Representation and Exploitation of Event Sequences

Autor: Tirso Varela Rodeiro

Tesis doctoral UDC / 2020

Directores:
Antonio Fariña Martínez
Miguel Rodríguez Luaces

Representation and Exploitation of Event Sequences

Autor: Tirso Varela Rodeiro

Tesis doctoral UDC / 2020

Directores:
Antonio Fariña Martínez
Miguel Rodríguez Luaces

PhD thesis supervised by
Tesis doctoral dirigida por

Antonio Fariña Martínez
Departamento de Computación y Tecnologías de la Información
Facultad de Informática
Universidade da Coruña
15071 A Coruña (España)
Tel: +34 981 167000 ext. 1352
Fax: +34 981 167160
`fari@udc.es`

Miguel Rodríguez Luaces
Departamento de Computación y Tecnologías de la Información
Facultad de Informática
Universidade da Coruña
15071 A Coruña (España)
Tel: +34 981 167000 ext. 1254
Fax: +34 981 167160
`luaces@udc.es`

Miguel Rodríguez Luaces y Antonio Fariña, como directores, acreditamos que esta tesis cumple los requisitos para optar al título de doctor internacional y autorizamos su depósito y defensa por parte de Tirso Varela Rodeiro cuya firma también se incluye.

A Víctor Solís,

seguimos yendo, amigo; seguimos yendo.

Para el niño, enamorado de mapas y estampas, el universo es igual a su vasto apetito.

Charles Pierre Baudelaire

El mapa no es el territorio.

Alfred Korzybski

No creo que el mundo haya mejorado gracias a nosotros, tampoco creo que nadie llore nuestra muerte, no hemos realizado muchas buenas acciones... pero, ¿cuánta gente ha viajado lo que nosotros y visto lo que nosotros?

Peachy Carnehan / Michael Caine

Beati hispani quibus vivere est bibere.

Julius Caesar Scaliger

Acknowledgements

I cannot begin this thesis without making a brief aside to remember all those people who have contributed to such an event. I feel compelled to share the credit (or blame) of this big step for man and insignificant leap for mankind.

Following the footsteps of Proust, I'll start from the beginning. The doctoral diploma will also belong to María de los Ángeles and José Antonio; authors of my days, and consequently, as authors of this research as I am.

Obviously, acknowledgements extend to the different branches —and twigs (Mael and Catalina)— of my lush family tree, paying special attention to the linage of Grandal and the D'Vinte dynasty. There are three gentlemen whose role during this long journey deserve a special mention: Francisco Rodeiro, source of inspiration during the process; José Vasco, provider of a reliable steed when my chores took me to the green meadows of Pocomaco and doctor Lorenzo Varela, family pioneer in the study of users flow in public transport and permanently exiled on the Scandinavian coasts because of it.

Once the blood ties are dispatched I must speak of the spinal cord of that brief period of time where everything had a golden glow under the protection of a rubber fortress. The names of the heroes and heroines of those deeds still echo between the walls of my heart: Noa, Dani, Iván, Gonzalo, Patri, Víctor, Anxo, Ana...

The next point is a sneak peek of the research introduced in this thesis, an exercise on aggregated compression of subjects that should not be forgotten: liceists, vallisoletanos, marines, australians, the IT crowd

and people from Canterbury. In addition to Rebeca Vegara, because of that issue that cannot be named.

Of course, I am grateful to the Databases Lab for supporting me along the path, highlighting the support of Miguel Luaces, Antonio Fariña and Nieves Brisaboa since without them this thesis would have never been possible. Besides them, every member of the clan deserves to be on this page, from Carlos and the Enxenio Football Team to Daniil; with special mention to the actors of the Caceres cabin: Fernando, the idyll from Tenerife; David, the apostate; Álex, the inca explorer and Adrián, from the Anxeriz party commission.

I would also like to thank Diego Seco, María Andrea Rodríguez-Tastets, Hideo Bannai and Andrew Turpin who opened the gates of their kingdoms, gave me the keys to their cities and allowed me to establish a base camp where I could feel safe in the ends of the world. Other important international mentions are: Gonzalo Navarro, Nicola, Gilberto, Carlos, Claudio, Pedro, Isyed, Jose, Diego Díaz and —mainly— Sindy (still my wife within the glorious state of Nevada).

An allusion to the Polish command is mandatory: Emilio, Toni, Zas and Montesquieu The Rascal. With them I achieved that aura of invincibility that governed this journey from Szewska Street in Krakow to that hangar on the outskirts of Melbourne.

Last, and therefore most important, appreciation for the usual gang: Borja, outgoing and womanizer; Cristina and Souto, honest dancers; Luís, spider-man; Felipe and Dani, the Fuji-san advocates; Jose and Fai, the pirate kings; Canedo and Martín, spark plug mechanics; Money and Quel, the flying dutchwomen and Castellanos, the poet. Nevertheless, from this last group there are five people who deserve to top this long list of supporters:

- **Eva V. F.**, the sibyl able to read polylactic acid phalanges.
- **Andrea F. A.**, the voice of reason during this Odyssey.
- **Pablo R. P.**, the man who finally walked the 1000 miles.
- **Ana M. X.**, the sister who drowned her twin in the Danube.
- **Armando C. P.**, the light that illuminated my route worldwide.

Agradecimientos

No puedo comenzar esta publicación sin pararme un breve momento a recordar a todas aquellas personas que han contribuido a tal evento, sintiéndome obligado a compartir el mérito (o la culpa) de este gran paso para el hombre e insignificante paso para la humanidad.

Siguiendo los pasos de Proust, comenzaré por el principio. El diploma doctoral pertenecerá honoríficamente a María de los Ángeles y José Antonio, autores de mis días; y en consecuencia, tan autores de esta investigación como yo mismo.

Evidentemente, los agradecimientos se extienden a las diferentes ramas —y ramitas (Mael y Catalina)— de mi frondoso árbol genealógioco, prestando especial atención al linaje de los Grandal y la dinastía D'Vinte. De entre todos ellos, me gustaría hacer destacar tres caballeros que marcaron una pequeña diferencia durante este arduo camino. El primero, Franciso Rodeiro, fuente de inspiración durante este proceso. El segundo, José Vasco, proveedor de un fiable corcel cuando mis quehaceres me llevaron hasta los verdes prados en los aledaños de Pocomaco. Por último, el doctor Lorenzo Varela, pionero familiar en adentrarse en el estudio del flujo de usuarios en transporte público y exiliado por ello de forma permanente en las costas escandinavas.

Una vez despachados los lazos de sangre debo hablar de la médula espinal de aquel breve lapso de tiempo donde todo tenía un resplandor dorado al amparo de una fortaleza de caucho. Los nombres de los héroes y heroínas de aquellas gestas aún retumban entre las paredes de mi cabeza: Noa, Dani, Iván, Gonzalo, Patri, Víctor, Anxo, Ana...

Lo siguiente es un avance forzoso de la investigación aquí presentada, un ejercicio de compresión agregada de sujetos que no deben caer en el

olvido: liceistas, vallisoletanos, militares, australianos, informáticos y gentes de Canterbury. Amén de Rebeca Vegara, por lo de aquella vez.

Por supuesto, debo agradecer al Laboratorio de Base de Datos el haber permitido esta hazaña, destacando el apoyo de Miguel Luaces, Antonio Fariña y Nieves Brisaboa sin el que esta tesis nunca hubiese sido posible. Además de ellos, cada miembro del clan merece estar en esta página, desde Carlos y el Enxenio F.C. hasta Daniil; con mención especial a los actores del camarote cacereño: Fernando, el idilio tinerfeño; David, el apóstata; Álex, el inca y Adrián, de la comisión de fiestas de Anxeriz.

Gracias también a Diego Seco, Maria Andrea Rodríguez-Tastets, Hideo Bannai y Andrew Turpin que me abrieron las puertas de sus reinos, me entregaron las llaves de sus ciudades y me permitieron establecer un campamento base donde sentirme seguro en los confines del mundo. Otras menciones internacionales de importancia son: Gonzalo Navarro, Nicola, Gilberto, Carlos, Claudio, Pedro, Isyed, Jose y —principalmente— Sindy (todavía mi mujer en el estado de Nevada).

Es obligatoria la mención al comando polaco: Emilio, Toni, Zas y el bribón de Montesquieu. Con ellos alcancé ese halo de invencibilidad que rigió esta andanza desde la calle Szewska de Cracovia hasta aquel hangar en las afueras de Melbourne.

Por último, y por ello más importante, reconocer a los de siempre: Borja, sociable y mujeriego; Cristina y Souto, danzarines y honrados; Luís, el hombre araña; Felipe y Dani, amigos de Fuji-san; Jose y Fai, los reyes del pirata; Canedo y Martín, mecánicos de bujías; Money y Quel, las holandesas errantes y Castellanos, el poeta. No obstante, de este último grupo son cinco las personas que merecen rematar esta larga lista de simpatizantes a la causa:

- **Eva V. F.**, la sibila capaz de leer falanges de ácido poliláctico.
- **Andrea F. A.**, la voz de la razón durante (casi) toda esta odisea.
- **Pablo R. P.**, el hombre que finalmente caminó las 1000 millas.
- **Ana M. X.**, la hermanísima que ahogó a su gemelo en el Danubio.
- **Armando C. P.**, la luz que iluminó mi ruta alrededor del mundo.

Abstract

The Ten Commandments, the thirty best smartphones in the market and the five most wanted people by the FBI. Our life is ruled by sequences: thought sequences, number sequences, event sequences... a history book is nothing more than a compilation of events and our favorite film is just a sequence of scenes. All of them have something in common, it is possible to acquire relevant information from them. Frequently, by accumulating some data from the elements of each sequence we may access hidden information (e.g. the passengers transported by a bus on a journey is the sum of the passengers who got on in the sequence of stops made); other times, reordering the elements by any of their characteristics facilitates the access to the elements of interest (e.g. the publication of books in 2019 can be ordered chronologically, by author, by literary genre or even by a combination of characteristics); but it will always be sought to store them in the smallest space possible.

Thus, this thesis proposes technological solutions for the storage and subsequent processing of events, focusing specifically on three fundamental aspects that can be found in any application that needs to manage them: compressed and dynamic storage, aggregation or accumulation of elements of the sequence and element sequence reordering by their different characteristics or dimensions.

The first contribution of this work is a compact structure for the dynamic compression of event sequences. This structure allows any sequence to be compressed in a single pass, that is, it is capable of compressing in real time as elements arrive. This contribution is a milestone in the world of compression since, to date, this is the first proposal for a variable-to-variable dynamic compressor for general

purpose.

Regarding aggregation, a data warehouse-like proposal is presented capable of storing information on any characteristic of the events in a sequence in an aggregated, compact and accessible way. Following the philosophy of current data warehouses, we avoid repeating cumulative operations and speed up aggregate queries by preprocessing the information and keeping it in this separate structure.

Finally, this thesis addresses the problem of indexing event sequences considering their different characteristics and possible reorderings. A new approach for simultaneously keeping the elements of a sequence ordered by different characteristics is presented through compact structures. Thus, it is possible to consult the information and perform operations on the elements of the sequence using any possible rearrangement in a simple and efficient way.

Resumen

Los diez mandamientos, los treinta mejores móviles del mercado y las cinco personas más buscadas por el FBI. Nuestra vida está gobernada por secuencias: secuencias de pensamientos, secuencias de números, secuencias de eventos... un libro de historia no es más que una sucesión de eventos y nuestra película favorita no es sino una secuencia de escenas. Todas ellas tienen algo en común, de todas podemos extraer información relevante. A veces, al acumular algún dato de los elementos de cada secuencia accedemos a información oculta (p. ej. los viajeros transportados por un autobús en un trayecto es la suma de los pasajeros que se subieron en la secuencia de paradas realizadas); otras veces, la reordenación de los elementos por alguna de sus características facilita el acceso a los elementos de interés (p. ej. la publicación de obras literarias en 2019 puede ordenarse cronológicamente, por autor, por género literario o incluso por una combinación de características); pero siempre se buscará almacenarlas en el espacio más reducido posible sin renunciar a su contenido.

Por ello, esta tesis propone soluciones tecnológicas para el almacenamiento y posterior procesamiento de secuencias, centrándose concretamente en tres aspectos fundamentales que se pueden encontrar en cualquier aplicación que precise gestionarlas: el almacenamiento comprimido y dinámico, la agregación o acumulación de algún dato sobre los elementos de la secuencia y la reordenación de los elementos de la secuencia por sus diferentes características o dimensiones.

La primera contribución de este trabajo es una estructura compacta para la compresión dinámica de secuencias. Esta estructura permite comprimir cualquier secuencia en una sola pasada, es decir, es capaz

de comprimir en tiempo real a medida que llegan los elementos de la secuencia. Esta aportación es un hito en el mundo de la compresión ya que, hasta la fecha, es la primera propuesta de un compresor dinámico "variable to variable" de carácter general.

En cuanto a la agregación, se presenta una propuesta de almacén de datos capaz de guardar la información acumulada sobre alguna característica de los eventos de la secuencia de modo compacto y fácilmente accesible. Siguiendo la filosofía de los actuales almacenes de datos, el objetivo es evitar repetir operaciones de acumulación y agilizar las consultas agregadas mediante el preprocesado de la información manteniéndola en esta estructura.

Por último, esta tesis aborda el problema de la indexación de secuencias de eventos considerando sus diferentes características y posibles reordenaciones. Se presenta una nueva forma de mantener simultáneamente ordenados los elementos de una secuencia por diferentes características a través de estructuras compactas. Así se permite consultar la información y realizar operaciones sobre los elementos de la secuencia usando cualquier posible ordenación de una manera sencilla y eficiente.

Resumo

Os dez mandamentos, os trinta mellores móbiles do mercado e as cinco persoas máis buscadas polo FBI. As secuencias gobernan a nosa vida: secuencias de pensamentos, secuencias de números, secuencias de eventos... un libro de historia non é máis que unha sucesión de eventos e o noso filme favorito só é unha secuencia de escenas. Todas elas teñen algo en común, pódese extraer información relevante de todas elas. Ás veces, é posíbel acceder a información oculta acumulando algún dato (p. ex. os viaxeiros transportados por un autobús nun traxecto son a suma dos pasaxeiros que se subiron na secuencia de paradas realizadas); outras veces, a reordenación dos elementos por algunha das súas características facilita o acceso a elementos de interés (p. ex. a publicación de obras literarias no 2019 pode ordenarse cronoloxicamente, por autor, por xénero literario ou incluso por unha combinación de características); pero sempre se buscará almacenalas no espacio máis reducido posíbel.

Por iso, esta tese propón solucións tecnolóxicas para o almacenamento e posterior procesamento de secuencias, centrándose en tres aspectos fundamentais que se poden atopar en calquera aplicación que precise xestionalas: o almacenamento comprimido e dinámico, a agregación ou acumulación de algún dato sobre os elementos da secuencia e a reordenación dos elementos da secuencias polas súas diferentes características ou dimensións.

A primeira contribución deste traballo é unha estrutura compacta para a compresión dinámica de secuencias. Esta estrutura permite comprimir calquera secuencia nunha soa pasada, é dicir, é capaz de comprimir en tempo real a medida que van chegando os elementos da secuencia. Esta achega é un fito no mundo da compresión xa que, ata a

data actual, é a primeira proposta dun compresor dinámimco "variable to variable" de carácter xeral.

En canto á agregación, preséntase unha proposta de almacén de datos capaz de gardar a información sobre algunha característica dos eventos da secuencia de modo compacto e sinxelamente accesible. Seguindo a filosofía dos almacéns de datos actuais, o obxectivo é evitar repetir operacións de acumulación e axilizar as consultas agregadas mediante o preprocesado da información manténdoa nesta estrutura.

Por último, esta tese aborda o problema da indexación de secuencias de eventos considerando as súas diferentes características e posíbles reordenacións. Preséntase unha nova forma de manter simultáneamente ordenados os elementos da secuencia por diferentes características a través de estruturas compactas. Permítese así consultar a información e realizar operacións sobre os elementos da secuencia usando calquera ordenación posíbel dunha forma sinxela e eficiente.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

A long time ago, long before the advent of Kindles and iPads, mankind managed to bend space and time with the invention of clay tablets. This little gadget allowed to enclose an abstract thought in a physical and easily transportable medium. We all know, or at least imagine, the impact caused by this invention in the societies of our ancestors: education, treaties, accounting, letters, legends... Tablets improved over time, from the use of different materials (wood, metal, etc.) to the inclusion of a spatula on the back of the stylus[1] to erase what was written. However, this excellent tool had one significant drawback: very limited writing space. Even if it was written on both sides, long texts could not fit. Therefore, the longer the text to be written, the more tablets were necessary, with the corresponding complications that this entails (prize, weight, more likely to lose any of the tablets leaving a disjointed text...). It would take centuries for the invention of a new mechanism to deal with these obstacles: papyrus. These plant fiber constructions were thin, flexible, light and, despite having only one usable face, a single roll of usual dimensions could contain a complete Greek tragedy. Besides, papyrus can be rolled up, storing a large amount of text in little space. Just like writing has evolved to store more information in less space, the proposals in this thesis continue to battle against spatial limitations by storing data in flexible, thin and compact "virtual papyri" that have additional functionalities such as querying, ordering

[1] The needle used to write over clay tablets.

and aggregating its contents.

The incorporation of papyri into society also brought new dilemmas to be solved; for example, in the Great Library of Alexandria there were so many papyri stored that it was impossible to find a specific work. To solve this problem it was necessary to establish an order and organize somehow the chaotic library. Zenodotus, first director of the library of Alexandria, contributed to the resolution of this problem by naming each part of Homer's poems using alphabetically arranged letters for the first time. A little later, Callimachus, Zenodotus' successor, would invent the first book catalogs, labeling them as "Pinakes". The Callimachus system divided works into six literary genres and five types of prose, and within each category, the works were arranged alphabetically by author. The impact of this small reorganization was such that variations of the original "Pinakes" continued to be used until the 19th century. Following the ideas of the power in reorganization by different dimensions, this thesis uses a similar approach to improve query and search times. Like the work of the librarians of Alexandria, if the information is organized by the appropriate dimensions it will be really simple to find what we are looking for. Thus, we provide a new indexing structure that allows to rearrange information according to the particular needs of different queries and different contexts easing the search.

Centuries later, a cholera outbreak occurred in England in the autumn of 1848, causing great mortality. At that time, the mode of transmission of this disease was not known, confronting two theoretical currents. On the one hand, there were those who argued that cholera was acquired by contact with the patient or with their belongings. On the other hand, there were those who thought that certain atmospheric conditions, especially the winds, transmitted toxic fumes from one place to another. Dr. John Snow, who did not trust any of these theories, set out to find the real cause of the infection; this is how the idea of the popularly known as the "cholera map" arose (Figure 1.1). Following the trail of a severe outbreak in the south of the city, Dr. Snow made a map of the sector, in which he marked the points corresponding to deaths from cholera and the different existing drinking water pumps, graphically demonstrating the spatial relationship between cholera deaths and the

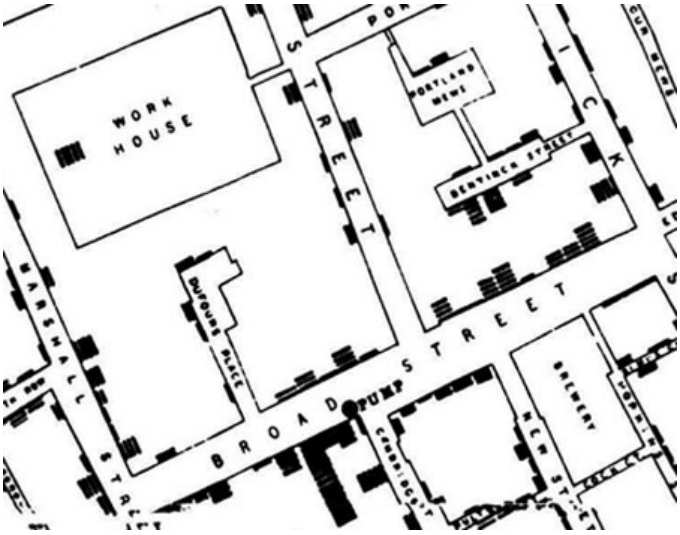


Figure 1.1: Snow's map of cholera deaths in the Broad Street area. The water pump is located at the intersection of Broad and Cambridge Street. Black bars reflect the number of deaths. 1854. (Source: https://johnsnow.matrix.msu.edu/book_images12.php)

Broad Street pump. Finally, the study of the pump showed that 20 feet underground, a sewer pipe passed very close to the water source of the pump and that leaks existed between the two water courses. Thus, in the wake of doctor Snow, systems in charge of gathering and aggregating data to provide hidden information in the individual events are common nowadays. This thesis introduces new approaches to tackle this same problem focusing on aggregating large amounts of data without compromising query speed.

1.1 Motivation

As the introduction implied, this thesis addresses three of the main problems that can be found in the exploitation of general sequences and, particularly, in event sequences: dynamic compression, indexing by multiple dimensions and aggregated data exploitation. An event sequence can be described conceptually as a one-dimensional arrangement

of a series of elements, each of which represents an event that generally occurs at a particular time. In addition to time, we can usually find other dimensions, characteristics or attributes with different values for each event.

Consider, as an example of event sequences, the publications of literary works written in Spain during the Golden Age (16th and 17th centuries). The publication of each work constitutes an event that, in addition to the temporal parameter, has other relevant characteristics such as author or literary genre. This event sequence does not have repeated elements, but it could happen in other types of elements. Another example is the case of a sequence of words in a text to be transmitted, each word can be seen as an event with a transmission time beside other characteristics such as size, syntactic category, etc. Generalizing, an event sequence is an ordered list of elements as:

$$E^1_{a,i,\ldots x}; E^2_{b,j,\ldots y}; E^3_{a,k,\ldots z}; E^4_{c,i,\ldots y}; \ldots$$

Where the superscript identifies the event and the subscripts a, b, c identify its values for characteristic 1, values i, j, k for characteristic 2 and values x, y, z for characteristic n. The previous examples show a property of the sequences, they can have a high degree of repetition that could be exploited by compression techniques. For example, existing compressors for English texts can exploit that some words appear much more often than others reducing the size of the original text to a third part of its size.

Although chronological order in event sequences is always useful, alternative organizations often suit better. Considering the Spanish Golden Age publications, the obvious order is chronologically by date of publication, but it could be also useful to order them by author and then by time or by literary genre and author or a combination of these or other characteristics. This combination of dimensions allows an order hierarchy such as first novels, next plays and last poems; then, within each genre, alphabetical order by authors and, finally, chronological order in a third level. Also, it could be possible to rearrange the hierarchy for other exploitation interests in any combination of dimensions: author in the first level and then order by genre; date in first place, next author and last genre, etc. A system able to handle this varying configurations

of the same data would be really useful.

