the i-th day of information ($1 \leq i \leq N_1$) that is included into S.

Then, RePair is applied and, afterwards, we remove those unique separators from the compressed sequence C'. After this step, a final compressed sequence C and a set of rules R are created. Recall C can contain both original symbols (terminals) from S and non-terminal values corresponding to the left-hand side of the rules from R.

The RePair process is illustrated in the middle part of Figure 1. Note that since the values from S range within [0..5], they correspond to terminal symbols in C, and thus non-terminals become values ≥ 6. In the Figure, the latter appear bold-faced and underlined in both C and R.

Step 6 Aligned with the N_1 *ones* in DpP, a new structure $D2C[1..N_1]$ that stores N_1 pointers to the starting positions within C of each of those daily trajectories is added. This allows for partial decompression of the information of a single day for any given resident.

Let us assume we want to retrieve the locations of the resident 3 ($i = 3$) during *day-2* ($j = 2$). First, we count the number of ones until the position $p = (i-1) \cdot d + j$ in DpP. This operation is called $rank_1(DpP, p)$. So $rank_1(DpP,\ (3-1) \cdot 4 + 2) = 7$. We have that $D2C[7] = 10$, and therefore *day-2* of resident 3 starts at $C[10]$. Since $D2C[8] = 13$ points to the starting position of the next trajectory in C, we know that the searched trajectory is represented between positions 10 and 12 of sequence C. Hence, being $C[10..12] = \langle \underline{\mathbf{9}}\underline{\mathbf{6}}1 \rangle$, we finally recover locations $\langle 15551 \rangle$ by expanding rules $R_4 : \underline{\mathbf{9}} \rightarrow 1, 5$ and $R_1 : \underline{\mathbf{6}} \rightarrow 5, 5$; and finally gathering the terminal value $1 = C[12]$.

Step 7 Finally, additional data was kept for each rule from R to speed up the expansion of non-terminal symbols (see bottom-right part of Figure 1), and to skip the processing of some of these non-terminal symbols when looking for the location at a time instant within a day. In practice, for each rule $R_i : A \rightarrow X, Y$ there are three values *nterm*, *lastLoc*, and *allEq* that respectively indicate the number of terminals A expands into (this can be used to avoid expanding some non-terminals at decompression time), the last one of those terminals, and whether all the terminals recovered are either identical.

Looking at Figure 1, note that rule $R_1 : \underline{\mathbf{6}} \rightarrow 5, 5$ expands into only 2 identical terminals $\langle 55 \rangle$. Therefore, we set values $nterm[1] = 2$, $lastLoc[1] = 5$, and $allEq[1] = 1$ (*true*). However, rule $R_6 : \underline{\mathbf{11}} \rightarrow \underline{\mathbf{8}}, \underline{\mathbf{10}}$ recursively expands into 5 terminals $\langle 12321 \rangle$, thus $nterm[6] = 5$, $lastLoc[6] = 1$, and $allEq[6] = 0$ (*false*).

4.2 Supporting random access and partial decompression

Two low-level operations were implemented for *indRep*, one (`syncC`) to access the beginning of any given day on C that allows for partial decompression, and the other (`expand`$^+$) to expand symbols from C. The later operation is boosted with respect to the regular expansion of non-terminals in the regular RePair by exploiting the information added to the rules.

- $[l, r] \rightarrow$ `syncC`$(user, day)$: Given a resident and a day when he/she was present, it returns the starting and ending positions of the slice of C that store the compressed locations for that user and day. As discussed in Step 6, to solve $[l, r] \rightarrow$ `syncC`$(user, day)$ we compute $l \rightarrow D2C[rank_1(DpP, (user-1) \cdot d + day)]$ and $r \rightarrow D2C[rank_1(DpP, user-1) \cdot d + day + 1)] - 1$.

- $str \leftarrow$ `expand`$^+(s)$: Given a symbol s from C it recovers str, that is, the subsequence of terminals s expands into. If s is a terminal, $str \leftarrow \langle s \rangle$ is returned. If s is a non-terminal, it is associated with the rule $R_j : s \rightarrow x, y$, being $j = s - n_l$ (recall terminals are numbered $0..n_l$). Then, $eq = allEq[j]$ has to be checked. If $eq = 1$, we return $\langle aa \ldots a \rangle$, where $a = lastLoc[j]$ and the length of str is $nterm[j]$. Otherwise, we recursively call `expand`$^+$ on x and y (i.e. $str \leftarrow$ `expand`$^+$`(x)` $\cdot$ `expand`$^+$`(y)`).