On the other hand, some applications that work on event sequences may need a quantitative analysis, that is, they need the count of how many events with certain characteristic exist during an interval (temporal or from any other dimension). For example, an application that needs to solve queries like "How many plays Lope de Vega wrote?" or "How much the amount of works published by Cervantes is?". Obviously, in these short sequences it could be affordable to sequentially count the works that match the criteria, but for most other cases it might be necessary to precalculate the data and store it separately as it is done by data warehouses. Therefore, there is a need of a solution able to compete against classic data warehouse techniques considering space compression at the same time. Besides, those classic solutions do not save the original order of the information limiting the possible range of queries. There is a lack of aggregated solutions in the state of the art capable of handling queries of the form "How many times a poem from Quevedo was published right after a poem of Gongora?".

Ultimately, a considerable amount of events may have among their characteristics some with spatial nature (e.g. the sequence of street segments[2] traveled every day by a fleet of taxis). Namely, adding space-time dimensions to the event sequences let us use aggregation procedures to extract accumulative space-time knowledge easily. In such context, an event could be the movement of a particular taxi through a segment and the associated characteristics could be date and time, driver license and the identifier of the particular street segment. It is important to note that in this case the considerations on the compression needs of the sequence apply as well as its reorganization and accumulation for solving complex queries. For example, ordering the event sequence chronologically could be useless, but reordering the sequence by driver first with a chronological order on a second level would allow us to recover the trajectory of each taxi trivially. It could be also interesting to have a first level order by days and within each day an order by driver to rebuild the routes of each day. Again, it could be really convenient to precalculate the aggregation of events with common characteristics (e.g. segment identifier) in order to quickly solve queries as "How many times

[2]Considering a street segment as the section between two road crossings with no entrances or exits.

the street segment X has been traversed in the last month?" or "How many times has driver #47 passed through segment Y in the last week?". Given that the set of street segments is constant, its sequence grows with a high repetition rate. Thus, it would be extremely advantageous to have a method to compress the sequence of street segments traversed by a driver each day.

1.2 Contributions

This thesis defines strategies and efficient technologies to represent general event sequences in order to take care of some exploitation needs that are not satisfied in the state of the art. These three needs are:

- **Dynamic Compression.** We designed and implemented a dynamic variable-to-variable compressor (*D-V2V*) that can be used in real-time transmission scenarios where it is not possible to preanalyze the sequence as in other static approaches.

  Sequence compression is an active research field where many strategies have been devised (typically classified into statistical and dictionary based). Nevertheless, most of them are static or semi-static compressors (they need at least two passes over the sequence to calculate symbol frequencies and build a model). There are a few dynamic statistical compressors ([Fal73, Gal78, Far05]) but those compressors are focused on symbol frequency and do not take advantage of recurring sequences of symbols as grammar-based compressors do ([ZL77, ZL78, LM99]). Namely, statistical compressors assign variable-length codes to input symbols (e.g. each word) exploiting frequency distribution while dictionary-based compressors assign fixed length codes to variable length symbol sequences, hence, taking advantage of repeated sequences. Therefore, these compressors are called *fixed-to-variable compressors* and *variable-to-fixed compressors* respectively.

  This thesis introduces a variable-to-variable compressor that takes advantage not only of variable-length event sequence repetitions but also of their frequency distribution. There is already a semi-static variable-to-variable compressors [BFL$^+$10]. However, this work

introduces the first general-purpose dynamic variable-to-variable compressor able to exploit the existence of subsequences of co-occurring elements while still assigning variable-length codewords that will be shorter (following a statistical approach) for the most frequent symbols. In order to compare the efficiency of this proposal with other state of the art solutions, we have used word sequences since other competitors were not able to handle generic event sequences.

- **Aggregation of preprocessed values.** In this research line we proposed and tested a general-purpose compact structure (*Total Matrices*) which is able to solve multidimensional aggregated queries in constant time.

  Our work is based on a technique presented 35 years ago [Cro84] to improve texture-map computations with the help of an aggregated matrix representation. In our research, we have generalized this technique to accumulate the number of event occurrences with characteristics matching a particular dimensional criteria. Namely, our proposal can efficiently handle multidimensional range queries without degrading performance. Besides, as part of the space reduction concern of this thesis, several compressed options for *Total Matrices* are introduced.

  To demonstrate the general character of our approach, this method was used to solve aggregated queries in two different relevant application domains: public transportation and mobile workforce management.

  In the former scenario, events of the non-aggregated sequence represent the passenger boardings (or alightings) on a particular bus in a particular stop and time. *Total Matrices* allow to answer queries about how many passengers boarded (or alighted) on a range of consecutive stops of a particular line during the journeys of a given time interval (e.g. rush hour) or on a range of consecutive stops (e.g. suburban stops). Thus, public transportation decision-makers can easily exploit demand information for each line or stop during any time interval.

  On the other hand, we introduced drivers' matrices on logistics in

order to exploit truck drivers activities (e.g. driving in slow traffic, driving off planned route, visiting a customer's facility, etc.). It is important to note that these activities are semantic annotations for the trajectory performed by each truck. In this context, the events of the sequences are the activities, i.e. the semantic label for each segment of the truck trajectory. Besides the activity label, these events have other dimensions of interest such as worker identifier, start time, duration, etc.

- **Multiple indexing of event sequences.** We have created an indexing method (*Stacked Wavelet Trees*) based on concatenating several indexing trees such as the output of one tree is the input of another, each one reorganizing the data based on a different dimension.

  As it has been previously stated, the order of event sequences may change depending on the exploitation needs that we are aiming to solve, even being necessary several different orders depending on the domain. For instance, if the goal is to retrieve the trajectory of a taxi driver during a particular day, the best approach would be to order the sequence of street segments traversed by taxi drivers during a year first by day and, within each day, order the elements by driver. Yet, if we aim to know which drivers traversed a particular street during the evening, the best order would be to sort those segments first by street segment and a second level order by time. Note that it is only possible to create an index for the event sequence once the sequence is ordered. Our approach, based on *wavelet trees*, enables to index event sequences in a flexible way allowing their exploitation by the order that best suits each query.

In addition to study compression, aggregation and indexation of event sequences, this thesis has faced real problems that can be solved with these techniques. Thus, an additional contribution of the thesis is the modeling of real problems as sequences of events and their solution using the techniques proposed in the thesis.

1.3 Structure of the Thesis

The structure of the thesis is as follows. First, in Chapter 2 some state of the art technologies used for this thesis are presented (e.g. *bitvectors*, *wavelet trees*, etc). Highlighting the practical approach of this work, Chapter 3 describes the application domains where our proposals were employed and the obstacles they can overcome. Chapter 4 introduces our dynamic variable-to-variable compressor and its experimental evaluation on texts written in natural language. Our technique based on aggregation and its applications are explained in Chapter 5 along with the experimental results obtained in the fields of public transportation and mobile workforce management. Chapter 6 covers the novel indexing approach based on *wavelet trees* and its evaluation, also in the mobile workforce management domain. Finally, this manuscript concludes with two appendices: Appendix A lists the relevant works published during the development of this thesis and Appendix B includes a brief summary of this text in Spanish as it is required by the current regulations of the PhD program this work is submitted to.

Chapter 2

Basic concepts and technologies

This chapter briefly describes some state of the art knowledge and some structures that were used as building blocks of our work and will be mentioned in the following sections. These previous contributions may be classified into four main categories:

- Basic structures. In Section 2.1, we focus in two basic components such as *bitvectors*, that are present in most compact data structures, and a compressed representation of trees named *LOUDS*, that are of interest in further sections to represent hierarchies of elements.
- Text compression. The most relevant ancestors of our dynamic variable-to-variable compressor are detailed in Section 2.2.
- Index structures. Section 2.3 takes care of two self-indexing structures that will be of great interest in Chapter 5 and Chapter 6.
- Data aggregation. Finally, Section 2.4, provides a general overview of classic data warehouses as well as some interesting aggregated structures.

All these popular proposals will converge in future chapters of this thesis either as part of a new contribution or as a baseline to test our results against a robust well-known adversary.

2.1 Basic structures

This section contains a brief explanation of two basic tools such as *bitvectors*, which not only permit to represent a sequence of binary events but also to efficiently perform some basic operations that make them a fundamental part of many compact data structures, and *LOUDS*, a bitvector-based compact representation of ordered trees that not only yield optimal space, but also supports navigation.

2.1.1 Bitvectors

Bitvectors are the basic components of many compact data structures and, also, a keystone along this thesis. A bitvector $B[1, n]$ is a sequence of zeroes and ones of length n. The following operations are expected to be supported:

- $rank_1(B, i)$ returns the number of set bits in $B[1..i]$. Alternatively, $rank_0(B, i) = i - rank_1(B, i)$ and also $B[i] = rank_1(B, i) - rank_1(B, i - 1)$.
- $select_1(B, i)$ returns the position in $1..n$ where the ith 1 occurs. Therefore, $rank_1(B, select_1(B, i)) = i$.
- $access(B, i)$ returns the original value $B[i]$.

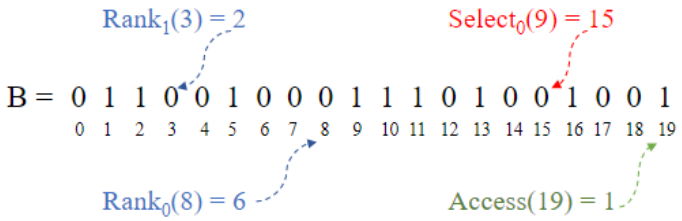


Figure 2.1: A bitvector and its three operations: *Rank*, *Select* and *Access*.

Rank and *Select* can be supported in constant time by using $o(n)$ extra bits [Jac89, Mun96]. There also exist compressed bitvector representations of B [RRR02, RRS07, OS07, GGG+07, Nav16, Góm20] with support to the three operations defined.

Among them, the compressed representation due to Raman, Raman, and Rao (RRR) [RRS07] exploits zero-order compression and permits to store a bitvector B in total space $nH_0(B)+o(n)$ bits, while providing constant-time *access*, *rank* and *select* operations. Other compressed representations exploit bitvector sparseness to further reduce space needs [OS07] or even are tailored to deal with bitvectors containing long runs of zeros and ones [Nav16, Góm20].

2.1.2 LOUDS

Level Order Unary Degree Sequence (*LOUDS*) [Jac89] introduced a new effective approach to encode ordered trees achieving the asymptotic optimum of two bits per node. The tree structure is represented by drawing the degree of each node in (left-to-right) level-order. Unary codes [Sal07] are used to encode the degree sequence, i.e. a degree is represented by the string 1^r0, being r a repetition of *degree - 1* ones. An example can be seen in Figure 2.2 where it is depicted a hierarchy tree and its *LOUDS* representation in bitvector D. The root level of the tree contains only one (10 in unary) node, the root node, and has not to be explicitly stored. In the first level we have 3 entries (1110 in unary), for each of them, we have two children in the next level (which translates in the unary code to $110-110-110$) and so on.

As each node is represented as a 1 at its parent encoding it will be necessary $n-1$ ones (excluding the root node); besides, as all the unary sequences that encode the degree of each node end with a 0 (there are n $0s$) the total length of the encoded sequence of degrees would be $2n-1$ bits for a tree with n nodes. This encoding procedure based on a bitvector that contains unary codes brings the capability of using supports *rank* and *select* operations to enable the basic navigation through the tree. As an example, it would be simple to calculate in the bitvector of Figure 2.2 the position of the first child of "Desserts" (4-th node counting the root) as $select_0(D, rank_1(D, 4))+1 = select_0(D, 4)+1 = 11+1 = 12$.

In Section 2.4.4 we will show how *LOUDS* is used as a main component of the *CMHD*, a state of the art solution for data aggregation.

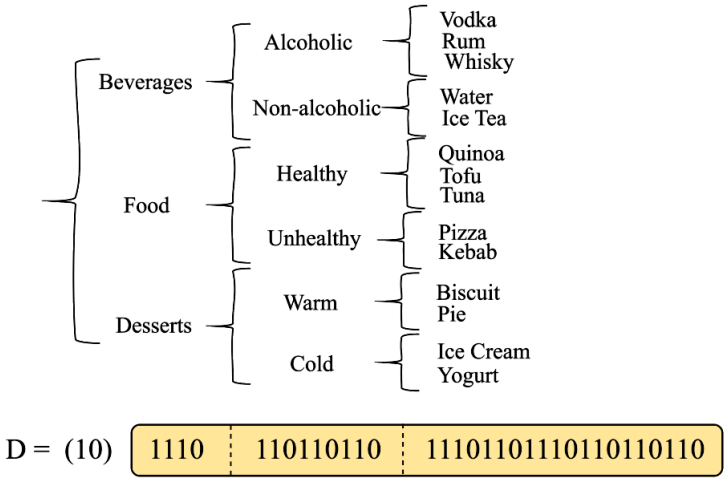


Figure 2.2: A hierarchy tree coded as a LOUDS bitvector.

2.2 Text compression

Text compression is an important subarea of data compression that has traditionally faced two main interest. On the one hand, it pursued the target of reducing the size of the source text as much as possible, even at the expense of sacrificing compression and decompression efficiency ([Shk02, Wel84]). On the other hand, the raise of text retrieval systems ([WMB94, BYRN08]) showed that it could become interesting to trade some loss of compression effectiveness by the capability of being able to handle a text database in compressed form. Therefore, the new requirements were to be fast at both compression and decompression and also to allow direct searches to perform efficiently within the compressed text (without the need for a prior decompression). This lead to the development of new semi-static word-based text compressors such as *Word-based Huffman* [Mof89], *Tagged Huffman* [dMNZB00] and, finally, *End-Tagged Dense Code* (*ETDC*) [Far05, BFNP07] which become the most suitable compressor for text databases.

All of them were word-based semi-static compressors that parsed the source text to gather the different input words and their frequency, that allowed to encode them using a variable-length coding that assigned

shorter codewords to the most frequent words. Since the codeword given to a word did not vary along the database, direct searches were possible by compressing the searched pattern and then searching for such compressed pattern within the compressed text.

To improve the compression ratios of *ETDC*, while still retaining some searching capabilities and decompression performance, a *variable-to-variable compressor* (*V2Vc*) [BFL+10] was created based on the idea of compressing not only words (as in *ETDC*) but also phrases of words.

In a different scenario, targeting at real-time compression, new dynamic compressors based on *ETDC* were also developed [Far05, BFNP10]. Even though we will discuss text compression scenario later on Section 3.1, we will present here *ETDC*, *DETDC* and *V2Vc* as the ancestors of the dynamic compressor that makes up one of the main contributions of this thesis and will be presented in Chapter 4.

2.2.1 End-Tagged Dense Code

As stated above, *End-Tagged Dense Code* (*ETDC*) [Far05, BFNP07] is a semi-static word-based compression technique. Basically, it performs a first pass over the input text to gather the different words (vocabulary of words) and their number of occurrences. Then, it sorts the words of the vocabulary decreasingly by frequency and, following a dense-coding, it assigns shorter codewords to the more frequent words. Finally, in a second pass over the source text it replaces the input words by the corresponding codeword, hence obtaining compression.

The main contribution of *ETDC* was the definition of a *dense coding* to create byte-oriented codewords (i.e. codewords are sequences of bytes rather than sequences of bits), despite previous alternatives such as *Plain Huffman* or *Tagged Huffman*, that used Huffman algorithm [Huf52] to obtain prefix-free[1] encodings. The advantage of the *dense* coding scheme of *ETDC* came from the idea of marking the first bit of the last byte of a codeword with a 1, whereas the first bits of the remaining bytes were set to 0. This made *ETDC* a prefix coding without the need for using Huffman coding (as in *TH*) and provided synchronization

[1] A prefix-free encoding ensures that no codeword is a prefix of a longer codeword, which is an useful property to support efficient decoding, as no look-aheads are required because each symbol is uniquely decodable.

capabilities that enabled both direct access to the compressed text (and consequently random decompression from any offset) and the possibility of performing fast compressed text searches by using Boyer-Moore-type searches [Hor80]. In addition, the coding schema of *ETDC* no longer depends on the actual number-of-occurrences of a word (as in Huffman coding) but only on the rank of each word in the decreasingly-sorted vocabulary, what makes it simple and fast. Below, we show how byte-wise codewords are assigned to each word, assuming such a decreasingly-sorted vocabulary of words.

- The first 128 words from the vocabulary (i.e. words ranked from position 0 to 127) are sequentially assigned 1-byte codewords from 10000000 (byte value 128) to 11111111 (byte value 255). Note that the first bit of the (unique) last byte is set to 1. This is depicted in Table 2.1.

- Words ranked from $2^7 = 128$ to $2^7 + 2^7 \times 2^7 - 1 = 16511$ are sequentially assigned two-byte codewords from 000000000:10000000 (word ranked 128); 000000000:10000001 (word ranked 129);... to 011111111:11111111 (word ranked 16511). The first byte of each codeword has a value in the range $[0, 127]$ (i.e. with the first bit set to zero) and the second byte contains values within $[128, 255]$ (i.e. with the first bit set to one).

- Word ranked $2^7 + 2^7 \times 2^7 = 16512$ is assigned the first tree-byte codeword 000000000:000000000:10000001, and so on.

The simple encoding schema introduced by *ETDC* and the fact of depending on the rank of the symbols within the sorted vocabulary rather than on their actual frequency values, allowed also the definition of on-the-fly encoding and decoding algorithms such that:

- $c_i \leftarrow$ *encode(i)* receives the word rank i and returns the assigned codeword in time proportional to $|c_i| = O(\log_8 i)$.

- $i \leftarrow$*decode(c_i)* receives a codeword c_i and, again in time proportional to $|c_i|$, returns the rank i of the corresponding symbol within the sorted vocabulary.

Word rank	Codeword assigned	# Bytes	# Words
0	10000000		
1	10000001		
2	10000010	1	2^7
...	...		
$2^7 - 1 = 127$	1111111		
$2^7 = 128$	00000000:10000000		
129	00000000:10000001		
130	00000000:10000010	2	$2^7 * 2^7$
...	...		
$2^7 * 2^7 + 2^7 - 1 = 16511$	01111111:11111111		
$2^7 * 2^7 + 2^7$=16512	00000000:00000000:10000000		
16513	00000000:00000000:10000001		
16514	00000000:00000000:10000010	3	$(2^7)^3$
...	...		
$(2^7)^3 + (2^7)^2 + 2^7 - 1$	01111111:01111111:11111111		
...	...	...	...

Table 2.1: *ETDC* distributes words in blocks according to their frequency. Shorter codes are assigned to symbols on higher blocks.

As stated above, *ETDC* become the most suitable compressor for text databases [BYRN08] due to its rather good compression effectiveness (compression ratio[2] around 32%), and mainly due to its fast decompression and the ability to efficiently perform direct searches within the compressed text. In addition, the simple encoding and decoding algorithms opened the door to creating new dynamic compressors (tailored for real-time transmission) as we will show in the next section.

2.2.2 Dynamic End-Tagged Dense Codes

When pursuing the target of creating efficient text compressors that could match real-time constraints, i.e. a word can be compressed and

[2]We show compression ratio as $100 \times \frac{\texttt{compressed text size}}{\texttt{size of original text}}$; i.e. the percentage of the compressed size with respect to the plain text size.

sent to a receiver as soon as it is read, dynamic *one-pass* compression is required. Despite semi-static compression, a one-pass statistical compressor must dynamically compute the model of the text (gathering words and their number of occurrences) as words are being read, instead of waiting for the whole text, or a large block of it, to become available. Therefore, using such varying model, each input word can be assigned a rather-optimal codeword. The decoder/receiver performs symmetrically to the encoder/sender by decoding the received codewords and recreating the same model held on the compressor from the decoded codewords.

Dynamic end-tagged dense code (*DETDC*) [Far05, BFNP08, BFNP10] takes advantage of the simple on-the-fly *encode* and *decode* algorithms from *ETDC* and permits both sender|compressor and receiver|decompressor to remain synchronized by simply keeping the same vocabulary of words sorted by frequency. *DETDC* algorithm is not complex, basically, assuming that, at a given moment, the vocabulary of the sender contains n words, when the sender inputs the next word w it could find it in its vocabulary at position i, so it simply sends $c_i \leftarrow encode(i)$ to the receiver. Otherwise, if w is a new word, it sends $c_n \leftarrow encode(n)$ (used as an escape codeword) followed by w in plain form. In any case, the encoder increases the frequency counter f of w to $f + 1$ and runs a simple *update* algorithm that swaps w with the first word that has frequency equal to f. This *update* algorithm keeps the vocabulary of words sorted by frequency and runs in $O(1)$ time.

The receiver is also very simple. It receives a codeword c_i and decodes it as $i \leftarrow decode(c_i)$. Then, if $i < n$ it has decoded the word w_i at the i-th entry of the vocabulary. Otherwise, if $i = n$ (escape codeword) it receives a new word in plain form and adds it at the end of the vocabulary. Finally, a similar *update* procedure to that of the sender is run to increase the frequency of that word and to keep the vocabulary sorted. Such *update* algorithm runs also in $O(1)$ time.

DETDC became the first word-based dynamic compressor matching real-time constraints. As the dynamic counterpart of *ETDC*, it obtained rather identically compression ratios and faster (one-pass) compression. Yet, due to the need of running the update algorithm each time a word is decoded, decompression speed and searching capabilities worsened.

2.2.3 Semi-static variable-to-variable compression

In 2010 the best option for building searchable compressed texts were *Tagged Huffman* [dMNZB00] and *ETDC*. Although, those compressors could not achieve compression ratios below 30% in natural language English text; i.e. searchable compressed text could not compete against strong compressors such as dictionary-based compressors (e.g. Lempel-Ziv family [ZL77, ZL78, Wel84]) or k-order statistical compressors (e.g. PPM [CW84, Shk02]). Thereby, it was necessary a compressor able to mix the best of both worlds, having strong compression ratios and fast searching capabilities within the compressed text. This is how *Variable-to-variable compressor* (*V2Vc*) [BFL$^+$10] was born.

This novel solution uses the same encoding schema as *ETDC* but it considers not only words as the input symbols to encode but also sequences of words (phrases). Therefore, it combines both, the idea of processing variable-length input symbols (as usual in dictionary-based compressors) and the assignment of variable-length codewords to the symbols (as typical from statistical compressors). These made *V2Vc* the first variable-to-variable compressor of the state of the art. The key concept in this approach is to select "good" phrases. To achieve that, *V2Vc* follows these steps:

- **Parsing and selection of candidate phrases**. *V2Vc* uses auxiliary structures to handle all the possible word sequences in a text, their frequency and the number of words that compose each of them (phrase length). In particular, it uses a word-based *suffix array* [MM93] built over the source text, that basically consists of an array of pointers to each word beginning that keeps all the possible suffixes from any word position to the end of the source text sorted lexicogrpahically. It also uses a *longest common preffix* (*LCP*) [MM93] structure that permits *V2Vc* to gather the actual number of occurrences of any phrase pointed from the *suffix array*. In order to select the best phrases, two heuristic techniques have been used:
  - **H1:** Select all the longest phrases that has at least a given frequency threshold.
  - **H2:** Compare the profit obtained between two similar phrases.

Therefore, in this first phase the compressor becomes acquainted of which words appear in the original text (as in *ETDC*) and, immediately afterwards, it adds the chosen candidate phrases to the word vocabulary (*phrase-book*). Thus, *V2Vc* treats equally words and sequences of words as the source symbols that will later be encoded.

- **Gathering the final phrase-book and producing a phrase-tokenized representation of the text**. Since each word can be either encoded individually or inside a phrase, it is important to discern which alternative will be used for each word. Using the auxiliary structures mentioned before it is possible to build an intermediate representation of the original text where each word (or phrase) is replaced by an identifier associated to its position in the *phrase-book*.

- **Coding and codeword replacement**. In this phase, each identifier of the intermediate representation is replaced by a variable-length *ETDC* codeword using $C_i d \leftarrow ETDC.encode(id)$ and considering the *phrase-book* ordered by frequency. This makes up the final compressed sequence.

Once the compressed text is obtained, it is mandatory to follow a common communication protocol between compressor and decompressor so that the decompressor can know the contents of the *phrase-book* used by the compressor. Therefore, we have to include a header (as in *ETDC*) with all the words in the *phrase-book* in plain form, and also the phrases. To represent phrases in a compact way the first time a phrase is encoded in the compressed file, it is represented (compressed) as the sequence of all the codewords associated to its individual words. Namely, if the first occurrence of a particular phrase starts at position i in the compressed text, we only have to store in the corresponding entry x of the *phrase-book*. A pair (i, k) indicating that the definition of the words of that phrase appear compressed from position i on within the compressed file and the number of words k it contains. Subsequent occurrences of the phrase x will be encoded as $C_x \leftarrow ETDC.encode(x)$

On the other hand, the decompression process rather is identical to decompression in *ETDC*. Although, each time the first position of

an encoded phrase x is reached, the decompressor recovers the plain representation (string) of the phrase. Afterwards, that phrase is inserted in the *phrase-book* at the x-th entry so next time the codeword C_x is decoded for the compressed text, it is handled as the codeword of any single word, i.e. output string at position *phrase-book*$[x]$.

Following this strategy, *V2Vc* reaches competitive compression ratios (around 22%) even when compared with strong compressors such as *p7zip*. *V2Vc* has a slow compression process due to the complex candidate phrase detection procedure but it owns a fast decompression (analogous to *ETDC* thanks to having decompress a smaller compressed file). In addition, it is able to search over compressed text even faster than *ETDC*.

2.3 Index structures

The possibility of retrieving the desired information faster is one of the main concerns in computer science. Luckily, at this point the state of the art have provided some clever solutions that may apply to a very wide range of contexts. This section describes two of them as basic indexing engines in the contributions of this research and in many other compact structures.

2.3.1 FM-index

The *Burrows-Wheeler Transform* (*BTW*) [BW94] is a data transformation algorithm that creates a matrix whose rows are cyclical shifts of the same text/sequence that are kept sorted in alphabetical order. The key idea is to build blocks enhancing the locality of repetitions ergo improving the compression. Since all rows are cyclical, it is possible to traverse/recover each element of the original sequence in reverse order just using the first (F) and the last (L) columns of the matrix. Figure 2.3 displays an example of the matrix containing all the cyclical shifts of the sequence $S = \langle abracadabra\$\rangle$. Note that $BWT(S)$ is defined as the last column L of that matrix. Yet, since F is a permutation of L that contains the same symbols but sorted lexicographically considering the cyclical string starting on them, we can map any symbol $c = L[i]$

into its corresponding position j in F by just counting the number of times (k) character c occurs in $L[1, i]$ (i.e. we know that $L[i]$ contains the k-th occurrence of c). Therefore, j can be easily obtained as the initial position of the range of symbol c within F added to $k - 1$. This is called the LF mapping of the BWT.

Therefore, we know that the cyclical shift of S starting at $L[i]$ is identical to the one starting at $F[j]$. Since the row j is also cyclical, if we access $L[j]$ we will obtain the entry preceding $L[i] = F[j]$ in S. By repeating this operation, we can recover all the entries of S, in reverse order. For example, if we start at row $i = 4$ (which contains the original string S) we see that $L[4]$ contains the first occurrence of '$', which is found at position $j = 1$ in F. Consequently, $L[1]$ contains 'a' and we have recovered $S[11, 12]$ ="a$". Since $L[1]$ holds the first occurrence of 'a' in L, which is found at $F[2]$, we access $L[2]$ and recover $S[10] =$ 'r'. Now, it is the first occurrence of 'r' in L, which is found at $F[11]$. Consequently, $L[11]$ recovers 'b' and we have already recovered the substring $S[9\,12]$ ="bra$". We can continue the process until recovering the whole source sequence S. The LF mapping and the fact that all the cyclical shifts of S appear sorted in F has led to the raise of a large family of self-indexing structures based on the BWT, whose best-known component is the *FM-index* [FM00].

Given a sequence $S[1, n]$ built on an alphabet $\Sigma = [1, \sigma]$, the *FM-index* [FM00] provides a self-indexed representation of S based on the BWT of S and the use of backward search for identifying pattern occurrences. Figure 2.3 shows how this index is able to find the occurrences of the pattern $P[1, p]$ ="bra" within S through backward search. It starts looking for the range in L containing the last symbol of the pattern ('a') and traverses the matrix by navigating from F to L until either it retrieves a range with the occurrences of the whole pattern or it runs out of results. In practice, this operation is performed using L and two auxiliary matrices (C and Occ) in order to achieve $O(p)$ time. Array C (see Table 2.2) stores, for each symbol $c \in \Sigma$, the amount of occurrences in S of all the symbols that are lexicographically smaller than c (i.e. the starting offset of the range associated to symbol c in F). Matrix $Occ(c, q)$ (see Table 2.3) keeps the amount of occurrences of character c in the prefix $L[1..k]$.

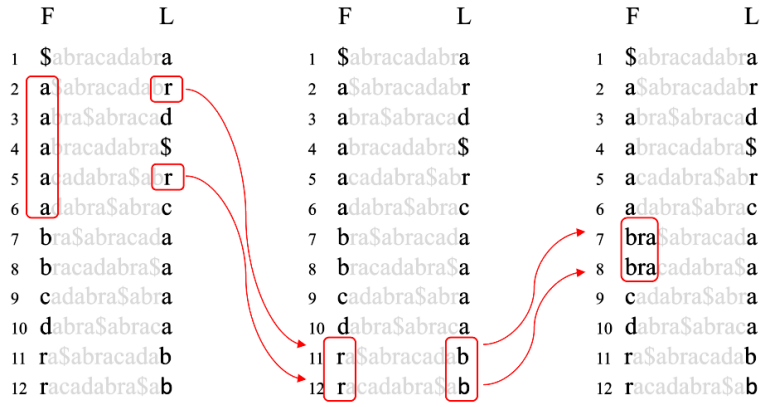


Figure 2.3: *BWT* and *FM-Index* example: Given the text "abracadabra$" we show the cyclical shifts of the *BTW* matrix and highlight F and L. Also, we show how to find all the occurrences of the pattern "bra" using backward search.

Continuing with the example of Figure 2.3, the first step to find the pattern "bra" is to retrieve the range in F of the last character of the pattern 'a' as $[C[\text{'a'}] + 1,\ C[\text{'a'}{+}1]] = [\underline{2}, \underline{6}]$. To calculate the range of all the suffixes containing "ra" we need to compute $[C[\text{'r'}] + Occ[\text{'r'}, \underline{2} - 1] + 1,\ C[\text{'r'}] + Occ[\text{'r'}, \underline{6}]] = [10 + 0 + 1,\ 10 + 2] = [\underline{11}, \underline{12}]$. Finally, the last step to find all the occurrences of pattern "bra" is to calculate $[C[\text{'b'}]{+}Occ[\text{'b'}, \underline{11}{-}1]{+}1,\ C[\text{'b'}]{+}Occ[\text{'b'}, \underline{12}]] = [6{+}0{+}1,\ 6{+}2] = [7, 8]$. Thus, there are 2 occurrences of the sought pattern in this text.

This structure requires $5nH_k(S) + o(n)$ bits of space and permits to search for the occurrences of a pattern $P[i, m]$ in time $O(m{+}occ\ log^{1+\epsilon}n)$ (occ being the number of occurrences of P within S). Several variants of this scheme exist [FM01, FM05, FMMN07, MN05] which induce different time/space tradeoffs for the counting, locating, and extracting operations that respectively counts the number of occurrences of P in

c	$	a	b	c	d	r
C[c]	0	1	6	8	9	10

Table 2.2: Auxiliary array C for column L in the matrix of Figure 2.3.

	a	r	d	$	r	c	a	a	a	a	b	b
	1	2	3	4	5	6	7	8	9	10	11	12
$	0	0	0	1	1	1	1	1	1	1	1	1
a	1	1	1	1	1	1	2	3	4	5	5	5
b	0	0	0	0	0	0	0	0	0	0	1	1
c	0	0	0	0	0	1	1	1	1	1	1	1
d	0	0	1	1	1	1	1	1	1	1	1	1
r	0	1	1	1	2	2	2	2	2	2	2	2

Table 2.3: Auxiliary structure to compute the number of occurrences of character c in the prefix L[1..k] (*Occ(c,k)*) in constant time.

S, locates the positions where P occur in S and extracts/recovers any substring $S[i, j]$ from S.

Section 5.3 will use a *FM-Index* to search patterns over an auxiliary structure in order to improve the performance and range of action in one of our proposals.

2.3.2 Wavelet Tree

The *wavelet tree* (*wt*) is a data structure that, as the *FM-index* discussed above, permits to represent any general sequence in a self-indexed way. Given a sequence $S[1, n]$ over an alphabet $\Sigma[1, \sigma]$, a *wavelet tree* is built as a balanced binary tree that subdivides the symbols in the sequence represented within each node according to their position in the alphabet. In particular, depending on whether those symbols either fall within the first half of the alphabet (Σ_L) or within the second half (Σ_R).

Considering the whole source sequence S, the root node of the tree contains a bitvector $B[1, n]$, such that $B[i] = 0 \iff S[i] \in \Sigma_L$, and $B[i] = 1$ otherwise. Then, the sequence handled by the root node S is divided into two subsequences S_L and S_R that are represented within its two children nodes. Basically, S_L contains all the symbols $s \in \Sigma_L$ from S (in the same order they occurred in S) and S_R all those symbols $s \in \Sigma_R$. The process is repeated recursively, considering the subsequence S_L in the left sub-tree, and the subsequence S_R in the right sub-tree. The resulting tree has $\lceil \log \sigma \rceil$ levels, and a total of n bits per

level, for a total size of $n\lceil \log \sigma \rceil$ bits. Pointers between nodes can be greatly reduced using an implementation in which all the bitvectors of the same level are concatenated [CN09], hence keeping total space as $n\lceil \log \sigma \rceil + O(n \log \sigma)$.

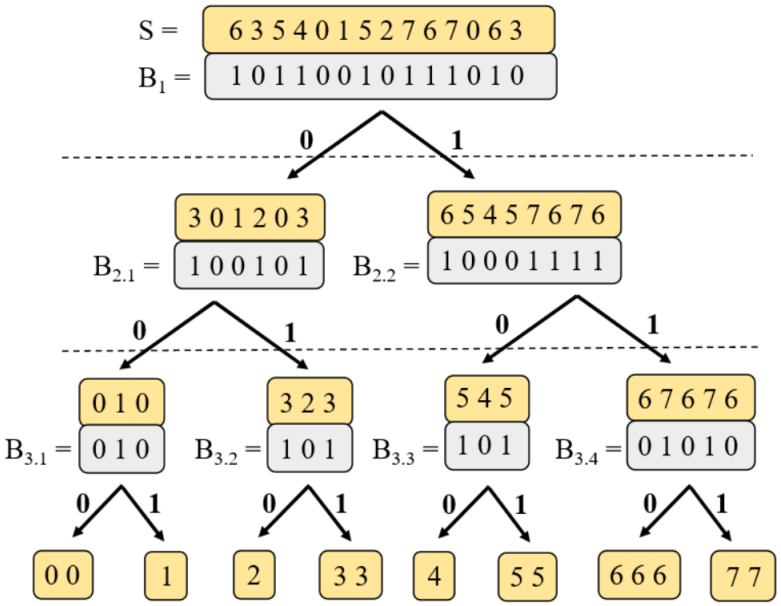


Figure 2.4: Wavelet tree for sequence $S = \langle 6\,3\,5\,4\,0\,1\,5\,2\,7\,6\,7\,0\,6\,3 \rangle$.

Instead of considering the partitioning of Σ at each node, we could also consider that each symbol $s_i \in \Sigma$ is assigned a codeword c_i, so that at the root node, we set $B[i] = 0 \iff$ the first bit of the codeword of $S[i]$ is a 0, and $B[i] = 1$ otherwise. Then, we repeat the process recursively as above, yet considering the k-th bit of the codeword of the symbols represented within the nodes of the k-th level.

Assuming a binary encoding for the symbols, Figure 2.4 depicts the wavelet tree associated to the sequence $S = \langle 6\,3\,5\,4\,0\,1\,5\,2\,7\,6\,7\,0\,6\,3 \rangle$ where only grey shaded areas (the bitvectors B_1, $B_{2.x}$ and $B_{3.x}$) are stored. Note that each node at the k-th level represents all those symbols from the original sequence that share the same $k-1$ initial bits in their encodings. For example, the node containing $B_{3.3}$ contains only symbols

4 and 5, whose binary encodings start by bits 1 0.

With this simple structure, *wavelet trees* are able to represent the original sequence S, while providing also support for *access*, *rank*, and *select* operations over the symbols of S. The two former roam the tree from the top to the leaves whereas the latter does it in the opposite direction as we will discuss below. Those operations are defined as:

- $rank_c(i)$ returns the number occurrences of c in $S[1..i]$.
- $select_c(i)$ returns the position of the i-th occurrence of symbol c within S.
- $access(i)$ returns the original value $S[i]$.

To solve *access(i)* operation, we start at position i in the root level and we only need to descend the tree considering the value of the bitvector $B[i]$ at each node. We check whether we have to move to either the left or right child, and respectively, we track the corresponding position at the next level as $i \leftarrow rank_0(B, i)$ or $i \leftarrow rank_1(B, i)$. Thus, the solution for $access(3)$ (the first 5 in the S) begins with a descent through the right branch as $B_1[3] = 1$. Since $rank_1(B_1, 3) = 2$, we can locate our target in the second level, in the 2-nd position of $B_{2.2}$. Since $B_{2.2}[2] = 0$, we need to descend through the left branch this time. At the third level, the sought solution is at position $B_{3.3}[rank_0(B_{2.2}, 2)] = B_{3.3}[1] = 1$. Since we have traversed the branches $0 \cdot 1 \cdot 0$, that correspond to the (binary) encoding of symbol $S[3]$, we conclude that $access(3) = 101 = 5$.

Operation $rank_c(i)$ would be solved in a similar way, performing either a $rank_0$ or a $rank_1$ operation depending on the bit of the encoding of c in the previous level. Yet, in this case, we only need to report the final position within the corresponding leaf.

To solve $select_c(i)$, a bottom-up traversal of the wavelet tree is required. First, we locate the leaf corresponding to symbol c according to its encoding. From there on, we track the i-th position from that leaf (such leaf would be devoted only to c symbols) up to the corresponding position at the root node. This is done using *select* operations within the bitvectors of the nodes traversed up to reaching the root. For example, if we want to retrieve the position of the first occurrence of symbol $c = 5$ of S, we should begin in the first position of the leaf containing symbols

$c = 5$. Since the last bit of the encoding of 5 is a *one*, we would start the bottom-up traversal at $i \leftarrow select_1(B_{3.3}, 1)$. Now, as the bitvector $B_{3.3}$ is within the left child of $B_{2.2}$, we should look for the first 0 of $B_{2.2}$, i.e. $i \leftarrow select_0(B_{2.2}, 1) = 2$. As $B_{2.2}$ is within the right child of B_1, the position at the top bitvector we are looking for is obtained as $i \leftarrow select_1(B_1, 2) = 3$.

In addition to plain *wt* representations, that essentially require the same space as the original uncompressed sequence S, several compressed representations exists, yielding space proportional to the zero-order entropy of S. Particularly, one approach to achieve this is to use compressed bitvectors within the *wt*. By using the variant due to Raman, Raman, and Rao (RRR) [RRS07] discussed in Section 2.1.1, we can store a *wt* in total space $nH_0(S)+o(n \log \sigma)+O(\sigma \log n)$ bits, while operations can still be solved in $O(\log \sigma)$ time. An alternative compression strategy for *wavelet trees*, where the binary encoding of the symbols seen above is replaced by a prefix-free variable-length encoding (using Huffman encoding), as in [FGNV08], leads to a Huffman-shaped *wavelet tree* where the overall size is reduced to $n(H_0(S) + 1) + o(n(H_0(S) + 1)) + O(\sigma \log n)$ and the operations are performed on average in $O(H_0(S))$ time, whereas the worst-case complexity is still $O(\log \sigma)$.

Wavelet trees will be used in Chapter 6 as building blocks of a more complex structure created to index information attending different criteria.

2.4 Data aggregation

Despite that commercial data warehouses have gained the control of the market, it is still possible to improve them from a spatial point of view (among others). The following subsections fathom into how classic ideas evolved until achieving this maturity level and how compression techniques may open new paths in this field.

2.4.1 Data Warehouses

It was the best of times. It was the worst of times. Supported by these words of Dickens, William H. Inmon began the defense of his

data warehouse model to settle the discussion once and for all [Inm11]. For decades, the data warehouses world has been divided between two different main approaches and this controversy is still in vogue nowadays.

The seminal idea for this great debate was established in 1958 by an IBM employee called Hans Peter Luhn. Luhn was concerned about the inefficient communication and the human effort on the dissemination of information. Therefore, he proposed a system to accommodate all information problems of an organization [Luh58]. With a little human help, his system was able to find the action-points (individuals, departments, divisions...) and the activities that characterized each of them. Once the hierarchy was established, the system was in charge of the acquisition, storage and distribution of new information through each action point. Taking advantage of the new improvements on the automation of electronic devices, he managed to accomplish his goals in a fast and efficient way for that time. In addition, he coined a term that would later name one of the most prolific areas in information systems research: *Business Intelligence*.

Having in mind that the arrival of Database Management Systems (DBMS) did not occur until the '60s [Bac66] (Figure 2.5), it is understandable the oblivion suffered by the generation of business knowledge during this period. The effort was focused on managing the unstoppable flow of information [WL64] or tackling the new data retrieval problems [SL65, Cle67]. Nonetheless, this does not mean that Luhn's pioneer approach was mistaken. Data generation was growing exponentially year after year, to such an extent that the amount of information generated only in 2007 was larger than all the information created since the invention of writing [BvDD09]. The need for exploiting all this data became clear again considerably later, when the foundation of operational databases was settled.

It is at this point of history when Ralph Kimball and William H. Inmon faded in. They both immersed in the information systems area during the '80s [SIKH82, Inm86, IB86] but it was during the '90s when the race for building a database-oriented to decision-making begun. Both of them aimed to develop a system capable of managing information in such a way that (following Luhn's goals) it could acquire, store, disseminate, calculate and maintain huge amounts of different data

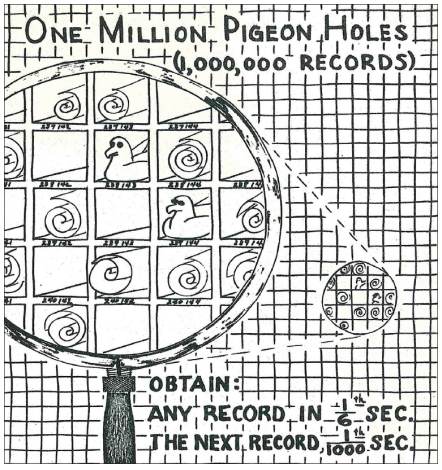


Figure 2.5: Picture from a 1962 internal General Electric document explaining the idea of random access storage using pigeon holes as a metaphor. (Source: https://wp.sigmod.org/?p=688)

while generating understandable and accurate information using efficient methods. Thus, data warehouses were born. Both approaches shared the same central idea: separate analysis workload from transaction workload extracting analytic and historical data derived from multiple sources, cleaning it and loading it into a warehouse. In [KR02], Kimball defines the basic requirements of a data warehouse as:

- The DW/BI system must make information easily accessible.
- The DW/BI system must present information consistently.
- The DW/BI system must adapt to change.
- The DW/BI system must present information in a timely way.
- The DW/BI system must be a secure bastion that protects the information assets.
- The DW/BI system must serve as the authoritative and trustworthy foundation for improved decision making.
- The business community must accept the DW/BI system to deem it successful.

However, as the proverb says, the devil is in the details. Inmon thinks about his data warehouse as a centralized repository for an entire enterprise while Kimball defines his data warehouse as a copy of transaction data specifically structured for query and analysis. Basically, this implies that Inmon defends a top-down approach where the data warehouse has an atomic nature and Kimball supports a bottom-up approach where the union of modules shapes the data warehouse.

Therefore, the former needs a high-skilled team and a considerable amount of time for deploying but reduces redundancy, has a simpler maintenance and an enterprise-wide coverage. On the contrary, the latter uses a modular approach that needs redundancy and complicates maintenance but can be deployed fast by average workers, it is ready to use as soon as the first module is prepared and each area of the enterprise is able to work on its own. There is still no consensus on which solution is better, there are several aspects to consider before choosing one, being even possible a hybrid warehouse using both proposals. Luckily, it is not the aim of this manuscript to choose between them but to introduce new contestants to the fight. Chapter 5 and Chapter 6 will present two new approaches for storing and indexing data destined to aggregation queries opening a new path to compact data warehouses. Our structures follow the principles of Luhn and satisfy Kimball's rules while reducing redundancy as Inmon's method does.

2.4.2 OLAP

Edgar Frank Codd is one of the main characters in database history. Not only he proposed the *Relational model* in 1970 [Cod70] but he also coined the term *On-Line Analytical Processing* (*OLAP*) in 1993 [CS93]. Codd's main idea was always to create an analytic abstract model rather than a particular technology, his beliefs were already hidden in [Cod79] where he states: *There is a strong emphasis on structural aspects, sometimes to the detriment of manipulative aspects. Structure without corresponding operators or inferencing techniques is rather like anatomy without physiology.* Thus, the aim was to introduce a wider vision that complemented the straightforward report functionalities and overcame the limitations of the spreadsheets of the time.