For an example of `expand`$^+$, assume s is the non-terminal $\underline{\mathbf{6}}$. Therefore we have that j is $s - n_l$, with $n_l = 5$, so $j = 1$. $R_1 : \underline{\mathbf{6}} \rightarrow 5, 5$. This expands into two identical terminals with $allEq[1] = 1$. The execution then returns $\langle 55 \rangle$ because $nterm = 2$ and $lastLoc = 5$.

Next, we describe a more in-depth example step-by-step. Refer to the dashed arrows at the bottom of Figure 1 to follow the rules of the grammar involved in the execution.

Suppose $s = \underline{\mathbf{14}}$. Since s is a non-terminal, $j = s - n_l$ is calculated. In this case $j = 9$. The associated rule is $R_9 : \underline{\mathbf{14}} \rightarrow \underline{\mathbf{8}}, \underline{\mathbf{13}}$ with $allEq[9] = 0$. Next, `expand`$^+$ is executed to expand both the non-terminals $\underline{\mathbf{8}}$ and $\underline{\mathbf{13}}$.

(i) Now, $\underline{\mathbf{8}}$ is a non-terminal associated to the rule $R_3 : \underline{\mathbf{8}} \rightarrow 1, \underline{\mathbf{7}}$ $(j = 8 - 5 = 3)$ with $allEq[3] = 0$. It is then necessary to call `expand`$^+$ again on 1 and $\underline{\mathbf{7}}$. Since 1 is a terminal, the call to `expand`$^+$`(1)` returns $\langle 1 \rangle$. On the other hand, $\underline{\mathbf{7}}$ is again a non-terminal associated with the rule $R_2 : \underline{\mathbf{7}} \rightarrow 2, 3$ $(j = 7 - 5 = 2)$ with $allEq[2] = 0$. Subsequent calls to `expand`$^+$ return $\langle 2 \rangle \leftarrow$ `expand`$^+$`(2)` and $\langle 3 \rangle \leftarrow$ `expand`$^+$`(3)`, as both 2 and 3 are terminals. Therefore, we obtained $\langle 123 \rangle \leftarrow$ `expand`$^+$$(\underline{\mathbf{8}})$.

(ii) Now, we need to expand the symbol $\underline{\mathbf{13}}$. Note that $\underline{\mathbf{13}}$ is a non-terminal associated with the rule $R_8 : \underline{\mathbf{13}} \rightarrow 5, 1$ $(j = 13 - 5 = 8)$ with $allEq[2] = 0$. Therefore, `expand`$^+$ on 5 returns $\langle 5 \rangle$ and on 3 $\langle 1 \rangle$. Note that once again, the two values are terminals. To sum up, we obtain that `expand`$^+$$(\underline{\mathbf{14}})$ = `expand`$^+$$(\underline{\mathbf{8}})$ $\cdot$ `expand`$^+$$(\underline{\mathbf{13}})$ $= \langle 123451 \rangle$.

Using these low-level operations, it is possible to support other types of higher-level operations:

- $loc \rightarrow$ `getLocation`$(user, day, t)$: it returns the location of a given $user$ at a particular time instant t of a day.

- $locs \rightarrow$ `getLocations`$(user, day, t_s, t_e)$: It returns all the locations within the time-interval $[t_s, t_e]$. This operation is similar to `getLocation`, yet, instead of simply recovering a single terminal corresponding to the location at time instant t_s, the execution continues from there on until $t_e - t_s + 1$ terminals are recovered.

The full procedure to solve `getLocation` is as follows:

(i) First, we obtain $[l, r] \rightarrow$ `syncC`$(user, day)$. Now $C[l..r]$ is the slice of C corresponding to day. We set $p \rightarrow l$ and $offt \rightarrow 0$.

(ii) We need to move p forward until reaching the position containing the target location at time t. For each step we either either set $nt = 1$ if $C[p]$ is a terminal, or $nt = nterm[C[p] - n_l]$ otherwise.

(iii) Then, if $(nt + offt < t)$ we set $p \rightarrow p + 1$, and $offt \rightarrow offt + nt$, otherwise the traversal stops.

(iv) Now it is necessary to expand (with `expand`$^+$) the symbol $C[p]$, to gather its k-th terminal, having $k \rightarrow t - offt$.