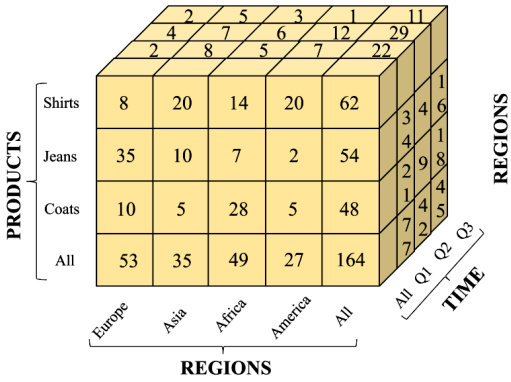
To accomplish that, [CS93] defined an *OLAP* system as a synthesis

of the historical and transactional data that enables dynamic multidimensional analysis fulfilling the following characteristics:

1. Multidimensional Conceptual View
2. Transparency
3. Accessibility
4. Consistent Reporting Performance
5. Client-Server Architecture
6. Generic Dimensionality
7. Dynamic Sparse Matrix Handling
8. Multi-User Support
9. Unrestricted Cross-dimensional Operations
10. Intuitive Data Manipulation
11. Flexible Reporting
12. Unlimited Dimensions and Aggregation Levels

Despite the increasing popularity that columnar databases are experiencing nowadays, data cubes [GBLP96] have been the ruling technology to implement OLAP systems during the last decades. As Figure 2.6 shows, data cubes can be seen as an aggregated representation of the well-known relational tables. These precalculated accumulations enable any system to answer aggregation queries efficiently saving a lot of time. Besides, using its resemblance to a Rubik's cube, it is easy to understand by the final user how it is possible to flip the faces to retrieve the information sought. In fact, OLAP cubes define three basic sets of operations: *roll up* and *drill down* let the user navigate upwards and downwards through the hierarchies, *slice* and *dice* allow to extract subcubes and *pivot* is in charge of rotating the cube.

It is important to notice that data warehouses and OLAP systems crack data according to diverse dimensions pursuing the same goal:



TIME: All	PRODUCTS			
	Shirts	Jeans	Coats	Total
Europe	8	35	10	53
Asia	20	10	5	35
Africa	14	7	28	49
America	20	2	5	27
Total	62	54	48	164

Figure 2.6: Sales information of a clothing enterprise characterized by three dimensions (product, region and time) represented using a data cube (left) and a classic table (right).

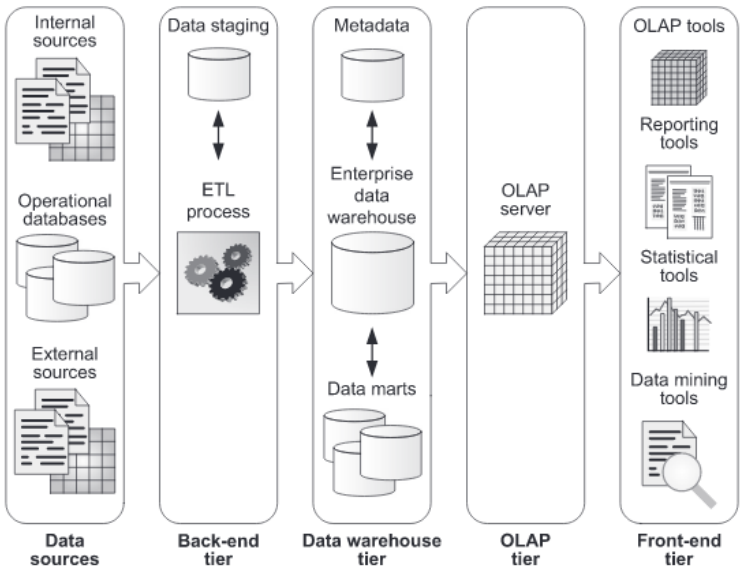


Figure 2.7: Common Data Warehouse structure [VZ13].

deliver data that is understandable to the business users and deliver fast query performance. Nevertheless, they belong to different business

intelligence strati and they can be used separately. Data warehouses are a place to store in a formalized analyzable format while OLAP is a method to analyze the data. Even so, it is usual to mix the capabilities of both creating a system like the one displayed in Figure 2.7. Again, Chapter 5 and Chapter 6 will demonstrate that our proposals lay the groundwork for a new mix using OLAP operations over a compact approach.

2.4.3 Summed Area Tables

In 1984, Franklin C. Crow introduced the *Summed Area Tables* in computer graphics [Cro84] to speed up mipmapping, which requires to be able to compute the average value of a given rectangle within an image.

By using *Summer Area Tables*, given a matrix $A[1,r][1,c]$, for which we want to solve the operation $countRange(A,[x_1,y_1],[x_2,y_2]) \leftarrow \sum_{i=x_1}^{x_2}\sum_{j=y_1}^{y_2} A[i][j]$, the key idea of this approach is to create a new matrix $M[0,r][0,c]$ where all the cells in both row 0 and column 0 are set to zero, and any other cell $M[x][y]$ stores the total sum of all the previous cells within A (to the left and up); i.e. $M[x][y] \leftarrow \sum_{i=1}^{x}\sum_{j=1}^{y} A[i][j]$. An example showing matrices A and M is depicted in Figures 2.8(a) and 2.8(b) respectively. Using M allows us to solve *countRange* operation in $O(1)$ time as:

$$\begin{aligned}\mathsf{countRange}(\mathsf{A},[\mathsf{x}_1,\mathsf{y}_1],[\mathsf{x}_2,\mathsf{y}_2]) \leftarrow M[x_2,y_2]-M[x_2,y_1-1]-\\ M[x_1-1,y_2]+M[x_1-1,y_1-1]\end{aligned} \tag{2.1}$$

Basically, from a geometric point of view, Figure 2.8(b) shows that to compute $countRange(A,[3,2],[7,4])$ we subtract from $M[7,4]=64$ (sum of all the values in $A[1,7][1,4]$) both the values in the area depicted with horizontal bars ($M[7,1]=19=$ sum of values in $A[1,7][1,1]$) and those values in the area depicted with vertical bars ($M[2,4]=18=$ sum of values in $A[1,2][1,4]$). Since we are subtracting the sum of values in the area depicted with both vertical and horizontal lines twice ($M[2,1]=6=$ sum of values in $A[1,1][2,1]$) we still have to add that value ($M[2,1]=6$) once. Consequently, we obtain $countRange(A,[3,2],[7,4]) \leftarrow 64-19-18+6=33$.

		X							
		1	2	3	4	5	6	7	8
Y	1	2	4	3	2	1	3	4	2
	2	1	3	2	1	3	1	3	1
	3	2	1	3	2	2	1	3	1
	4	2	3	3	5	2	1	1	1
	5	2	3	1	1	3	1	1	1

(a) Non-aggregated matrix example.

		X								
		0	1	2	3	4	5	6	7	8
Y	0	0	0	0	0	0	0	0	0	0
	1	0	2	**6**	9	11	12	15	**19**	21
	2	0	3	10	15	18	22	26	33	36
	3	0	5	13	21	26	32	37	47	51
	4	0	7	**18**	29	39	47	53	**64**	69
	5	0	9	23	35	46	57	64	76	82

		X								
		0	1	2	3	4	5	6	7	8
Y	0	0	0	0	0	0	0	0	0	0
	1	0	2	**6**	9	11	12	15	**19**	21
	2	0	3	10	15	18	22	26	33	36
	3	0	5	13	21	26	32	37	47	51
	4	0	7	**18**	29	39	47	53	**64**	69
	5	0	9	23	35	46	57	64	76	82

(b) Aggregated matrix containing the same information as the image below, it can compute in constant time the sum of the blue area.

Figure 2.8: *Summed Area Tables* detailed geometrical explanation: Figure2.8(b) represents the aggregated matrix built over the simple matrix depicted in Figure 2.8(a). The non-aggregated matrix needs to compute each cell within the submatrix individually in order to calculate the total sum of the area shaded in blue. The aggregated representation can solve the operation in constant time using the greater value of the submatrix minus the non selected areas (left and top) plus the small area that was subtracted twice.

Summed Area Tables and *countRange* operation played a key role during the development of this research. Section 5.1 generalize these concepts and introduce some space improvements as long as some general-purpose uses of this seminal work.

2.4.4 CMHD

The *CMHD* [BCPL+16], short for *Compact representation of Multi-dimensional data on Hierarchical Domains*, is a brand new structure

that uses compact data structures to solve aggregated range queries on multidimensional grids.

CMHD could be seen as a compressed approximation of a multidimensional data cube (see subsection 2.4.2). Returning to the idea of a multidimensional matrix where each cell saves a value, *CMHD* recursively divides the matrix into several irregular submatrices built from the hierarchy levels of the dimensions. A two-dimensional (products and regions) sales example can be found in Figure 2.9. For each dimension, a hierarchy of three strati is presented where products are aggregated into several categories and cities are grouped into countries and continents.

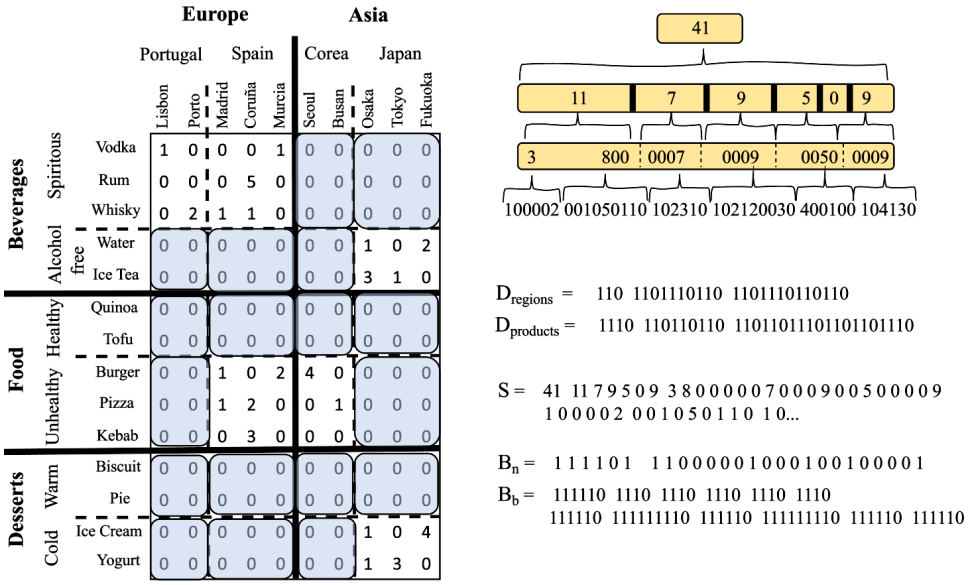


Figure 2.9: *CMHD* example storing the amount of products sold attending the region and type of product. Left figure represents the conceptual model of the structure, the gold tree serves as the conceptual information saved and the vectors are the actual data stored.

The same information stored in the conceptual matrix is represented in the tree at the right side of the figure. As the number of characteristics may be irregular (as in 2.9) submatrices cover irregular partitions of the grid too. Each of these uneven submatrices is represented in the tree on the right as the total sum of its elements beginning at the root with the

whole amount of products sold (the sum of all cells in the bigger matrix) and descending for each one of its divisions. This decomposition does not end until all branches have reached empty submatrices.

As this structure only stores an aggregated value in each intersection of same level elements of different dimensions, it can be easily added another level to any hierarchy by dividing all the elements of a level in just one element (itself); thus, creating a new level identical to the previous one. This aspect could be useful to match elements of different strati (e.g. "Cold desserts" in "Fukuoka") in case any query requires more flexibility.

The described tree is stored through several compact data structures to minimize space usage. *LOUDS* (see Section 2.1.2) is used to encode the dimensional hierarchies in a compact fashion ($D_{regions}$ and $D_{products}$) while sequence S contains the full list of the values of the matrix, from the total accumulation of all submatrices to each cell value. Besides, fast searches through the structure are provided by *rank* and *select* operators over aligned bitvectors B_n, in charge of inner nodes, and B_b, delimiting the branches of the tree. The former traverses the tree level-wise saving a 1 if the node has offspring, whereas the latter preserves the original tree distribution for navigational purposes.

CMHD will be used as a baseline in Chapter 5 in order to test the results of our proposed aggregated structures.

Chapter 3

Application contexts

The aim of this research is to present new general-purpose solutions to common problems in several different contexts. As a matter of landing this conceptual ideas, this chapter offers details of different real application contexts where our proposals improve the current solutions. These environments will be used in following chapters to explain in detail the behaviour of each proposed structure and its experimental evaluation.

- **Text compression**. A well-known context that needs almost no explanation. Being texts one of the main containers of information, it is capital to process, manipulate, store and handle them using as little space and time as possible.
- **Public transportation**. Managing the passenger flow information in a city is not as trivial as it could be thought. Passenger trip tracking, balanced load of subway lines, passenger preference between several means of public transport or the effective design of bus lines are complex problems that do not have a unique bulletproof solution. New contributions are necessary to solve these problems building a brand new framework of action.
- **Mobile workforce management**. Logistics is also a complex field that has been widely nurtured by technological advances in the last decades. Within this large area, our work focuses on the mobile workers that are needed in any logistics project; tracking the

sequence of their activities and analyzing the obtained information could lead to vital hidden information for companies and to optimize decision-making procedures.

3.1 Text compression

Recalling the first example described in the introduction of this manuscript, text compression has always been a target for mankind, we have spent our entire history trying to fit texts in the smaller possible space in order to simplify the exchange of ideas while reducing costs, weights and efforts. Even so, improvements in this field were slightly frozen until the arrival and increasing popularity of text databases which reopened the search for the most compact text representation. However, the goals did not change over time, text representation compression techniques should not only reduce the storage needs drastically, but also handle texts efficiently in compressed form.

For the ideas proposed in this text, lossless compression is the only option as we cannot afford to compromise data quality not being able to restore the original information. Traditionally, there are two well-known types of lossless text compression methods:

- **Dictionary methods:** These compressors select sequences of symbols and encode each sequence as a token using a dictionary; thus, they are also known as variable-length-symbols-to-fixed-length-codes compressors. The dictionary holding the symbol sequences may be either static or dynamic. The former is stable, occasionally allowing additions but never deletions, whereas the latter stores sequences previously found along the input stream, enabling additions and deletions as new input is being read.

- **Statistical methods:** These compressors use a statistical data model, so that the compression quality they achieve depends on the suitability of the model. Therefore, they are fixed-length-symbols-to-variable-length-codes compressors. They can be either semi-statical or dynamic. In the former case, a first text processing permits to build the model of the text and it is followed by a encoding step where each symbol is given a codeword whose length

depends on its frequency. After that, a second pass replaces each original symbol by the corresponding codeword. In the case of dynamic compression, a unique pass is performed. Therefore, the model is built as text is being read. Each time a symbol is processed, it is given a codeword according to the current model of the already processed text and, then, such model is updated so that the encoding schema can provide optimal codewords for subsequent symbols.

The most representative dictionary-based compressors are the Lempel-Ziv family [ZL77, ZL78, Wel84] being the base of many popular compression programs as *gzip*, *p7zip*, the UNIX program *compress* or *GIF*. As a statistical model is not needed, they can perform compression as the text comes since there is no requirement of preprocessing the input. Therefore, each time a substring is read from the source stream, it is searched in the dictionary, and that substring is substituted by its assigned codeword. Strong compression are reached as each codeword is smaller and lighter than the original substring.

On the other hand, the first statistical compressors based on Huffman coding [Huf52] using character-oriented modeling obtained rather poor compression ratios on text collections (compression around 60%). However, when Huffman coding was coupled with a word-based modeler during the late '80s [Mof89] the compression ratio obtained by those semi-static compressors became close to 25% when applied to English texts, and they set the basis to build modern text retrieval systems over them [WMB99]. This boosted the interest of new compressors not only yielding fast decoding/retrieval but also allowing queries to be performed in compressed form.

At the end of the '90s, *Plain Huffman* (*PH*) and *Tagged Huffman* (*TH*) [dMNZBY98, NdMN$^+$00] replaced the bit-oriented Huffman by byte-oriented Huffman to speed up decoding at the cost of loosing compression effectiveness (now around 30%). In addition, *TH* reserved the first bit of each byte to gain synchronization capabilities. Compression ratios worsened to around 34% but random decompression and fast Boyer-Moore-type searches [BM77] became possible.

In the same line, the use of *dense codes* [BFNP07] allowed *(s,c) Dense Code* (*SCDC*) and *End-Tagged Dense Code* (*ETDC*) (see Section 2.2.1)

to not only retain the same capabilities of *TH* but also improve its compression ratios, which became very close to those of *PH*. They also owned a simpler coding scheme that did not depend on the Huffman tree. Indeed, assuming we have n source symbols s_i $(0 \leq i < n)$ with decreasing probabilities, the codeword c_i corresponding to the i-th symbol can be obtained as $c_i \leftarrow encode(i)$, and the rank i corresponding to c_i can be obtained as $i \leftarrow decode(c_i)$. Both *encode* and *decode* algorithms perform in $O(|c_i|)$ time [BFNP08].

Unfortunately, since *PH* is the optimal word-based byte-oriented zero-order compressor, all those efficient and searchable word-based compression techniques could never reach the compression of the strongest dictionary compressors (e.g. *p7zip*). However, those strong dictionary compressors need more time to achieve those compression ratios and do not allow searching for patterns on a compressed text.

This motivated the creation of *V2Vc*, the first word-based compressor [BFL+10] merging both, variable-length-symbols-to-fixed-length-codes compressors (dictionaries) and fixed-length-symbols-to-variable-length-codes compressors (statistical). Thus, variable-to-variable compressor keeps the best characteristics of both worlds obtaining fast searchable compressed texts with good compression ratios.

V2Vc basically uses additional structures to detect "good" phrases, builds a *phrase-book* that included both words and phrases of words and, then, encodes it using *ETDC*. This opened a new door towards generic hybrid compressors. A detailed explanation of this structure can be found in Section 2.2.3.

Nevertheless, some modern scenarios add a new fundamental requirement to classical text compressors: real-time constraints along transmission through a network. Not only compression ratios have to be satisfactory but also speed at compression is critical; besides, there is no text to preprocess in order to create a model (or dictionary) as phrases are created on the fly. Those critical scenarios are not as unusual as it could seem at first sight, on a day-to-day basis we are surrounded by HTTP pages sent by a server during a HTTP session, digital journals streaming book pages or news to electronic readers or even satellites sending real time pictures of space regions. Therefore, one major interest during the development of this thesis was to tackle this question

providing a new *variable-to-variable* general-purpose compressor that fulfills these requirements as it will be explained in Chapter 4.

3.2 Public transportation

Most transport companies have focused on providing helpful information to their passengers regarding the existing offer, hence providing not only information related to their transport network (e.g. maps with the lines, their schedule, etc.) but also, in many cases, real-time information with the actual position of a vehicle, remaining time to destination, remaining time until next vehicle arrives, and so on. Yet, in order to balance their resources, matching the available offer with the actual passengers' demand is a goal that requires gathering information regarding how users move along the transportation network. Obtaining such data necessarily required using some technology to track user's movements along the transport network. Currently, it is not uncommon to find cities having a smart card system [Bly00] where each user has a personal public transport card with which he or she can pay for trips over several means of transport (bus, subway, etc) and also to switch between them. The use of these smart cards lets the transportation system know where and when an individual started a trip, which provides valuable information for the administrators of the system. Indeed, the ending stop of a trip can be also estimated from boarding records alone [Wan11, AAMF16]. Several works have also been presented regarding how travelers switch lines or where they finish their trips in order to completely track down any traveler [MP12, BFG+18]. With the increasing use of these passengers tracking technologies on public transportation networks, it is now becoming possible to gather (or accurately estimate) the actual trips a given user made along the network. This new available information has been exploited in several scopes. For example, to gather the preferences of users on the chosen mean of public transportation to travel through a city [LVBD17], to monitor urban traffic [LJZL17], or to study the traffic effects of the congestion charges introduced in some urban areas around the world [BK15]. Obviously, the constantly growing amount of information regarding the usage of the network brings new space challenges to overcome.

In [BFG+18] a system to track passengers over various bus lines in a compressed way was presented. The system had a model on its own in order to handle individual journeys and to be capable of doing a thorough scrutiny of the routes taken by each person. The most relevant concepts of the model were:

- **Stops:** Places were commuters may get on or alight a vehicle. Stops must have an associated geographical point.
- **Lines:** They are regarded as the ordered sequence of stops where a vehicle (e.g. bus or train) may stop. A line is considered to have a direction, that is, the full round route is stored as two different lines (forward and backwards).
- **Journeys:** They are referred to as vehicle trip, that is, a particular traversal of a transport vehicle over a line. It departs at a given day and time from the first stop of a specific line and follows the complete sequence of stops of that line until the ending stop, allowing passengers to get on and to get off in each stop.
- **Stages**: They are defined as each chunk of a user trip defined by the boarding on a vehicle at a particular *boarding* stop of a given journey from a given line and the alighting at a different stop of the same line and journey.
- **User trips:** The concatenation of one or more stages defines a user trip. It represents a user traveling from an initial stop within a line until the next line-switching stop (initial stage), intermediate stages for each new vehicle the passenger boards (and then alights), and a last stage until reaching the ending stop of the trip.

As illustrated in Figure 3.1, it is possible to represent a public transportation network on a two layer model: the bottom layer serves as the static network including stops ($s_i \in S$) and lines ($l_i \in L$) and the top layer represents the particular journeys ($j_i \in J^l$) made by vehicles that make stops at specific times. The system handles all this information with four arrays:

- Array $lineStop_i$. It keeps the stops that belong to line l_i. Therefore, $s_a \leftarrow lineStop_i[j]$ indicates that stop s_a is the j-th stop of line l_i.

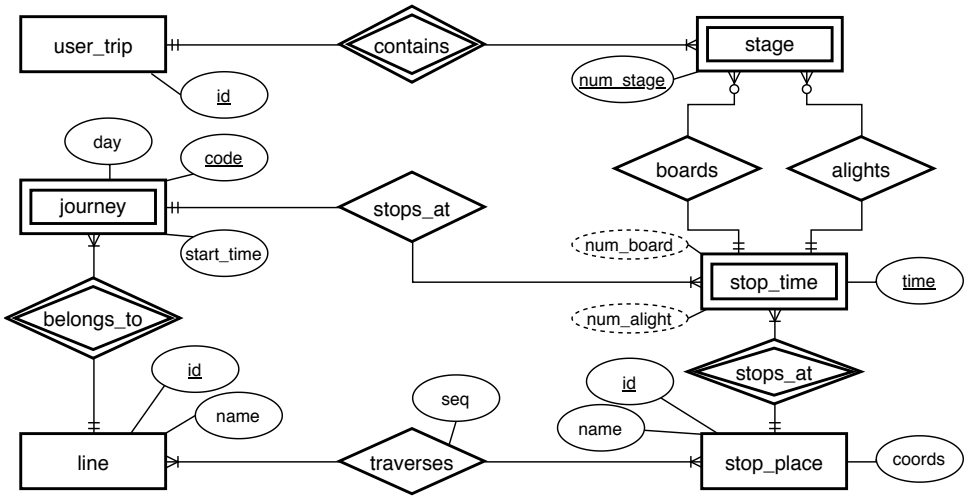


Figure 3.1: Entity-Relationship diagram modeling the elements of a public transportation network and user trips made along it.

Note that given s_a, we could be interested in retrieving its position j among the stops of line l_i; we will refer to such operation as $j \leftarrow lineStop_i^{-1}(s_a)$.

- Array $avgTime_i$. Given a line l_i, $avgTime_i[j]$ indicates the average time (in seconds) that the vehicles corresponding to its journeys need to move from initial stop to the j-th stop.
- Array $initialTime_i$. Considering all the journeys $j_k \in \{1..J^{l_i}\}$ of a line l_i, $initialTime_i(k)$ stores the departure time for the k-th journey of line l_i.
- Array $stopLine_i$. It keeps the non-decreasingly sorted sequence of lines that make a stop at stop s_i. Therefore, $l_x \leftarrow stopLine_i[j]$ indicates that l_x is the j-th line that stops at s_i.

As it is depicted in Figure 3.2, arrays $lineStop_i$, $avgTime_i$, and $initialTime_i$ are stored for each line $l_i \in L$. In addition, an array $stopLine_i$ was kept for each stop $s_i \in S$. It is important to notice the need of two aligned arrays for each line, one with the sequence of stops, and another with the average accumulated time to reach each stop from

the first stop of the line. It could be possible to store the actual time in which each vehicle traversed the stops in a given journey. Yet, as such an accurate time is not relevant, instead it is just kept for each line the time (in average) that a vehicle would require to reach any of the stops from the starting stop of the line, storing only the initial departure time of each journey. This saves much space and still permits to estimate the time when each journey would reach any stop.

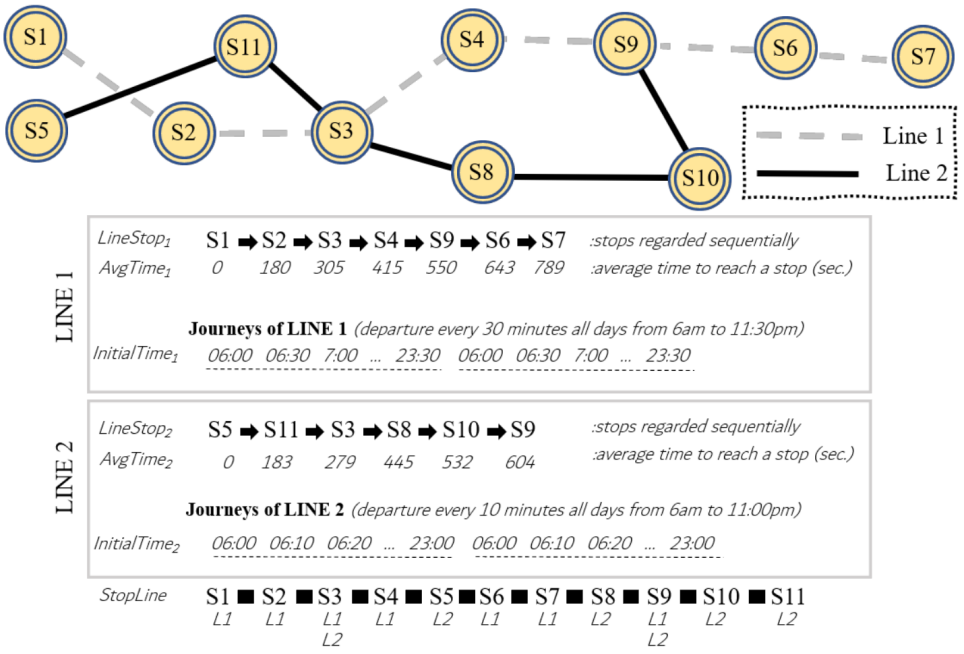


Figure 3.2: Network example where two different lines share stops along their routes.

Figure 3.2 includes an example of a bus network with two lines (1 and 2). For each line we show the stops that compose it (e.g. Line 2 contains the sequence of stops $<S5$, $S11$, $S3$, $S8$, $S10$, $S9>$) and the accumulated times from the initial stop (e.g. the average time to reach the second stop $S11$ from the starting stop of the line is 183 seconds). We also include the starting times for each journey of each line. In this case, the first journey of Line 1 starts at 6:00 am, the second one at

6:30 am, etc. Note that given a line X we have direct access to the information related to the i-th stop. Yet, given a stop, we do not know the line (or lines) it belongs to. To overcome this issue, we include, for each stop Y, the list of lines that include such stop Y. It is referred to as an inverted index for stops in the bottom of Figure 2. To sum up, we saw that to represent the network offer we need: a sequence of stops for each line; a schedule with the starting times of the journeys of each line; and an inverted index to mark the lines each stop belongs to.

In this manuscript, Chapter 5 takes up the baton of this environment adding aggregated capabilities to the original contribution. The scheme displayed in Figure 3.2 represents all the offer information but it is not completely useful until it is compared with the demand (the actual real usage passengers do). Our proposal will tackle the accumulative flow of commuters in order to extract knowledge about the loads of the vehicles during rush hours or the total amount of passengers that boarded a bus in a particular stop during a time window.

3.3 Mobile Workforce Management

There is no doubt about the technological rising in every level of our society, from social networks aware of our complete planning to smart vehicles capable of controlling our driving; there are even fridges that control our diet. In this modern society, we all rely on devices in order to take part in this entangled dough the globalized world has become. It has been reported that more than 65% of adults in advanced economies are owners of a smartphone but it also stands out that most devoted social network users are found in regions with lower internet rates [Pou19].

As a consequence of the vast amount of positioning devices distributed world-wide (e.g. the GPS included in a smartphone), the last years have seen an increasing interest in geographical information and the evolving position of moving objects (vehicles, people, etc.). This has been especially profitable for enterprises whose employees need to visit different locations around the territory to get the work done (e.g. plumbers, art dealers, delivery staff, etc). As technology took over

this area, an endless assortment of tools were served to monitor mobile workers, handle schedules, client support, analysis of performance, etc. The bundle of processes, services, software and networks that make possible to achieve and optimize those activities are called *Mobile Workforce Management* (MWM). Hence, MWM strategies take advantage not only of the historical movement of mobile employees but also the tasks they had performed and how much time they required. Comparing what was scheduled with what really occurred generates useful information for business process management or detecting critical points. For instance, it would be really useful for a delivery company to detect if their workers spend too much time in traffic jams so they could change the planned routes.

Nonetheless, concerns about how to take advantage of trajectories of moving objects through computers are not new. There is already in the '90s a seminal concern about how to mix spatial data and temporal information, works like [Lan89, MK90, FCF92, Lan93] predicted the inception of the spatio-temporal databases. However, it is not until the early '00s when this tendency got consolidated with the arrival of contributions with the aim of laying the groundwork for future references, that is the case of [KGT99, GBE+00, RDW+02] and the European project *Chorochronos* [FGG+99, KSF+03].

At that point in history, the definition of a trajectory was already clear:

Definition 3.3.1. *A trajectory of a moving point is represented by a sequence of tuples. Each tuple contains a position in space p_i and a time t_i. [EGSV99]*

As trajectories of mobile objects started to capture the attention of several fields such as market opportunities [DBS04], robot autonomy [MM98, Edm01], personalized services [ZM01, BFL+03], military defense [BHL+00] or even the outbreak of location privacy issues [MFD03]; scientists realized about the significance of the analysis of this kind of information. However, one variable was missing in the equation. They knew the time (when) and the location (where) but they were not able to know the activity that the mobile object was performing

(what). This idea began to take shape when researchers focused in the behavioural study of moving objects.

Mountain and Raper [MR01] focused on the segmentation of trajectories into small manageable pieces defined by sequential points matching a particular predicate. This is how they coined the term *episode*.

Definition 3.3.2. *An episode represents a discrete time period for which the user's spatio-temporal behaviour was relatively homogeneous. [MR01]*

As a landing example, they used their brand new approach to distinguish between the transport mean used by a user during his/her trajectory: on foot (low speeds, high sinuosity...), low-speed motor (higher speeds, less sinuosity, more network constraints...), etc. Vazirgiannis and Wolfson [VW01] endorsed the relevance of this direction by introducing a new spatio-temporal model including causal aspects of mobile objects.

In [KRS04] an example of a conceptual model enriched with geospatiotemporal annotations was presented, introducing common basic requirements for upcoming geospatial semantic models:

- Allow the data analyst to model non-geospatial, non-temporal, geospatial, and temporal aspects of the application in a straightforward manner.
- Provide a framework for expressing the structure of spatio-temporal data that is easily understood and communicated to the users.
- Support a mechanism for a methodical translation into implementation-dependent logical models.
- Include a mechanism to represent various spatial and temporal granularities (e.g. minute, second, degree).

Shortly after, a concern about adapting trajectories to the decision making context begun. Contributions as [OOR+07] or [PRD+08] tried to build the foundation taking advantage of the already known general purpose data warehouses and the *Online Analytical Processing* queries (see Section 2.4.1). Establishing the latter two goals: to facilitate knowledge discovery from Trajectory Data Warehouses and to support

high-level OLAP analysis. A wider overview of the situation at the beginning of the '10s can be found in [PSR+13]. Although, the complexity of this field and its relatively recent debut brought several approaches along the last decade [VZ13, WFR+13, TTFR15, OA16]; even one new proposal has been published as this lines are being written [Kwa20].

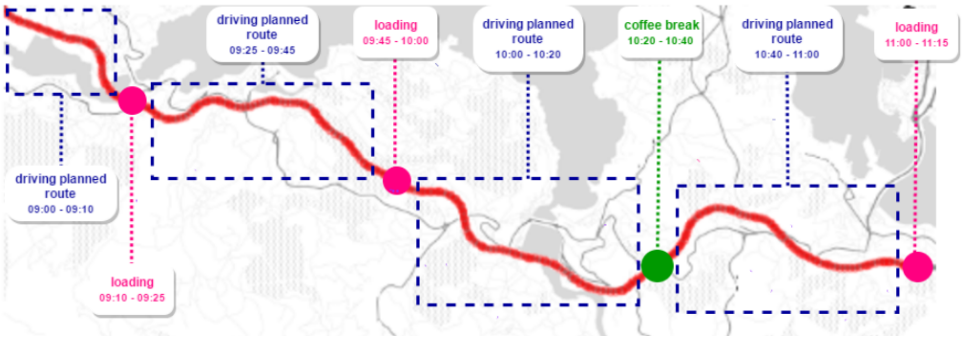


Figure 3.3: Trajectory annotated with semantic activities. [BLPP17]

Following these lines, [BLPP17] proposed to collect the full range of data captured by the sensors of the smartphones carried by the workers aiming to reach largely improved information. This contribution introduced a module for MWM systems able to annotate trajectories of employees with high level activities (e.g. taking a break, driving on planned route, etc.) using the *episode* approach. This achievement is possible by analyzing the trajectories and synchronizing, for each of them, the data received through three dimensions: the sensors of the mobile device (location, direction, acceleration, etc.), the MWM context (scheduled tasks, task type, etc.) and the geographical information of the domain (headquarters, offices, clients, etc.). At the end of the process, this module provides the semantic trajectory information obtained for each worker trip on a daily basis.

This system was first tested in a local transportation company, its business is organic waste management, and they have a fleet of several trucks roaming around the region looking for new waste to collect. Undoubtedly, they have to deal with some of the most common trajectory problems; but, in this work the topic at hand will be the activities carried

out by each one of the truck drivers during their workday. Reducing the usual behaviour of a regular truck driver to a collection of nine relevant activities:

- A_1 Being at headquarters.
- A_2 Working at a customer place (loading).
- A_3 Normal transit on planned route.
- A_4 Slow transit on planned route.
- A_5 Normal transit out of planned route.
- A_6 Slow transit out of planned route.
- A_7 Taking a break.
- A_8 Undefined/unknown activity.
- A_9 Inactive.

Figure 3.3 shows an example of a semantic trajectory annotated with these high level activities. Once the segments of each mobility worker trajectory are delimited and enriched with activity information the data is ready to be exploited in order to extract some advanced knowledge pursuing decision making improvements. Specifically, Chapter 5 and Chapter 6 will focus on extract aggregated knowledge efficiently (e.g. if drivers spent too much time at a costumer facility maybe it is time to check the loading procedure) and on indexing the activity events suitably to the query criteria (e.g. if several drivers traversed non-planned routes after been driving in slow transit perhaps it is time to check the planner algorithm).

Chapter 4

Dynamic variable-to-variable compression (D-V2V)

This chapter introduces our proposal, *D-V2V*, a new dynamic (one-pass) variable-to-variable compressor. Variable-to-variable compression aims at using a modeler that gathers variable-length sequences of input symbols (words) and a variable-length statistical encoder that assigns shorter codewords to the more frequent symbols. In *D-V2V*, we process the input text word-wise to gather variable-length sequences of symbols that can be either terminals (single words) or non-terminals, subsequences of words seen before in the input text. Those input symbols are set in a vocabulary that is kept sorted by frequency. Therefore, those variable-length sequences of symbols can be easily encoded with dense codes. *D-V2V* permits real-time transmission of data, i.e. compression can begin as soon as the text is available to be transmitted. Our experiments show that *D-V2V* is able to overcome the compression ratios of the *V2Vc*, the state-of-the-art semi-static variable-to-variable compressor, and to almost reach *p7zip* values. It also draws a competitive performance at both compression and decompression.

In this chapter we detail our new proposal, Section 4.1 details the compression and decompression procedures, Section 4.2 shows performance comparisons between our proposal and well-known compressors and conclusions can be found in Section 4.3.

4.1 Dynamic variable-to-variable compressor

A text can be seen as a particular sequence of events where each event is the occurrence of a word. These odd events have also several dimensions such as syntactic category, order inside a phrase, number of letters, etc. An interesting approach to word category order analysis can be found in [BCN10] where a new method to classify the different parts of XML texts (tags vs content) is presented.

Although, when a text written in natural language is being created or transmitted (e.g. writing a text message), the incoming order is the correct one; that is, there is no need to apply reorganization techniques, the main concern should be just to compress the original input as much as possible without losing any information. Our proposal tackles this problem being capable of compressing a text *on the fly* as it is input by the sender/compressor.

Our compressor parses the input text to either detect sequences of words that occurred before or new words and either sends to the receiver a codeword referring to an existing sequence or notifies the occurrence of a new word so that te receiver can keep synchronization with the receiver. Since more frequent symbols are given shorter codewords, and large sequence of words can be encoded with a unique codeword, compression is achieved.

Therefore, *D-V2V* is a dynamic (one-pass) compressor that processes the input text and gathers symbols that represent sequences with a variable number of words. Accordingly, we define a symbol as:

Definition 4.1.1. *A* symbol *is a variable-length sequence of consecutive events sorted by its occurrence order. In a text, symbols are* phrases *composed by one or several* words*.*

We use a sort of trie to help the parser to detect sequences of words that occurred before. We keep those symbols sorted by frequency. In this way, we can use the *ETDC* coding algorithm (see Section 2.2.1) to encode them directly from their positions in the vocabulary. The decompressor/receiver is simpler because it only has to decode the received codewords and to keep the table of symbols sorted by frequency (synchronized with the sender).

In the next subsections we conceptually describe the *parser* and the *encoder* procedures of the sender/compressor component, and also the *decoder* procedure that is the core of the receiver/decompressor component.

4.1.1 Parsing algorithm

Our parser scans the input text and splits it into simple and compound tokens. Therefore, those tokens can be:

- *terminal symbols*. Those representing just one word. They are created when a new word is parsed.
- *non-terminal symbols*. Those composed by two different symbols, which can be terminals or non-terminals. Therefore, each non-terminal, represents at least a sequence of two words.

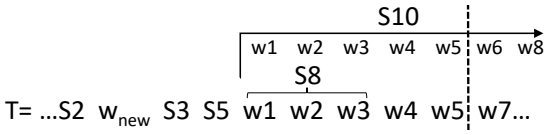


Figure 4.1: Non-terminal creation example.

During the parsing step, the sender reads the text one word at a time. Depending on the previous occurrences of the read word the algorithm will follow one of these paths:

- If the next read word was not in our vocabulary, two symbols are created:
  1. A terminal symbol S_{new} which represents the new word (the sender will notify the receiver about this new word as we will show in the next section).
  2. A non-terminal S_{new+1} which appends S_{new} to the previous sent symbol.

  For example, in Figure 4.1, after sending S_2 we read the new word w_{new}. Therefore, a terminal symbol S_{new} is created for that word and then a non-terminal symbol S_{new+1} is created for $S_2||S_{new}$.

- If the next read word is a prefix of any symbol from the vocabulary, we store such word in RS (read sequence). We keep reading the text word by word and append those words to RS until RS becomes an unknown sequence. At this moment, we send the symbol which corresponds to the longest known prefix of RS. Then, a new non-terminal symbol containing the current sent symbol and the previous one is created.

In the example of Figure 4.1, let us assume that S_8 is a non-terminal that contains the words $w_1w_2w_3$, and S_{10} contains $w_1w_2w_3w_4w_5w_6$, where S_{10} is the unique non-terminal symbol starting by $w_1w_2w_3w_4w_5$. After sending the symbol S_5, we are at w_1 and we read the next words $w_1 \dots w_5$ one word at a time. We keep reading words until we reach w_7. At that moment, $RS \leftarrow \underline{w_1w_2w_3w_4w_5}w_7$ is not a prefix of the sequence in S_{10}; therefore, we stop processing the text. Note that, since the symbol containing the longest known prefix of RS corresponds to S_8, we send S_8 (the way to encode S_8 will be explained in detail in the next section) and we create a new non-terminal symbol S_{new} for $S_5||S_8$. We will continue parsing from w_4 on.

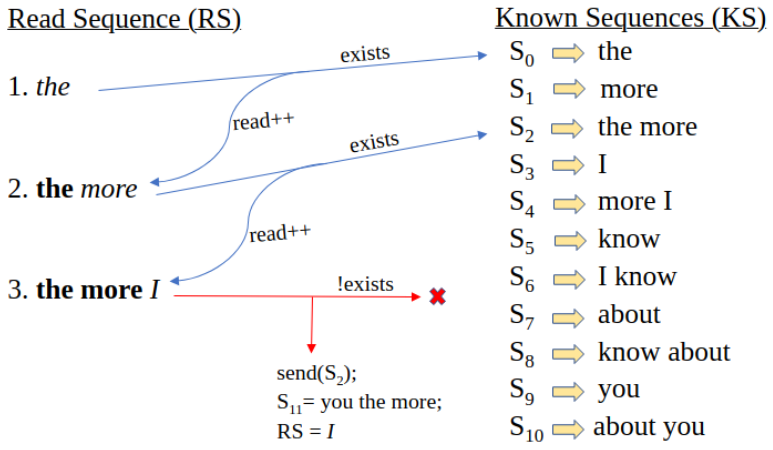


Figure 4.2: Sequence of events during the parsing of a text with D-V2V. KS stores the words and phrases that appeared previously in the text while RS is a buffer trying to obtain the largest known sequence for the incoming text.

In practice, we are using a set of *known sequences* KS which stores every previously processed terminal and non-terminal symbols.

If we are sending the message "*the more I know about you the more I know about me*", at the beginning we have $KS = \emptyset$ and we read the first word "*the*". Since KS is empty, there is no sequence which starts with "*the*", thus we add it to KS, at position $i = |KS| = 0$, the symbol $S_i = S_0 =$ "*the*", and we send S_0 to the receiver. Then we read "*more*", which is also a new word. Now we have to add both the new terminal symbol $S_1 =$"*more*" (S_1 is also sent to the receiver) and the new non-terminal symbol $S_2 =$ "*the more*" to KS.

As displayed on Figure 4.2, after processing the word "*you*", KS is composed by {S_0:"*the*", S_1:"*more*", S_2:"*the more*", S_3:"*I*", S_4:"*more I*", S_5:"*know*", S_6:"*I know*", S_7:"*about*", S_8:"*know about*", S_9:"*you*", S_{10}:"*about you*"} and we continue reading "*the*". Since the current read sequence $RS =$"*the*" exists in KS, we read the next word and append it to RS. Now, $RS =$"*the more*" matches the symbol S_2 stored in KS. In the next step, we update RS to "*the more I*". Since that sequence is not included in KS, we send S_2 to the receiver and we create a new non-terminal that includes the previous and the current sent symbol: $S_{11} = S_9||S_2 =$"*you the more*".

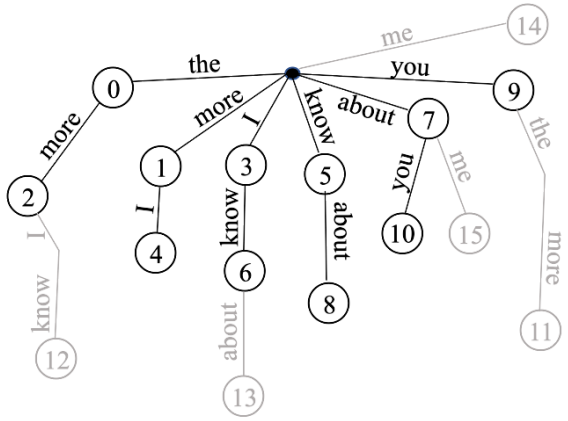


Figure 4.3: Tree used during compression when processing the sentence: "`the more I know about you the more I know about me`". The black branches in the tree represent its stage after processing the word "`you`".

We need a mechanism to check if RS is within the set of known sequences KS and to obtain its symbol identifier, i.e. its rank in KS. In order to perform those tasks efficiently we use a structure based on the Patricia-tree [Mor68], where each branch represents a sequence, and all the sequences that start with the same prefix descend from the same node.[1] The last property is important, as it allows us to search incrementally for the longest sequence contained in RS. For example, after reading the second "*the*" (RS ="*the*") we access to the tree in Figure 4.3 and go through the branch labeled with "*the*" reaching the node-⓪, which contains the identifier of S_0. Then, we read "*more*" (RS ="*the more*"), hence we descend from node-⓪ to node-②, which contains the symbol S_2. Finally, we read "*I*" (RS ="*the more I*"). Since we cannot descend from node-②, the longest known sequence is "*the more*" and its associated symbol is S_2.