Note that to solve `getLocations` we would assume $t = t_s$ and recover the next $t_e - t_s$ locations form there on. These two operations greatly benefit from the capability of skipping steps of the decompression process. For example, let us assume the subsequence $\langle 15555552 \rangle$ within the original sequence S, where all the 5s have been compressed into a single non-terminal $\underline{\mathbf{X}}$ where the extra information corresponding to the rule is: $nterm = 6$, $lastloc = 5$, and $allEq = 1$. The compressed representation of this subsequence would be $\langle 1\underline{\mathbf{X}}2 \rangle$. In this example, the decompression process would read $\underline{\mathbf{X}}$ and, after checking that it is composed of the same value repeated six times, it skips all the recursive calls to expand, recovering $\langle 555555 \rangle$. The decompression continue by jumping straight to the next symbol.

5 EXPERIMENTAL DATA AND RESULTS

We present the experiments performed to compare our proposal with other popular compression techniques.

5.1 Experimental data

As the real installation of Bluetooth devices was delayed because of the pandemic, we used synthetic data for our experiments. Instead of using existing data generators for indoor trajectories, such as Vita [10], we developed our own generator to create ad-hoc datasets according to the typical patterns of a nursing home. We focused on creating synthetic trajectory data for our specific context, using information about the actual schedules, allocated rooms, and space information of the nursing home we collaborated with for this work.

A total of 198 different locations associated to the receivers in the nursing home were mapped to a graph where the nodes represent the locations, and the edges link the connected locations. This graph defines every possible movement and we can use it as base for simulating the positioning data in the nursing home across any given amount of time.

We considered 34 caregivers and 69 residents, according to the current reality of the nursing home. For every minute the movements of every person were calculated based on their definition in a JSON database. Such information considered:

- For the caregivers: their real schedules (considering three work shifts: morning, afternoon, and night) were used as a template to define their daily duties and work hours.
- For the residents: three profiles were defined: *quiet*, *wanderer*, and *dependant*. These profiles informed of the tendency for the residents' movements. *Quiet* residents tend to favor one location. *Wanderer* residents would stay still if they are in a favored location, but have a tendency to wander aimlessly when in any other. A random-walk algorithm was used to simulate the wandering. Lastly, *dependant* residents cannot move on their own and need the help of a caregiver.

In practice, the raw data being compressed consist of a data file and a headers file. The data file is a sequence of 32-bit integers, where we only store the locations for those time instants when each person is inside the facilities. The headers file is a text file where each line represents a trajectory and has the format {*person-id, day, start-time, end-time*}. This allows us to establish the boundaries of the data for each person and day within the first file.

For our experiments, we generated two synthetic datasets using a simulation with 103 people (with residents being almost permanently in the building and caregivers working 8-hours shifts) that involved 2,000 days, assuming the locations of our collaborating nursing home where there are 198 locations. Therefore, the location identifiers included in the generated trajectories are represented with just one byte; i.e. $\lceil \log_2 198 \rceil = 8$ bits. Our datasets differ in the repetitiveness of the data being generated. In the first dataset, *low-rep dataset*, a day is composed of 1,440 time

Table 1: Experimental result of the comparisons

Method	Compression ratio (%)		Compression speed (in MB/s)		Decompression speed (in MB/s)		Retrieval time of a day (in μs)		Retrieval time of 30 locations (in μs)	
	low-rep dataset	high-rep dataset	low-rep dataset	high-rep dataset	low-rep dataset	high-rep dataset	low-rep dataset	high-rep dataset	low-rep dataset	high-rep dataset
7zip	6.57	1.85	23.60	46.00	340.98	104.93	–	–	–	–
gzip	8.28	2.21	88.38	145.83	322.44	124.63	–	–	–	–
dictzip	8.67	–	29.80	–	644.85	–	282	–	640	–
LZ-End	24.94	–	2.51	–	5.98	–	257	–	13	–
RePair	12.16	1.91	41.77	36.15	106.92	106.49	–	–	–	–
RePair (SDSL)	7.58	1.26	40.04	35.50	54.60	54.08	–	–	–	–
indRep	14.98	2.40	41.16	34.78	111.93	123.86	49	231	26	27
indRep (SDSL)	8.54	1.44	37.66	33.17	86.81	78.01	65	292	24	26

instants (intervals of 1 minute). Its overall size is 236.4 MB, from which the raw trajectories occupy 233.3 MB, and the headers file 3.1 MB. The second dataset, *high-rep dataset*, involves days of 17,280 time instants (intervals of 5 seconds), hence making the data more repetitive. The size of the dataset is 2803.1 MB (data file of 2800 MB and headers file of 3.1 MB).