4.1.2 Encoding procedure

Every parsed symbol must be encoded and sent to the receiver. We encode them using the encoding scheme of *End-Tagged Dense Code* (*ETDC*). We need to keep track of the number of times each symbol was sent (frequency) because, following *ETDC* encoding procedure, the codeword of a symbol depends only on its position within the vocabulary sorted by frequency. Recall *ETDC* assigns the shortest codewords to the most frequent symbols. Note that, each time a symbol is sent, its frequency is increased, and consequently, the codewords assigned to the symbols may change. For each parsed symbol S_i we send one codeword. In addition, when we send a terminal symbol for the first time ($S_i = S_{|KS|}$), we send that codeword, which acts as an escape codeword, followed by the word in plain format.

In order to encode the symbols, we use a *codebook* where we store all the information required to compute the codeword of each symbol. Each entry in the *codebook* corresponds to a symbol S_i and stores a tuple $\langle left, right, freq, voc \rangle$ as displayed in Figure 4.4:

- $left$ and $right$ represent the right and left nodes of each symbol. If the symbol is a non-terminal, $left$ and $right$ are pointers to the

[1] We implemented a bit-oriented tree where unary paths are stored in their parent node.

entries of the *codebook* where the left and right symbols of the non-terminal are stored. Otherwise, if the symbol is a terminal, $left$ stores the word itself and $right$ is set to -1.

- $freq$ stores the frequency of the symbol. It is important to notice that every non-terminal symbol is created with frequency 0.
- voc holds the position of the symbol within the vocabulary sorted by frequency.

The codeword c corresponding to the symbol S_i stored in the i-th entry of the codebook is obtained as $c \leftarrow ETDC.encode(voc[i])$.

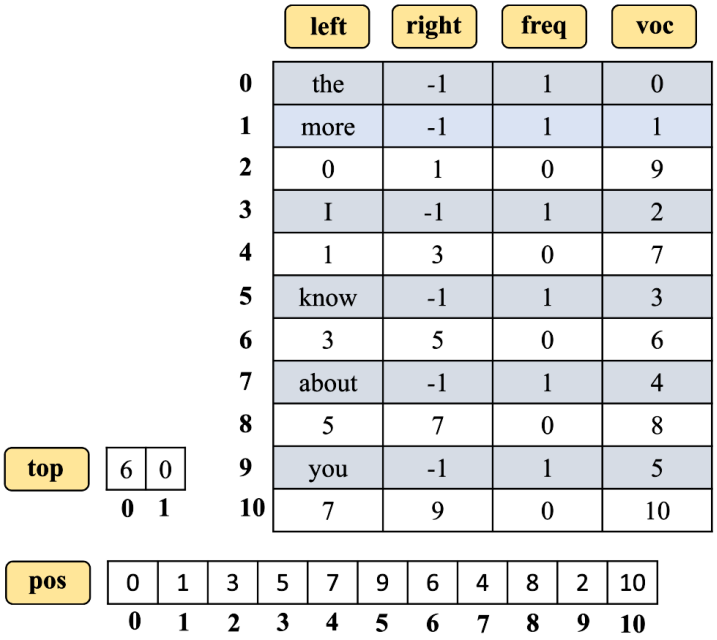


Figure 4.4: Structures used during compression when processing the sentence: "`the more I know about you the more I know about me`". *left* and *right* are either pointers to previous occurrences or containers of a new word and a void pointer, *freq* is the frequency of each symbol, *voc* represents the codes to be sent while *pos* and *top* are auxiliary structures to simplify updates and insertions.

To keep the vocabulary sorted by frequency we use two arrays: *pos* and *top*. Array *pos* keeps the symbols sorted by frequency in decreasing

order. Actually, $pos[i] = j$ indicates that the i-th most frequent symbol is stored in the j-th entry of the *codebook*. Consequently, note that all the symbols with the same frequency are pointed to from consecutive entries in *pos*. Array *top* contains a slot for each frequency value. For every possible value of a frequency f, $top[f] = x$ means that the first symbol with frequency f is at position x in *pos*. For example, in Figure 4.4 the array *top* indicates that the codewords of frequency 1 start at position 0 within *pos*. We can observe that the gap between $top[0]$ and $top[1]$ is 6, thus $pos[0..5]$ point to the 6 entries within *codebook* that hold all the symbols with frequency 1.

With the help of the arrays *pos* and *top*, we can easily add the new symbols at the end of *codebook*. Those arrays are also necessary to update the frequencies and positions in the vocabulary in $O(1)$ time without reordering the whole *codebook*. In our example, after inserting "*you*", the table remains in the state of Figure 4.4. As we explained before, in the next step we send S_2 ="*the more*" which is the symbol at position 2. Therefore, we increase the frequency at $freq[2]$ to 1. We look for the position of the first symbol with frequency 0 by using $top[0] = 6$. After that, we swap $voc[2]$ and $voc[6]$, so now $voc[2] = 6$ and $voc[6] = 9$. As we changed *voc*, we also have to update *pos* accordingly. Therefore, we modify $pos[6] = 2$ and $pos[9] = 6$. Finally, as now the list of symbols with $f = 1$ has been increased by one, the list of words with $f = 0$ starts one position further, so we update $top[0] = 7$.

To clarify this process, Table 4.1 shows the conceptual algorithm for processing the sentence "the more I know about you, the more I know about me" step by step. First, KS is empty, so we process the first word and keep reading as we do not have anything else in our known set. *Step 2* and *step 3* add a new read world to KS and build a new symbol appending this new word to the word processed immediately before. As "I" is read, a new symbol is created; although, the known sequences must be ordered by frequency so a swap between the new word and the synthetic symbol created in the previous step is needed. It is important to notice that each word in the symbol "the more" has been processed as different symbols but the entire phrase has only been created afterwards (i.e. freq = 0). *Step 5* creates another synthetic symbol with the last two symbols processed ("more I"). *Step 6* handles

Step	Read sequence	Known sequences	Sent
1	the	S_0=the	the
2	more	S_1=more	more
3	$\emptyset$	S_2 = the more	$\emptyset$
4	I	S_2 = I; S_3 = the more	I
5	$\emptyset$	S_4 = more I	$\emptyset$
6	know	S_3 = know; S_5 = the more	know
7	$\emptyset$	S_6 = I know	$\emptyset$
8	about	S_4 = about; S_7 = more I	about
9	$\emptyset$	S_8 = know about	$\emptyset$
10	you	S_5 = you; S_9 = the more	you
11	$\emptyset$	S_{10} = about you	$\emptyset$
12	the	S_0	$\emptyset$
13	the more	S_9	$\emptyset$
14	the more I	$\emptyset$	S_9
15	$\emptyset$	S_6 = the more; S_9 = I know	$\emptyset$
16	$\emptyset$	S_{11} = you the more	$\emptyset$
17	I	S_2	$\emptyset$
18	I know	S_9	$\emptyset$
19	I know about	$\emptyset$	S_9
20	$\emptyset$	S_7 = I know; S_9 = more I	$\emptyset$
21	$\emptyset$	S_{12} = the more I know	$\emptyset$
22	about	S_4	$\emptyset$
23	about me	$\emptyset$	S_4
24	$\emptyset$	S_0 = about; S_4 = the	$\emptyset$
25	$\emptyset$	S_{13} = I know about	$\emptyset$
26	me	S_{14} = me	me
27	$\emptyset$	S_8 = me; S_{14} = know about	$\emptyset$
28	$\emptyset$	S_{15} = about me	$\emptyset$

Table 4.1: *D-V2V* parsing process step by step for the text "the more I know about you the more I know about me". *RS* is a buffer trying to obtain the largest known sequence, *KS* is the symbol dictionary and last column reflects what the receiver gets.

a new read word, introducing it in the KS list and updating the list by swaping the new arrival with the least frequent symbol with smaller code (S_3).

This procedure is repeated until *step 12* where an already known symbol is found in the text (see Figure 4.2). As "the" is already in KS (S_0) we keep reading. Again, "the more" is a known sequence (S_9) so we read the next word. Since "the more I" does not exist inside KS, first we handle the known symbol "the more" (*step 15 and step 16*) and then "I" as the next symbol. Remaining steps of the example can be easy followed applying this same procedure.

4.1.3 Receiver procedure

The receiver works symmetrically to the sender. It decodes either a codeword corresponding to a known symbol or an escape codeword followed by a new word (terminal) in plain form. After decoding a symbol, we also add a new non-terminal composed of the last two decoded symbols to keep the codebook synchronized with the sender. This allows the receiver to rebuild the same model handled by the sender and to recover the original text. To carry this out the receiver also has a *codebook* and an auxiliary *top* array. The *codebook* is composed of columns *offset*, *length*, and *freq*.

Each time we create a new symbol (i.e. we either received a new word or we created a new non-terminal), we set in *offset* a pointer to the position of the first occurrence of that symbol within the decompressed text. The length (in chars) of the text represented by such symbol is kept in *length* and $freq$ stores the frequency of the symbol.

In Figure 4.5 we can observe the state of the receiver after decompressing "*the more I know about you the more*". Now the sender transmits the symbol S_8 ="*I know*" encoded with *ETDC*. The receiver decodes the codeword into 8. It accesses to the *codebook* at position 8 and retrieves $offset[8] = 9$ and $length[8] = 6$, thus the decoder recovers the sequence "*I know*" from the decompressed text from position 9 to 14. Afterwards, we update the decompressed text to "*the more I know about you the more*"||" *I know*". Then, we increase $freq[8]$ and we swap the rows in the *codebook* at positions 8 and $top[0] = 7$ (recall $top[0]$ is the first row with frequency equals to 0). Finally, since the first row

with frequency equals to 0 is moved to the next position, we update $top[0] = 8$.

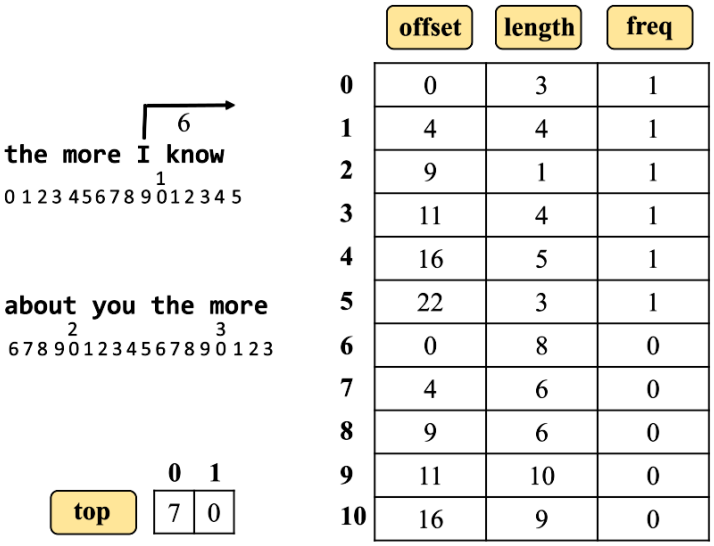


Figure 4.5: Structures used during decompression when processing the sentence: "`the more I know about you the more I know about me`". It is simpler than the sender as it only has to be synchronized with it keeping the table ordered by frequency.

Again, a full decompression example can be found in Table 4.2. *Step 1* and *step 2* just receive a plain word and store it in KS. Afterwards, a new symbol linking both is created with no frequency. In *step 5*, the plain word "I" arrives so it is saved in KS swapping positions with the least frequent symbol with smaller code (S_2). Next step generates a new synthetic symbol. The procedure keeps this pattern until *step 12* where a non-terminal symbol is received instead of a plain word. As S_9 is already a known sequence, we can just translate the code to the plain phrase "the more". As it is shown in the table, it is important to emphasize that the frequency of S_9 (a generated symbol not received until now) is updated at this point to keep our table synchronized with the *codebook* of the sender. *Step 13* trivially generates a new non-terminal symbol in our KS but in *step 14* S_9 arrives again. As the code had been updated since the last arrival, it now translates into "I

Step	Received	Known sequences	Decompressed
1	the	S_0=the	the
2	more	S_1=more	more
3	$\emptyset$	S_2 = the more	$\emptyset$
4	I	S_2 = I; S_3 = the more	I
5	$\emptyset$	S_4 = more I	$\emptyset$
6	know	S_3 = know; S_5 = the more	know
7	$\emptyset$	S_6 = I know	$\emptyset$
8	about	S_4 = about; S_7 = more I	about
9	$\emptyset$	S_8 = know about	$\emptyset$
10	you	S_5 = you; S_9= the more	you
11	$\emptyset$	S_{10} = about you	$\emptyset$
12	S_9	S_6 = the more; S_9 = I know	the more
13	$\emptyset$	S_{11} = you the more	$\emptyset$
14	S_9	S_7 = I know; S_9 = more I	I know
15	$\emptyset$	S_{12} = the more I know	$\emptyset$
16	S_4	S_0 = about; S_4 = the	about
17	$\emptyset$	S_{13} = I know about	$\emptyset$
18	me	S_8 = me; S_{14} = know about	me
19	$\emptyset$	S_{15} = about me	$\emptyset$

Table 4.2: *D-V2V* decompressing step by step the sentence "the more I know about you the more I know about me". It is important to note how the column "Received" is synchronized with the column "Sent" in Table 4.1.

know" its associated table update and non-terminal symbol generation. Decompression continues until the last symbol is received (*step 18*) and processed (*step 19*).

4.2 Experiments

As information retrieval became one of the predominant areas in the technological world, there was a need to establish some shared benchmarks that could act as a common reference in order to classify new algorithms and contributions. Therefore, the *Text REtrieval Conference (TREC)* datasets were created and used from then on as canonical testing texts.

We used three text datasets from TREC-2 and TREC-4 named Ziff Data 1989-1990 (ZIFF), Congressional Record 1993 (CR) and Financial Times 1991 (FT91). In addition, we created a large dataset (ALL) including ZIFF and AP-newswire from TREC-2, as well as Financial Times 1991 to 1994 (FT91, FT92, FT93, and FT94) and ZIFF from TREC-4. We also included three highly repetitive text datasets: *world_leaders.txt* (WL), *english.001.2.txt* (ENG) and *einstein.en.txt* (EINS) from PIZZACHILI.[2]

We performed experiments to compare the compression effectiveness as well as the performance at compression and decompression of *D-V2V* with those of *DETDC*[3] and *V2Vc*, which are respectively the previous dynamic word-based technique that makes up the basis of *D-V2V*, and the state-of-the-art when considering semi-static variable-to-variable compression based on dense codes. In the case of *V2Vc*, we considered the two variants proposed in [BFL+10] (see Section 2.2.3), i.e. *V2Vc* and $V2Vc_H$. The former one uses a simpler heuristic to gather phrases, whereas the latter uses a more complex heuristic that yields better compression at the cost of increased compression time. Given that in *D-V2V* new words are sent in plain form, we included two variants of *V2Vc* and $V2Vc_H$ that, as in [BFL+10], respectively represent the words in the vocabulary in plain form or compressed with *lzma*. In addition, we have included some of the most well-known representatives from different families of compressors: *p7zip* and *lzma*,[4] *bzip2*,[5] and an implementation of *re-pair*,[6] coupled with a bit-oriented Huffman.[7]

[2] http://pizzachili.dcc.uchile.cl
[3] http://vios.dc.fi.udc.es/codes
[4] http://www.7-zip.org
[5] http://www.bzip.org
[6] http://raymondwan.people.ust.hk/en/restore.html
[7] https://people.eng.unimelb.edu.au/ammoffat/mr_coder/

Our test machine is an Intel(R) Core(TM) i7-3820@3.60GHz CPU (4 cores - 8 siblings) with 64GB of DDR3-1600Mhz. It runs Ubuntu 12.04.5 LTS (kernel 3.2.0-126-generic). We compiled with gcc 4.6.4 and optimizations `-O9`. Our time results measure CPU user time.

4.2.1 Space requirements and memory usage

The compression ratios obtained are displayed in Table 4.3. We can see that *D-V2V* is able to improve the results of *V2Vc* (and $V2Vc_H$) in all datasets (results with '–' indicate failed runs) with the exception of the *lzma* version in dataset FT91. This is remarkable since we are sending new words in plain form, while the best values or *V2Vc* are drawn when it encodes the vocabulary of words with *lzma*.

		v2vc	$v2vc_H$	v2vc	$v2vc_H$		Repair	lzma			bzip2	**Size Plain**
	Detdc	*lzma*	*words*	*plain*	*words*	D-v2v	*+sHuff*	*def*	*-9 -e*	p7zip	*def*	**(KB)**
FT91	35,64	27,15	26,65	30,11	29,61	28,60	24,00	25,50	25,25	25,52	27,06	**14.404**
CR	31,99	23,55	23,13	24,73	24,31	22,86	20,16	22,05	20,83	21,63	24,14	**49.888**
ZIFF	33,79	24,01	23,60	24,66	24,25	23,14	20,33	23,40	21,64	22,98	25,10	**180.879**
ALL	33,66	22,81	–	23,39	–	22,67	–	23,23	21,34	22,80	25,98	**1.055.391**
WL	15,06	4,13	–	4,44	–	2,90	1,43	1,30	1,11	1,39	6,94	**45.867**
ENG	35,21	–	–	–	–	5,52	2,17	0,55	0,55	0,55	3,73	**102.400**
EINS	30,14	0,97	–	0,98	–	0,27	0,07	0,07	0,07	0,07	5,17	**456.667**

Table 4.3: Compression ratio (%) with respect to the size of the plain text dataset.

As expected, by using not only words in the vocabulary of symbols allows *D-V2V* to overcome the original *DETDC* by more than 10 percentage points in regular English datasets and completely blows *DETDC* out in repetitive collections. On regular texts, *D-V2V* and *p7zip* obtain similar values on the largest dataset, yet in the other datasets the fact of exploiting char-based rather than word-based regularities benefit *p7zip*, *lzma*, and *re-pair*. In repetitive text collections, char-level repetitiveness is higher than at word-level and, in addition, the fact of sending words in plain form harms *D-V2V* compression. In practice, even though compression is good in *D-V2V*, it is typically far from *re-pair*, *p7zip*, and *lzma*.

In Table 4.4, we can see memory usage at compression time. In this case, our current implementation of the trie in *D-V2V* requires lots of memory. At decompression time, we only have to deal with the

	Text	Detdc	v2vc *lzma*	v2vcH *lzma words*	v2vc *plain*	v2vcH *plain words*	D-v2v	Repair *+sHuff*	lzma *def*	lzma *-9 -e*	p7zip	bzip2 *def*
Compressor	**FT91**	24	52	52	52	52	1,194	380	94	192	165	7
	CR	53	157	157	157	157	2,635	1,286	94	504	193	7
	ZIFF	126	625	625	625	625	10,509	4,585	94	674	193	7
	ALL	207	3,509	–	3,509	–	46,160	–	94	673	193	8
	WL	49	255	–	255	–	478	1,268	94	469	193	7
	ENG	85	–	–	–	–	1,953	2,512	94	674	193	7
	EINS	152	44,821	–	9,851	–	6,521	10,859	94	674	193	8
Decompressor	**FT91**	4	10	20	20	10	20	13	9	15	17	4
	CR	6	65	65	65	65	51	30	9	50	19	4
	ZIFF	14	121	119	121	119	177	79	9	65	19	4
	ALL	57	378	–	378	–	931	–	9	65	19	4
	WL	5	52	–	51	–	6	11	9	46	18	4
	ENG	9	–	–	–	–	23	31	9	65	18	4
	EINS	14	73.81	–	73.81	–	5	5	9	65	18	4

Table 4.4: Memory usage (in MiB) at compression and decompression.

codebook (the size of *top* is negligible), and the memory usage becomes much more reasonable. Yet, the number of entries in the *codebook* is still very high in most datasets: {1,6M@FT91}; {4,3M@CR}; {15,4@ZIFF}; {81,2M@ALL}; {0,44M@WL}; {1,8M@ENG}; {0,3M@EINS}.

4.2.2 Compression and decompression times

Compression and decompression times are described in Table 4.5. *D-V2V* is faster at compression than *p7zip*, *lzma*and *re-pair*. It is on a par with $V2Vc_H$and it is slower than *bzip2*. Of course *DETDC*, which has not to deal with the detection of seen subsequences, is much simpler and faster than *D-V2V*.

At decompression, we can see that again *D-V2V* is the fastest technique in all cases, with the exception of *DETDC* and *V2Vc* when dealing with non-repetitive English texts. Note that, in this case, *V2Vc* compression effectiveness is similar to that of *D-V2V* and consequently both decode approximately the same number of codewords. However, *V2Vc* has not to perform an *update* procedure after decoding each symbol nor to generate a new non-terminal. *DETDC* has to decode more symbols than *D-V2V* due to its worse compression. Yet, again it is simpler because it does not have to deal with non-terminals, only with words. In the repetitive collections *D-V2V* compresses much more

than *V2Vc* and *DETDC*, which leads to a compressed file with much less codewords than those of *DETDC* and *V2Vc*, and this amortizes the cost of the *update* procedure required after decoding each codeword.

			v2vc	v2vcH	v2vc	v2vcH		Repair	lzma			bzip2
	Text	Detdc	*lzma words*		*plain words*		D-v2v	*+sHuff*	*def*	*-9 -e*	p7zip	*def*
Compr. time	**FT91**	0.15	1.31	2.26	1.28	2.27	5.81	8.37	9.17	10.66	9.06	1.19
	CR	0.53	5.77	19.01	5.72	19.10	19.03	39.84	32.61	44.09	33.67	4.07
	ZIFF	2.12	32.08	257.42	31.94	258.03	86.13	271.02	120.92	177.86	128.99	14.52
	ALL	13.25	292.39	–	289.05	–	573.09	–	711.23	1167.86	768.22	86.71
	WL	0.48	17.93	–	18.05	–	2.96	14.99	8.88	23.60	6.23	2.45
	ENG	1.71	–	–	–	–	13.39	58.97	28.37	57.34	28.31	8.45
	EINS	6.62	33205.00	–	33197.00	–	30.48	205.33	60.98	115.12	57.13	54.95
Decompr. time	**FT91**	0.09	0.08	0.09	0.06	0.06	0.09	0.15	0.18	0.17	0.19	0.47
	CR	0.28	0.22	0.23	0.20	0.21	0.31	0.65	0.54	0.54	0.54	1.54
	ZIFF	1.24	1.01	0.90	0.94	0.86	1.50	2.69	2.13	2.14	2.14	5.83
	ALL	7.68	9.13	–	8.99	–	11.63	–	12.13	12.25	12.15	33.54
	WL	0.16	0.06	–	0.06	–	0.02	0.28	0.05	0.05	0.09	1.00
	ENG	0.85	–	–	–	–	0.14	2.50	0.06	0.04	0.18	4.08
	EINS	3.22	0.33	–	0.24	–	0.05	1.71	0.15	0.15	0.71	9.40

Table 4.5: Compression and decompression times (in seconds).

4.3 Conclusions

This chapter described *D-V2V*, the first dynamic variable-to-variable general-purpose compressor. We showed that *D-V2V* obtains competitive compression ratios (similar to *p7zip*) in English texts and that it is fast at both compression and (mainly) decompression.