5.2 Experimental results

We ran experiments to measure the compression ratio, compression performance, and decompression speed of *indRep* both when a whole dataset is decompressed or when we simply retrieve a 1-day trajectory or just 30 consecutive locations for a given person from a whole dataset. We compared them with the generic implementation of RePair in order to show the boost in performance we obtain. Note that both RePair and *indRep* use internally 32-bit integer arrays among their components. Using the C++ SDSL library [2], we implemented variants were those arrays are replaced by bit-wise integer vectors where each integer is encoded with just $\lceil log_2 M \rceil$ bits, being M the largest value. This saves space, yet leads to a loss of performance, as bit-wise operations are required to recover the original 32-bit integers.

Given that, to the best of our knowledge, there is no specific trajectory compressor that permits to handle cellular-based trajectories, we compared our results with those achieved by some well-known general-purpose compressors that can work over the trajectories of objects represented as sequences of location identifiers, namely gzip, a linux utility for compression based on the use of LZ77 [13], as well as P7zip, another linux utility based on LZMA, an improved version of the LZ77.

One limitation of RePair, gzip, and P7zip is that they do not allow for random decompression, i.e. recovering a random snippet from the source data (e.g. a 1-day trajectory) requires starting decompression from the beginning. This is unfair when we compare them with *indRep*, which can be seen as an improved version of RePair where we can gather the actual starting location for each person at any given day within the compressed data. This led us to include in our comparisons two additional techniques: LZ-end [8], a LZ77-like algorithm that allows for random decompression, and dictzip, a linux utility that uses the LZ77 algorithm to compress data in 64 KiB blocks and allows for block-wise pseudo-random access to the source data.

Our experiments were run on an Intel(R) Core(TM) i5-9600K with 16 GB of RAM, running Ubuntu 18.04.4 LTS, and g++ version 7.4.0, using C++17 (-std=c++17). The source code for all the implementations is available at our git repository (`https://gitlab.lbd.org.es/lbd-open/indrep`).

Table 1 details, in columns 2-3, the compression ratios achieved for all the techniques while considering the two implementations for RePair and *indRep*. The compression ratio is computed as the percentage that the compressed representation occupies with respect to the raw data. As expected, the bit-wise implementations of both RePair and *indRep* show improved compression ratios with respect to the the plain implementations. For the low-rep dataset, 7zip achieves the best compression ratio, and the generic bit-wise RePair implementation comes right behind it (around 7.58%), with our bit-wise *indRep* implementation yielding 8.54%, lagging behind gzip. Also, our bit-wise solution overcomes dictzip. LZ-End obtains the worst compression ratio (24.94%).

When using the high-rep dataset, LZ-End and dictzip failed at compression.The other solutions showed a clear improvement in compression ratio, with both the SDSL-enhanced *indRep* and RePair overcoming the LZ-based alternatives. This rather expected improvement is due to both the grammar-based and the LZ-based compressors can exploit the long repetitive subsequences that occur in high-rep dataset to gain compression.

Columns 4–7 show the compression and decompression speeds (in Mb/s) measured on both datasets. LZ-End obtains the worst compression speed (2.51 Mb/s) on the low-rep dataset, followed by 7zip (11.94 Mb/s) and then by dictzip (16.67 Mb/s). The rest of solutions yield faster speeds, with the RePair-based techniques (including our *indRep*) obtaining values around 37–41 Mb/s on the low-rep dataset. Gzip is the fastest compressor on both datasets.

Regarding decompression, on the low-rep dataset dictzip is the fastest technique (282 Mb/s) and LZ-End is by far the slowest one (5.98 Mb/s). *indRep* achieves 249.79 Mb/s with its plain C++ implementation. However, that result drops to 156.05 Mb/s when using the bit-wise implementation. Both RePair and *indRep* draw a clear drop in performance when using their bit-wise SDSL variant. This is because both the C array and the grammar rules are processed bit-wise each time a expansion of a symbol is performed. In addition, we can see that RePair performs worse than our proposal because *indRep* can skip some recursive expansions of non-terminals during the decompression process.

For both datsets, the compression and decompression speeds are similar for the RePair-based solution, however, the LZ-based solutions showcase an improvement in compression speed for the high-rep dataset, but also achieve worse decompression speeds.

Finally, we show the average time required to retrieve the locations of a person within both a 1-day slice (columns 8-9), and a 30-locations slice (columns 10-11). Only the solutions with random access to the compressed sequence were compared to *indRep* when possible. The results show that *indRep* is faster than the others when retrieving a slice of the source sequence of locations. Only LZ-End can beat our implementations when recovering 30-locations slices.

6 CONCLUSIONS AND FUTURE WORK

We have presented the basis for an automatic indoor positioning system for a nursing home focusing on a novel compact representation of trajectories: *indRep*.

We based our proposal in RePair, which is known to yield great compression on highly repetitive data, but also included some enhancements over the regular RePair by annotating the grammar rules with additional information, as well as using additional data structures. Thanks to these enhancements, we were able to reduce the number of rule expansions, which boosts decompression performance, and allows for fast decompression of any given slice of data. This partial decompression proved faster than the other techniques tested. In summary, we conclude that our proposal is a good alternative for managing trajectory data. Note that this is not only about reducing storage space, but also about improving performance when the uncompressed data does not fit into main memory but the compressed one does.

As future work, we will tackle online compression so that we can append new data without recompressing the whole data from scratch. A simple solution would be to reuse the grammar rules and recompress only regularly to avoid loss of compression effectiveness. Yet reorganizing the internals of *indRep* to handle data daily-wise rather than person-wise would also help. Another important future-work task is to feed our solution with data from the actual indoor positioning system of our collaborating nursing home. This would permit us to validate that *indRep* maintains its results when working under real-world conditions.

Finally, we are also interested in extending the functionality of our representation to support detecting contacts among people, which not only is of special interest in the current pandemic scenario, but also carries the potential to enhance the internal processes of the nursing home such as polishing caregiver's schedules, as well as improving the treatment of residents such as verifying that no resident was untenanted along the day.

7 ACKNOWLEDGMENTS

Partially funded by the CITIC research center funded by Xunta/FEDER-UE 2014-2020 Program, grant ED431G 2019/01.

MICIU(PGE/ERDF) [MaGIST: PID2019-105221RB-C41; Datos 4.0: TIN2016-78011-C4-1-R; STEPS: RTC-2017-5908-7; BIZDEVOPS: RTI2018-098309-B-C32]. MICINN(PGE/ERDF) [EXTRA-Compact: PID2020-114635RB-I00]. Xunta/GAIN(ERDF) [CEDCOVID: COV20/00604; GRC: ED431C 2021/53] and by FPI Program [BES-2017-081390].

References

[1] Nieves R. Brisaboa, Adrián Gómez-Brandón, Gonzalo Navarro, and José R. Paramá. GraCT: A grammar-based compressed index for trajectory data. *Inf. Sci.*, 483:106–135, 2019.

[2] Simon Gog, Timo Beller, Alistair Moffat, and Matthias Petri. From theory to practice: Plug and play with succinct data structures. In *Proc. of SEA*, pages 326–337, 2014.

[3] Jun-Ho Huh and Kyungryong Seo. An indoor location-based control system using bluetooth beacons for iot systems. *Sensors*, 17(12):2917, 2017.

[4] Ibernex: Helpnex, 2014 (accessed June, 2020). https://ibernex.es/en/helpnex/.

[5] IndoorGML OGC, 2014 (accessed June, 2021). http://www.indoorgml.net/.

[6] Christian S. Jensen, Hua Lu, and Bin Yang. Indexing the trajectories of moving objects in symbolic indoor space. In *Proc. of SSTD*, pages 208–227, 2009.

[7] Alexandros Kontarinis, Karine Zeitouni, Claudia Marinica, Dan Vodislav, and Dimitris Kotzinos. Towards a semantic indoor trajectory model: application to museum visits. *GeoInformatica*, 25(2):311–352, 2021.

[8] S. Kreft and G. Navarro. Lz77-like compression with fast random access. In *Proc. of DCC*, pages 239–248, 2010.

[9] N. Jesper Larsson and Alistair Moffat. Offline dictionary-based compression. In *Proc. of DCC*, pages 296–305, 1999.

[10] Huan Li, Hua Lu, Xin Chen, Gang Chen, Ke Chen, and Lidan Shou. Vita: a versatile toolkit for generating indoor mobility data for real-world buildings. *Proceedings of the VLDB Endowment*, 9:1453–1456, 09 2016.

[11] Kai-Florian Richter, Falko Schmid, and Patrick Laube. Semantic trajectory compression: Representing urban movement in a nutshell. *J. Spatial Inf. Sci.*, 4(1):3–30, 2012.

[12] Sebastian Sadowski and Petros Spachos. Rssi-based indoor localization with the internet of things. *IEEE Access*, 6:30149–30161, 2018.

[13] Jacob Ziv and Abraham Lempel. A universal algorithm for sequential data compression. *IEEE Trans. Inf. Theory*, 23(3):337–343, 1977.