We have described the symmetric processes of both compressor (sender) and decompressor (receiver), and how they keep synchronization dynamically. The compressor gathers both words and sequences of words that occurred previously in the text, and encodes them statistically using dense codes. New words are notified explicitly after an escape codeword. When the receiver decodes a codeword (it could belong to either a word or a sequence) it simply outputs that symbol. Finally, both the sender and receiver update their model to: increase the frequency or the sent/received symbol and to run a simple *update()* procedure to kept the vocabulary sorted by frequency at both ends (hence allowing encoding with dense codes). Finally, they add a new non-terminal,

composed of the last two sent symbols, to the vocabulary. This will permit to encode further occurrences of those two symbols with just the codeword associated to the new non-terminal.

Chapter 5

Total matrices (T-Matrices)

Based on the ideas of classic data warehouses and OLAP systems, explained in Section 2.4.1, where precomputed data is stored separatedly in order to boost aggregated queries and improve decision-making procedures, this chapter introduces a compressed structure to achieve the same functionality, opening a new path towards a trajectory data warehouse. We have evaluated experimentally our proposal in two real contexts: bus passengers and truck drivers. To evaluate our proposal under higher stress conditions we generated a large dataset of synthetic realistic trajectories and we tested our system with those data to have a precise idea about its space needs and its efficiency when answering different types of queries.

This chapter is organized as follows: Section 5.1 introduces the details of our proposal, Section 5.2 analyzes the application of our research to the public transport context and displays the results obtained, Section 5.3 performs the same analysis in mobile workforce environments and Section 5.4 recapitulates the key concepts explained during this chapter.

5.1 General-purpose accumulative matrices

Summed Area Tables (SAT) [Cro84] (described in Section 2.4.3) were designed to solve a rather specific problem, yet, the idea of using accumulative matrices also seemed useful for problems within different

contexts. It is not unusual to find systems that need to solve queries over aggregated data quickly and one common solution is to have those aggregations precalculated in a similar way as in *SAT*.

Thus, a generalization of those accumulative matrices is proposed under the name of *Total Matrices (T-Matrices)*. This approach will be used to solve all kind of event-related aggregation problems regardless of the scope. Besides, following the path opened by OLAP systems, *T-matrices* exploit the multidimensional nature of event sequences supporting aggregated queries by each dimension or a combination of them.

The aim of our research is to introduce a solution able to count the number of events appearing in a sequence discriminating by any range or dimension. In order to achieve that, we generate a multidimensional matrix with as many dimensions as characteristics own the events on the sequence. Each cell contains the number of occurrences of an event with that particular combination of dimensions. In the sake of performance optimization, we apply *SAT* to our proposal representing the accumulative matrix.

$$S = E_{A\alpha}\, E_{A\pi}\, E_{B\alpha}\, E_{A\pi}\, E_{C\omega}\, E_{B\pi}\, E_{D\alpha}\, E_{C\omega}\, E_{B\omega}\, E_{C\omega}$$

	A	B	C	D
α	1	1	0	1
π	2	1	0	0
ω	0	1	3	0

Count Matrix

	A	B	C	D
α	1	2	2	3
π	3	5	5	6
ω	3	6	9	10

Cumulative Matrix

Figure 5.1: T-Matrix applied to a generic sequence S of bi-dimensional events.

Thus, reducing the problem to a sequence of bi-dimensional events S to ease the explanation, it is possible to use this structure to reorganize the original sequence in order to re-build it as a bi-dimensional matrix storing the count of each event based on both dimensions. As it can be seen in Figure 5.1, this simple matrix with the count of events involving such dimensions (left) is easily transformed into an aggregated

matrix following the *SAT* approach (right). Once the structure is set, it is straightforward to calculate the sum of any relevant submatrix. Continuing with the example in the figure, the way to calculate the sum of each cell shaded in blue is different for each matrix. The basic matrix on the left needs to add cell by cell every slot within the shaded area ($T = 1 + 1 + 0 + 3 = 5$); on the other hand, the accumulative matrix on the right just needs to apply *SAT*'s formula ($T = 9 - 2 - 3 + 1 = 5$).

T-Matrix

	A	B	C	D
α	1	2	2	3
β	3	5	5	6
δ	3	6	9	10
ε	3	6	10	11
θ	5	10	14	17
λ	6	11	15	19
π	6	11	15	19
σ	7	16	20	25
ψ	7	16	22	29

Diff T-Matrix

	A	B	C	D
α	1	2	0	1
β	2	5	0	1
δ	3	6	3	4
ε	3	6	4	5
θ	5	10	4	7
λ	5	11	4	8
π	5	11	4	8
σ	9	16	4	9
ψ	9	16	6	13

Figure 5.2: An example of how a *T-Matrix* and a *Diff T-Matrix* are binded. In this representation, central column (B) values remain the same and the other columns are calculated as additions (right side columns) or subtractions (left side columns). For instance, $A\lambda = 11$ - $5 = 6$ and $C\lambda = 11 + 4 = 15$.

As the number of event dimensions in a sequence grow, the size of the matrices also grow accordingly. This will increase the magnitude of the quantities stored in it and, as a consequence, it will increment space requirements to handle those greater numbers. Figure 5.2 (left) shows how easily the aggregation values can grow. With the aim of tackling this problem, we have designed a simple improvement over *T-Matrices*. We focus on compression and exchange some query time for a space usage reduction 5.2 (right). Just maintaining the values in the central column (shaded in the figure) and storing the rest of values differentially with respect to it, yields an important reduction in space usage. We

named this approach *Diff T-Matrix*.

Going one step further, it is possible to generalize the *Diff T-Matrix* for the sake of compression. The key idea is to store more than one column as constant absolute values and to save other cells as differentials. Each of these constant columns are called *sampled columns*. We can also take advantage of current computer architectures using sampled rows instead of sampled columns to benefit from the cache locality optimization.

T-Matrix

	A	B	C	D
α	1	2	2	3
β	3	5	5	6
δ	3	6	9	10
ε	3	6	10	11
θ	5	10	14	17
λ	6	11	15	19
π	6	11	15	19
σ	7	16	20	25
ψ	7	16	22	29

Blocks T-Matrix

	A	B	C	D
α	1	2	2	3
β	2	3	3	3
δ	2	4	7	7
ε	2	4	8	8
θ	4	8	12	14
λ	6	11	15	19
π	0	0	0	0
σ	1	5	5	6
ψ	1	5	7	10

Figure 5.3: An example of how to sample a *T-Matrix*. Rows α and λ remain unchanged in both matrices and all the other rows in *Blocks T-Matrix* are calculated differentially with respect to them. For instance, C$\beta = 3 + 2 = 5$.

Namely, current computers store matrices row by row and, as it is very expensive to retrieve information, they fetch not only the required data but also the contiguous information just in case it could be used in the imminent future instead of searching for new information again. Thus, by using sampled rows in our proposal (*Blocks T-Matrix*) we can improve compression while we limit the performance drawbacks of more complex retrieval queries.

Also, the number of samples could be varied to achieve a better performance. The trade-off between performance and space requires different parameters for different contexts so it would need an

experimental evaluation to find them out. Figure 5.3 displays an example on how the original *T-matrix* could be compressed using two sampled rows.

This generic structure was employed in two specific fields of application: public transportation and mobile workforce management. For the first scenario, it was necessary to develop an approach where the load of each bus line could be calculated straightforwardly. The latter was a procedure to fathom into the semantic trajectory analysis. Both approaches will be treated deeply in the following subsections.

5.2 T-Matrices in public transportation

Based on the representation explained in Section 3.2, it is necessary to find a way to solve queries about the network load, asking for the total number of users that boarded or alighted within a stop at a given time/journey. Furthermore, it is also interesting to obtain the average load of a bus or a train between any two stops from a given line.

5.2.1 Data structures

Using two *T-Matrices* for each bus line (board and alight), data can be easily aggregated either by stop (e.g. number of users that got on a vehicle at stop X); by time-interval, hence referred to a sequence of consecutive journeys within that time-interval (e.g. number of users that got on at any stop of the line on 24/08/2020); or by stop and time-interval (a recap example is offered in Figure 5.4). From which the following queries emerge:

- boardX$_{LT}$. Given a stop X, it counts how many passengers boarded a vehicle at X. It is optional to additionally restrict those boardings to any time interval T and line L.

- alightX$_{LT}$. Given a stop X, it counts how many passengers alighted from a vehicle at X. It is optional to restrict those alightings to any time interval T and line L.

- useL$_T$. Given a line L, it counts how many passengers boarded a vehicle of line L. We could additionally restrict those boardings to any given time interval T.
- boardT. Given a time interval T, it counts how many passengers boarded a vehicle (from any line).
- alightT. Given a time interval T, it counts how many passengers alighted from a vehicle (from any line).
- loadX$_{LT}$. Given a stop X from a line L, and being Y the stop after X within line L, we count the number of users that traveled from X to the next stop Y during a time interval T. We can see this query as measuring, for a given vehicle, its average load between two stops.

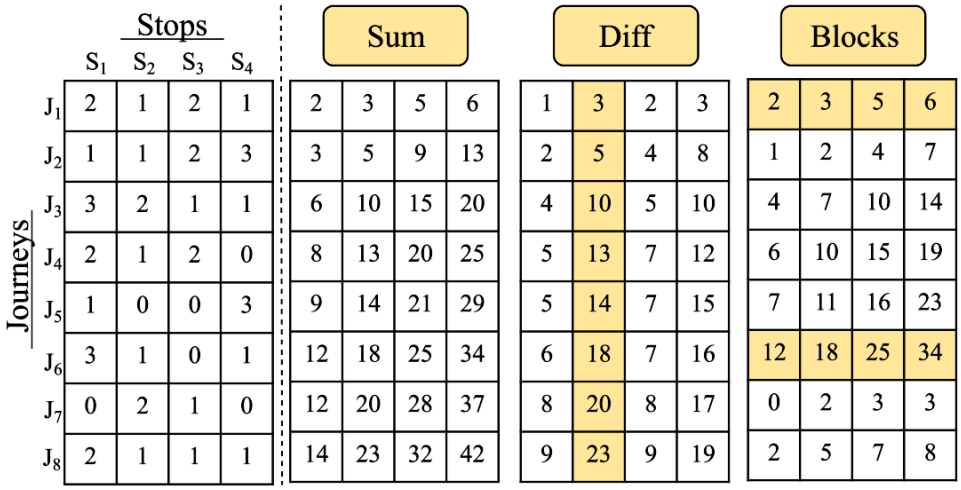


Figure 5.4: Public transport information stored into the three *T-Matrices* flavors: the original accumulative matrix (*Sum*), the relative matrix (*Diff*) and the sampled matrix (*Blocks*).

Thus, storing a board *T-Matrix* (M_L^b) and an alight *T-Matrix* (M_l^a) for each line enables one interesting property: most of the network load queries proposed can be solved as sums of ranges within the matrices M_l^b and M_l^a. For example, we can solve query boardX$_L$ (not constrained

to a temporal interval T) by simply summing the values of the column corresponding to stop X in matrix M_L^b. To solve boardX (neither time nor line constraints are set), we would have to obtain all the lines that contain the given stop X (using the array $stopLine_i$ defined in Section 3.2) and then sum the column associated to stop X along the different M_l^b board matrices corresponding to those lines where X occurs. The previous procedure, yet using the alighting matrices M_L^a rather than M_L^b, permits us to solve alightX_L and alightX type queries.

To deal with queries restricting counts to only those trips within a given time interval $T = [t_a..t_z]$, we have to obtain the journey codes associated to the journeys of a given line l that fell within T. For such sake, we call operation $[j_a..j_z] \leftarrow \texttt{GetJCodes}(l, s, t_a, t_z)$, which internally uses two of our common structures that keep both the average time required to reach any stop from a given line and the starting time corresponding to the journeys of each line. In the next step, since the obtained range $[j_a..j_z]$ makes up a range of contiguous *jcodes* for line l, those journey codes must be associated to a range of contiguous rows within the matrix corresponding to the queried line l. This enables us to both solve boardX_{LT} and alightX_{LT} as the sum of the values in the submatrix $M_L^b[j_a..j_z, s]$ and the submatrix $M_L^a[j_a..j_z, s]$ respectively (being s the column associated to stop X). Analogously, useL_T can be solved by summing the elements on the last column of the matrix corresponding to L (M_L^b), restricted to the range of *jcodes* for the time interval T.

Finally, we show how to combine the alighting and the boarding matrices to solve loadX_{LT}. Observe that the number of users within a bus at a given stop X can be obtained subtracting the number of users that get off the bus from the number of users that boarded the bus either before or at X. Therefore, given a line L, assuming we obtained the range of *jcodes* $[j_a..j_z]$ for T and considering that the range $[s_a..s_z]$ indicates the indices (i.e. columns) corresponding to the stops of line L until X, we solve loadX_{LT} in $O(1)$ time as:

$$loadX_{LT} = \sum_{s=s_a}^{s_z} \sum_{j=j_a}^{j_z} M_L^b[j,s] - M_L^a[j,s] = M_L^b.\mathsf{countRange}((\mathsf{s_a},\mathsf{j_a}),(\mathsf{s_z},\mathsf{j_z})) - M_L^a.\mathsf{countRange}((\mathsf{s_a},\mathsf{j_a}),(\mathsf{s_z},\mathsf{j_z}))$$

5.2.2 Experimental evaluation

We have analyzed the space required to keep all the individual components of our representations in main memory for our experimental dataset as well as the time required by each flavour of our proposal to solve each query. All the experiments in this subsection have been evaluated on a Intel Xeon E5-2620v4@2.1 GHz machine. The code was compiled with GCC 6.3.0 (with `-O3` optimization) and ran on a Debian 6.3.0.

We first explain in Section 5.2.2.1 how a realistic synthetic dataset was built, Section 5.2.2.2 deals with space consumption of our proposal and, finally, Section 5.2.2.3 compares the different speed trade-offs of the divergent *T-Matrices* solutions.

5.2.2.1 Experimental dataset

In order to test the behaviour of *T-matrices* in the scope of public transport, we have created a synthetic dataset of user trips. We used real GTFS[1] data describing schedules and routes for bus networks, we produced a synthetic collection of trips over a month, which imitates actual user trip patterns. We have constructed a combination of two transportation networks: the urban[2] and interurban[3] bus networks from Madrid. After extracting the network model, the following steps were used in order to generate user trips:

1. We extracted the identifiers for routes, including their journeys and their stops. With this we obtained two lines per route in most cases, because we consider the reverse direction of a route as a separate line. This step produced the common structures *lineStop* and *stopLine*.

2. For each stop, we built a list of connected stops, that are either directly reachable from the same line or located at a short walking distance (100 meters).

3. We imported the real schedules for our journeys (bus trips).

[1]Specification available at `https://developers.google.com/transit/gtfs/reference`
[2]Available at EMT `https://www.emtmadrid.es`
[3]Available at CRTM `https://www.crtm.es`

4. Using these schedules, we generated journeys over a virtual month (a 31 days span), cycling the days of the week that the schedules consider. This step produced the structures $avgTime$ and $initialTime$.

5. We generated user trips along those generated journeys. We chose a random stop, day and journey as the start of a trip and simulated boarding and traversing along that journey. After visiting each of its stops, we decide if the user will end the trip at that stop, with a probability starting at 0 and increasing by 0.01 after every visited stop. In addition, we also maintain fixed probability values for attempting to leave the current line in order to switch to another, if there is another line reachable from the current stop with a journey that will allow for a waiting time of at most 30 minutes in order to switch. Naturally, we forbid the possibility of switching to the same line or its reverse.

6. We stored those trips generated in the previous step in the format of consecutive stages. Within each stage[4] the boarding and alighting stop will always be associated with the same journey (and therefore line), i.e. our stages are of the form: $\langle(line_A, journey_A, boarding\_stop), (line_A, journey_A, alighting\_stop)\rangle$. With our set of parameters, around 56% of the generated trips contain only one stage, 33% contain two stages, 9% contain three and 2% contain four. In average, the number of stages per user trip is 1.56.

Following these steps, we prepared a dataset containing 10 million user trips, over a network composed of 11021 stops, and 1048 different lines. The average number of stops for each line is 27.07, and there are on average 1623 journeys per line, with a maximum of 9979 journeys per line.

Note that, given those numbers, the identifiers for our stops, lines, and journeys may be represented, respectively, as integer values of 14, 11, and 14 bits. Therefore, if we represented our dataset as a sequence of tuples of the form (s, l, j), the overall size of our dataset would

[4]We define a user trip or trajectory as a sequence of stages, where each stage indicates in which stops the user boards to and alights from a vehicle corresponding with a journey of a given line.

be of 165.39MiB. The dataset is available at `https://lbd.udc.es/research/XCTR/xctrdata.7z`.

5.2.2.2 Space requirements

In order to ensure the accuracy of the experiments, we have not only included *T-Matrices* in the measurement but also the minor common structures needed to interact with the public transport model (*stopLine*, *lineStop*, etc). The behaviour of these auxiliary structures is described in Section 3.2.

Structures	*stopLine*	*lineStop*	*avgTime*	*initialTime*	**Overall**
Sizes (KiB)	142	120	64	439	**765**

Table 5.1: Space requirements for the common structures.

The space required by our common structures is shown in Table 5.1. We have chosen to use fixed length integers for their representation, except for *initialTime* where we also compressed it using blocks with samples at the beginning of each block, and encoding the remaining values differentialy.

With this compression strategy, the overall space needed to represent the elements representing the transportation network for the purpose of our structures is lower than 1 MiB.

Variant	`Sum`	`Diff`	`Blocks`
Size (MiB)	55.49	43.81	28.68

Table 5.2: Space requirements for each T-Matrices variant.

In order to accurately evaluate the sizes of our proposed versions of *T-Matrices* discussed in Section 5.1, we added up the sizes of both matrices M_l^b and M_l^a for all the lines l in our dataset. We show these total sizes in Table 5.2, where we can conclude that *Blocks* is the most effective approach to save space for the accumulated values in our matrices, decreasing the memory fingerprint to around 50% of the uncompressed *Sum* counterpart. We can also see that *Diff* is also able to reduce the total space by around 20%. Yet, it is constrained by the shape of our matrices, making it less desirable in practice.

5.2.2.3 Performance at query time

In Figure 5.5, we compare the different variants of *T-Matrices*, we can observe that queries with a line restriction (boardX$_L$ and boardX$_{LT}$) are significantly faster than those with unrestricted line (boardX and boardX$_T$), since for the two latter queries a different matrix must be accessed for every line where the stop occurs in, as indicated above.

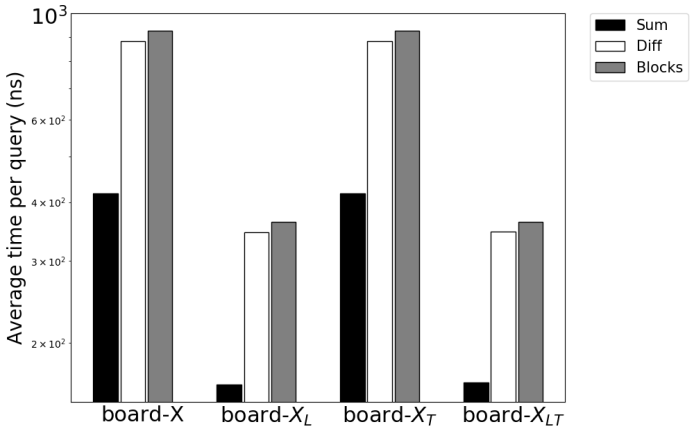


Figure 5.5: T-Matrices comparison. Logarithmic scale measured in nanoseconds.

We can see that the compression strategy used to decrease the size of the original *Sum* that query times typically worsen by around 50%. Finally, when comparing the two compression alternatives proposed, we can see that *Blocks* is slightly slower than *Diff*. Yet, its much better compression effectiveness clearly make *Blocks* the better compressed approach.

5.3 T-Matrices in mobile workforce management

Having in mind all the ideas discussed in Section 3.3, this work aims to merge all this knowledge in order to open a new path towards a trajectory data warehouse, that is, to build a structure capable of storing geospatial data in the same way as a classic data warehouse deals with aggregated information. This is specially hard in the field of geographic knowledge due to the complexity of obtaining accumulative data. We propose a new structure to deal with the aggregated information of the different activities carried out by moving workers along their trajectories.

5.3.1 Data structures

Back to the trajectory concept, we know now that any mobile object travels around particular points at a particular time and we can rebuild its trajectory just linking the points chronologically. At this time, we can also add some meaning to this trajectory storing what task was the object doing at each point, obtaining a semantic trajectory.

The main goal of this proposal is to develop a compressed representation of a collection of semantic trajectories in such way that it is still possible to answer different relevant queries efficiently. This subject has a special interest for companies having mobile workers (e.g. truck drivers, delivery workers, etc.) since time management of their employees is crucial and it is significant in this manuscript because it is the starting point towards building a trajectory data warehouse.

Under the usual tasks compilation of a truck driver introduced in Section 3.3, it is possible to assemble a semantic trajectory for each truck driver concatenating chronologically chore after chore. A straightforward approach for storing this information would be a matrix where moving objects take the part of rows and columns represent the time dimension. This time dimension is a discretization of the time in such a way that each column corresponds to a time interval related to the actual continuous time between two discretized time instants (e.g. 08:20–08:30). Thus, a cell within this matrix contains the identifier of the (most-representative) activity performed by a given mobile object at a particular time interval. For instance, according to the matrix in Figure 5.6, the tractor was performing the activity with id 7 from 08:10h to 08:20h.

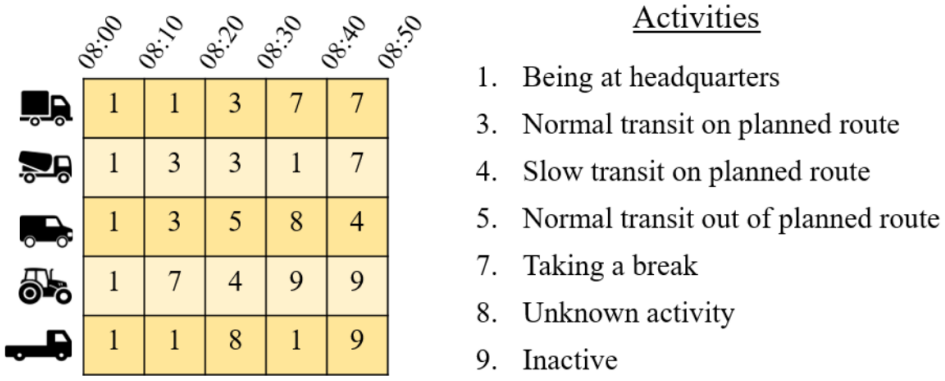


Figure 5.6: Naive matrix representation with appearing activities described.

With the aim of improving this naive solution, we have developed a brand new structure called *semantrix*. As it can be inferred from the previous example, the aim in *semantrix* is not only to work with trajectories but also with their aggregated data. Our proposal is able to store all the information included in the previous *naive* matrix, and remarkably decreases aggregated queries times and pattern matching. Three vectors are involved in this new structure: a bitvector B, an integer vector H and a vector of *T-matrices* S. The former two vectors permit us to compactly represent the original sequence of activities within the *naive* matrix to tackle non-aggregated knowledge (e.g. what was driver K doing at 13:20?). The later vector keeps one activity *T-matrix* for each possible activity so that, for each activity, it handles accumulative information for each vehicle and time-interval. Those structures, that are discussed below, are depicted in Figure 5.7.

- Representing the *naive* matrix: **Bitvector B and vector H.** Recall the information that regards the activity performed by each mobile object during each discretized time interval was stored in the *naive* matrix previously. In addition, a given row i ($1 \leq i \leq r$) keeps particularly the activities for the i-th mobile object during each of the I time intervals. Those r rows can be concatenated to make up a unique sequence of rows ($OS[1, rI]$) as depicted in the top of Figure 5.7. Note that since all those r rows have the same length I (number of time intervals) we retain the same direct-access

capabilities as in the *naive* matrix. Yet, we also have the same space needs. To compactly represent OS we use:

i) A **bitvector** $B[1, rI]$ aligned with OS where we set a 1 each time an activity switch occurs in OS; i.e. we set $B[1] = 0$ and then, $\forall i \in \{2..rI\}$ we set $B[i] = 1$ if $OS[i] \neq OS[i-1]$; we set $B[i] = 0$ otherwise. Finally, we also set a 1 at positions $B[1+kI]$ $\forall k = \{1..r\}$ to mark a row/mobile-object switch.

ii) An **integer vector** $H[1, o]$, such that $o = rank_1(B, rI)$ is aligned with the o ones in B, and stores the *ids* of the activities from OS associated to those ones in B. Therefore, $\forall i \in \{1, o\}$ we set $H[i] = OS[select_1(B, i)]$. Note that H contains, for each mobile object, a sequence with the identifiers of the activities it performed.

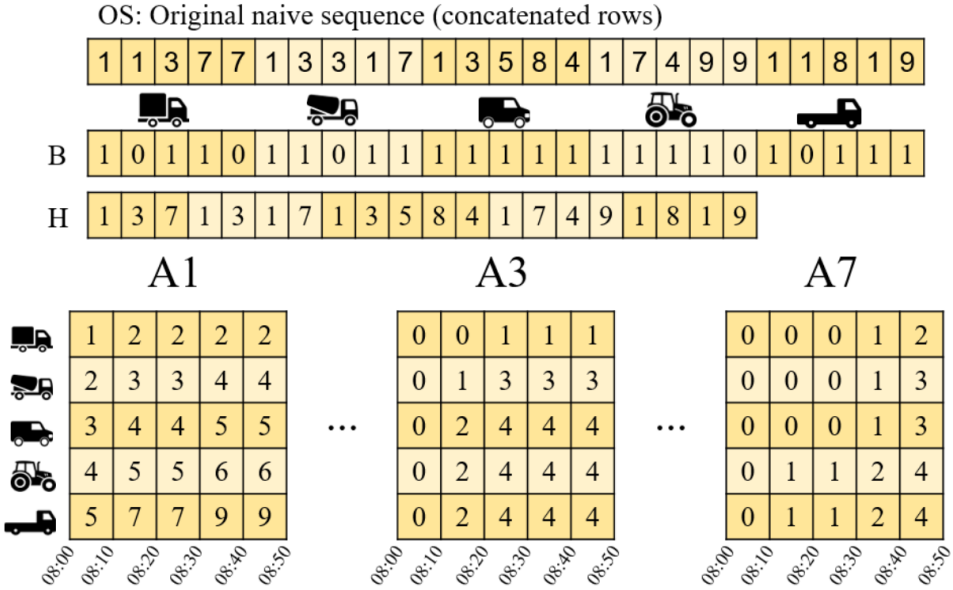


Figure 5.7: Structures involved in *semantrix*.

- Storing aggregated information related to each activity: **Vector of Activity matrices** S. We have included a vector of *T-matrices* (one per activity/event) that captures two-dimensional information: time and driver. In Figure 5.7, it is shown how the accumulative activity matrices S_1, S_3, and S_7 for the activities 1, 3 and 7 in our working example would look like (note that we are not showing the content of the other S_i matrices). For example we can see that $S1[4, 0] = 5$ (*T-matrix* for the first activity) indicates that the first four workers/vehicles performed task S_1 a total of five times, during the interval $[13{:}00, 13{:}10)$.

In our scope, we can distinguish among three main types of queries. We found *individual queries* that aim at gathering the content of one particular cell from the original *naive* matrix (e.g. "Which was the activity performed by a given mobile object O_j at a given time instant I_i?", or "Which is the list of activities performed by a given mobile object O_j during a given time interval $[I_s..I_e]$?"). There are also queries focusing on detecting if a given pattern of activities occurred (e.g. "How many times the activity A_i was followed by the activity A_j?"). Finally, we also have to deal with aggregated queries aimed to unravel the total values hidden within the matrix (e.g. "How much time was actually spent by all the mobile objects while performing the activity A_i during a given time window $[I_s, I_e]$?"). To support this types of operations we used the different structures within *semantrix*.

- **Individual queries:** These kind of queries are easily solved just using the bitvector B and vector H. First, with a *rank* operation over the bitmap we obtain the position(s) of interest; and then this position is used to access H to retrieve the activity/ies within the particular time window.

- **Pattern queries:** For these queries, a *FM-index* built on top of H vector is used (see Section 2.3.1). Therefore, we use its self-indexing capabilities to efficiently locate patterns of activities. Particularly, to solve a query of the form "How many times was activity A_i followed by A_j?" we simply rely on $count(A_iA_j)$ over the *FM-index* of H.

- **Aggregated queries:** Once again, most aggregated queries can be solved in constant time with the help of *T-matrices* and the *countRange* operation. Data can be aggregated either by vehicle, querying a row will obtain the number of times a vehicle have executed a task (e.g. "How much time was spend by vehicle F at headquarters?"); by time-interval, summing the values of a column calculates the number of times an activity was performed during a particular time interval (e.g. "How many vehicles were trapped on a traffic jam today at 8:30?"); or a combination of both (e.g. "How much time was spend by vehicles F and G taking a break yesterday?").

5.3.2 Experimental evaluation

It is worth recalling that the seminal idea for this work arose as a recent project shared with a local company devoted to the transportation of organic waste. Accordingly, the actual experimental evaluation is now taking place on a real environment. Our system is being used on a daily basis to manage the activities of the trucks from the enterprise.

We present experiments comparing our proposal *semantrix* with other representations and show both the space needs and their performance at query time for different types of queries. Below, we discuss the baseline representations used and we show the corresponding experimental results.

We have included in our experiments the *naive* original matrix discussed at the beginning of this section. Additionally, we have implemented a more elaborated baseline named *baseline+* (see Figure 5.8). It is based on the sequence of all the activities performed ordered both chronologically, and by moving object. It consists basically in the OS vector (i.e. sequence composed of the rows from the original matrix). Yet, we have also included a set of aggregated sequences to boost solving aggregated operations. There is one sequence per activity that gathers all the accumulative data in chronological order. Thereby, *individual* and *pattern* queries are solved with the activity sequence, while the accumulative sequences deal with the data warehouse-like queries.

Finally, we have also included in the comparison the *T-Matrix* variant

referred to as *Blocks T-Matrix* at the beginning of this section. On the contrary, we have not included the *Diff T-Matrix* approach as it was demonstrated to be less effective in Section 5.2.2.

OS: sequence of activities

1	1	3	7	7	1	3	3	1	7	1	3	5	8	4	1	7	4	9	9	1	1	8	1	9

Accumulative sequences

A1	1	2	2	2	2	3	3	3	4	4	5	5	5	5	5	6	6	6	6	6	7	8	8	9	10
A2	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
A3	0	0	1	1	1	1	2	3	3	3	3	4	4	4	4	4	4	4	4	4	4	4	4	4	4
⋮												⋮													
A9	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	2	2	2	2	2	3

Figure 5.8: *Baseline+* example.

We have measured average execution times from 10,000 randomly generated queries on an Intel(R) Core(TM) i7-3820 CPU @ 3.60GHz machine running Debian GNU/Linux 9.9. Our implementations use components from the SDSL Library[5]. The compiler used was g++ 6.3.0 and optimization flags were set to -O3.

This chapter continues as follows: Section 5.3.2.1 deals with the creation of our experimental datasets, Section 5.2.2.2 compares our proposal against the baseline just described and the basic naive matrix and Section 5.2.2.3 does the same for time measurements.

5.3.2.1 Experimental datasets

Since our system has not been used over a relevant amount of time (6 months or more) yet, there are not enough real data to test our proposal in a real environment. Nonetheless, we have generated a synthetic dataset according to the actual constraints and the current existing statistics, where we have recreated realistic information about daily truck activities in a collaborating company. We have discretized the time using 5-min intervals, which is a sensible time lapse considering the speed of the trucks. We assume a small company that has 20 trucks whose drivers work 8 hours every day of the week. Assuming those preconditions and the nine activities discussed above, three datasets with

[5]https://github.com/simongog/sdsl-lite

different temporal sizes were created: one month ($12 \times 8 \times 7 \times 4 = 2688$ time instants), six months ($12 \times 8 \times 7 \times 4 \times 6 = 16128$ instants) and one year ($12 \times 8 \times 7 \times 4 \times 12 = 32256$ instants).

5.3.2.2 Space requirements

We have compared the space requirements of the tested techniques. As shown in Figure 5.9, the original matrix (*naive)* needs, by far, less space as it only stores the activity values within the original matrix.

The others use roughly the same space. Yet, it is worth noting that *Block T-Matrices* (sampling every 4 rows) require around 15% less space than *semantrix* (i.e. Classic T-Matrices). *Baseline+* uses around 8% less space than *semantrix*.

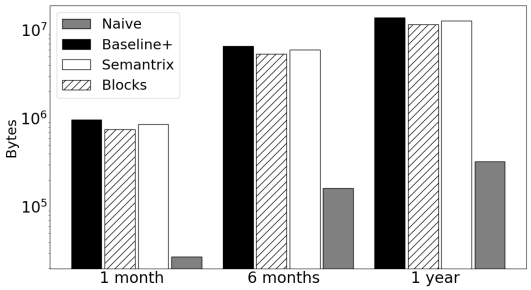


Figure 5.9: Space measurements.

5.3.2.3 Performance at query time

To test query performance, we have chosen one query of each type (we have skipped the results from single-query type as the results showed only negligible differences among all the techniques). For pattern-queries we used the query "How many times was activity X followed by activity Y?", and for aggregated-queries, we used the query "How many times were trucks 1, 2 and 3 performing activity X from 11am to 12pm (12 time Intervals)?".

We can see in Figure 5.10 that the techniques using pre-calculated aggregation (*Semantrix*, *Blocks* and *Baseline+*) are much faster solving both pattern and aggregation queries than *naive*. Actually, the *naive*

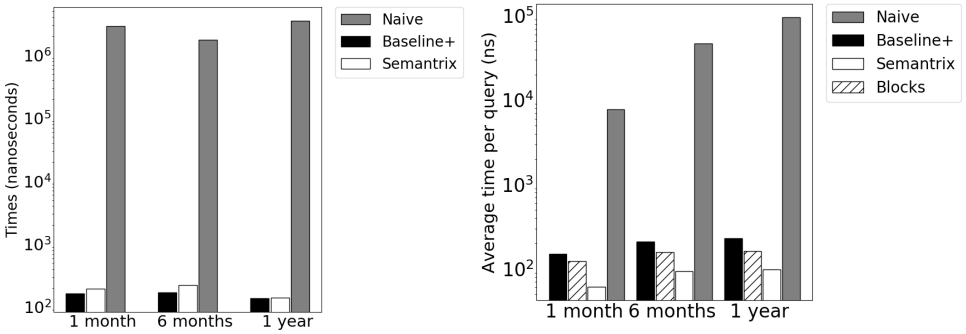


Figure 5.10: Times for pattern queries (left), and times for aggregation queries (right).

approach, which must traverse the original matrix, becomes several orders of magnitude slower than both *semantrix* and *baseline+* when solving pattern queries (Figure 5.10 (left)). However, these latter structures obtain similar results as they both rely on an *FM-index* to solve pattern queries (with the difference that *semantrix* needs an additional access to B). Note that such additional access to B is compensated by the fact that *semantrix* recovers/processes less data (H is smaller than OS vector) from its *FM-index* than *baseline+*.

For aggregated queries, Figure 5.10 (right) shows, as expected, that *semantrix* is clearly the fastest technique. As in the previous experiment, *naive* needs to explore the whole queried submatrix, whereas *semantrix* counterparts and *baseline+* benefit from their aggregated data. In this case, *blocks* is around 40% slower than *semantrix*.

5.4 Conclusions

We have introduced a general-purpose method to handle aggregated queries efficiently. Our proposal is able to store accumulative multidimensional data and it is capable of obtaining the sum of any subrange in constant time. It follows the same idea of classical datawarehouses where aggregated information and individual data are stored in different structures so, as part of our contribution, we have presented different *T-Matrices* flavours to reduce the space usage and stimulate compression.

In order to demonstrate the versatility of our proposal, we have tested it in two real-world scenarios: public transport and mobile workforce management.

In the former, we studied the real demand on passengers of a public bus system (whose requirements would be very similar to other transportation systems such as trains or subways). Our representation enables to handle adequately the aggregation of individual passenger trips allowing to compute load queries, stop statistics or time-window queries efficiently.

In the latter, we have analyzed the problem of representing and managing semantic trajectories in a compact and efficient way. We present a data structure named *semantrix* to handle a semantic data warehouse for the trajectories from mobile objects and we show how it supports different types of queries. The proposal works on top of the compressed activity sequence (ordered chronologically and by mobile-object identifier) which leans on a bitmap for individual and pattern-matching queries. The structure is crowned by a *T-matrix* for each activity enabling it to solve most accumulated queries in constant time. We have experimentally evaluated the proposed solution using realistic synthetic data that represent the truck movements of a real company. As a quality proof, it is worth recalling that our system is being used as part of a real company project, solving a real life problem.

Chapter 6

Event sequence indexing

This chapter explains a new technique for the efficient management of large sequences of multi-dimensional data, which takes advantage of regularities that arise in real-world datasets and supports different types of aggregation queries. More importantly, our representation is flexible in the sense that the relevant dimensions and queries may be used to guide the construction process, easily providing a space-time trade-off depending on the relevant queries in the domain. We provide two alternative representations for sequences of multidimensional data and describe the techniques to efficiently store the datasets and to perform aggregation queries over the compressed representation. We perform experimental evaluation on realistic datasets, showing the space efficiency and query capabilities of our proposal.

Section 6.1 opens this chapter exploring the motivation of this contribution, Section 6.2 details how our proposal works, Section 6.3 introduces some space footprint improvements on our structure, Section 6.4 analyzes the behaviour of our proposals and Section 6.5 contains a summary with the key ideas of this chapter.

6.1 Introduction

One of the main weaknesses of the classic accumulative solutions is that, while we are able to compute aggregations in a short time, the order of the events gets lost. For example, in a geospatial dataset, we

may represent the elevations of roads as a sequence of events. A classic accumulative solution could answer queries about how many kilometres a road X traverses at elevation 1000m, 1050m or 1010m, but we could also be interested in knowing if those kilometres at elevation 1000m are contiguous (the road has no slopes) or if they are short sectors because the road has many changes of elevation. In these cases where the order of the values of some variable is relevant, typical data warehouses are not capable of answering all the relevant queries because they cannot exploit the sequence of events (or values of the variable of interest: elevation in our example).

Considering the application context of Section 5.3, a system storing the activity log for a set of employees contains periodic entries, each of them including time, employee, and activity performed, among other data. From this dataset, we may be interested in obtaining the amount of time devoted to a specific activity on a given day or the amount of time that an employee has worked during a period of time. While this information can be computed from the original data, in most cases the information is processed and stored in separate representations specifically designed to answer aggregation queries. Once information is aggregated, the original data sequences are usually discarded or stored separately. Without going any further, observe that this is the reason why *T-matrices* need auxiliary structures to preserve the non-aggregated data to solve non-cumulative queries (see Section 5.3).

Data stored in databases or obtained from different tools usually have some kind of ordering (usually by time, but other sorting criteria may become relevant). This order may also lead to relevant queries. Following our previous example with employees and activities, we may be interested in retrieving the sequence of activities performed by an employee on a day, in addition to computing aggregated values. In a dataset with geographic information, such as taxi driver historical records, we may be interested not only in retrieving the position of a vehicle at a certain time but also in rebuilding an entire trajectory. In this case we need to store the original chronological order of the data as it would not be possible to retrieve any spatial trajectory without it. In addition, we can aim even further and try to calculate the number of times a vehicle drove through a certain area or even how many times a

taxi traversed a particular street segment. Therefore, we need a data structure that is not only capable of answering aggregation queries efficiently but also preserves the order of the original sequence to tackle all relevant queries in one single structure.

6.2 Indexing with Wavelet Trees

Consider the sequence S of events where each element is a multidimensional entry, from a given set of dimensions, of the form $(d_1, d_2, \ldots, d_k) \in D_1 \times D_2 \times \cdots \times D_k$. The queries of interest in this domain involve aggregations of values for specific values, or ranges of values, in one or more of the dimensions of the dataset. That is, we are interested in counting the number of entries in the sequence for a given value of D_i or for a given set of constraints across multiple dimensions, such as $d_1 = x$, $d_2 \in [d_{2\ell}, d_{2r}]$, etc.

A plain representation of the original sequence is quite inefficient to retrieve any kind of aggregate information. In order to answer these queries, additional data structures are used to keep track of the specific values, thus, storing only accumulated data. Hence, both the representation of the sequence and the additional data structures are required to keep the ability to access and decode the sequence of operations.

Our proposal is based on using *wavelet trees* (explained in detail in Section 2.3.2) to represent the sequence and provide efficient aggregation on a subset of the dimensions. The *wavelet tree* representation of a sequence provides a reordering of the elements in the original sequence according to the "alphabet order". In other words, when a *wavelet tree* of the sequence is built on an alphabet Σ, the elements of the sequence are stably-sorted by their position in this alphabet, so that the sequence can be accessed in the original order, using the root of the *wavelet tree*, or in alphabet order, on the leaves.

Consider a sequence of tuples $(d_1, \ldots, d_k)$ sorted by D_1, then D_2, and so on. If we build a *wavelet tree*, S_{D_i}, on this sequence according to D_i, we are in practice stably-sorting all the elements of the sequence by D_i. Hence, this *wavelet tree* provides the support to efficiently restrict our queries to specific values of D_i, thus effectively providing aggregation

capabilities in dimension D_i. For example, in order to count the number of entries for a given $c \in d_i$, we just need to compute $rank_c(S_{D_i}, n)$.

In order to provide aggregation capabilities in another dimension, we can easily repeat the process, creating a *wavelet tree* over the re-ordered sequence using a different dimension. If we build a new *wavelet tree*, S_{D_j}, using the values of D_j, we are in practice grouping elements according to that dimension, while keeping the previous ordering by D_i. Using rank operations, we can now easily compute $rank_{c'}(S_{D_j}, n)$ to count the occurrences of any $c' \in D_j$. In addition, we can easily restrict the previous query to the range covered by any c in D_i, thus we can also answer queries involving any pair of values in both dimensions. An example of our *stacked wavelet trees* is depicted in Figure 6.1.

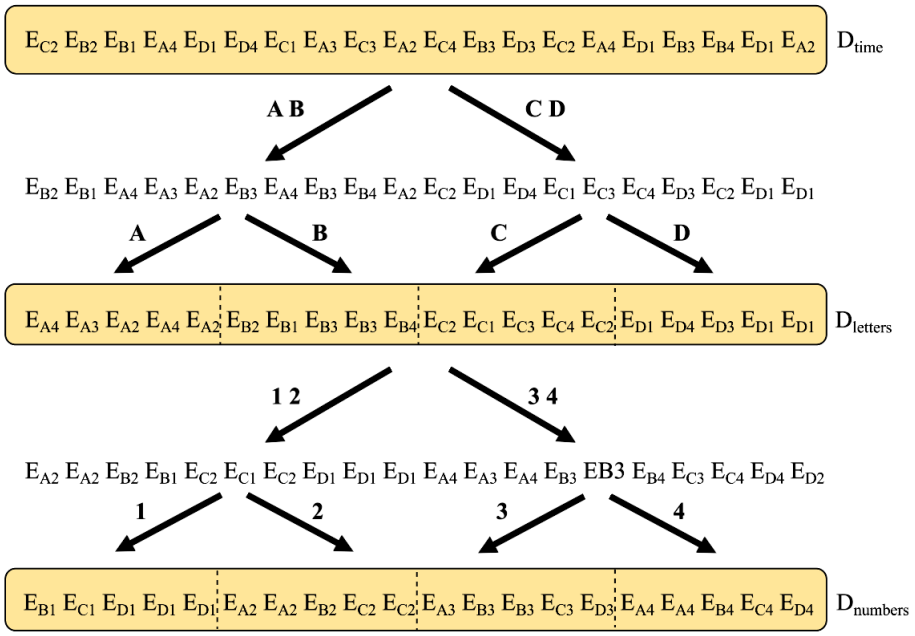


Figure 6.1: Reordering multidimensional events through *stacked wavelet trees*. Highlighted sequences are the resulting sequence after a dimensional reorganization. Vertical dotted lines mark the corresponding area to each value of the dimension (remaining within it the previous order as secondary).

Notice that, after reordering by a second dimension, the sequence

is still sorted by the previous one in each leaf (that is, they are sorted by D_j, and by D_i inside each group of D_j), we can also efficiently look for the elements in the sequence that have a specific value for D_j and a range of values of D_i: we just need to restrict the search in the second *wavelet tree* to the range of values for the other dimension, which can be easily computed. Following this idea, table 6.1 shows a simple example in which the data are originally sorted by day, then reordered by activity and finally by employee.

Our representation can efficiently compute queries involving consecutive elements in any of the reorderings. For example, count the number of entries involving activity E, which are 4, can be answered quickly in the first reordering. In the same reordering, the query can be also restricted to a range of days (e.g. entries involving activity E during days [1..2], which is 3). As for the second reordering, we may count the number of entries in which employee 2 was performing activity E during days [1..2], which is 2.

Original order			**Reordering by Act**			**Reordering by Emp**		
Day	**Emp**	**Act**	**Day**	**Emp**	**Act**	**Day**	**Emp**	**Act**
1	1	C	1	1	A	1	1	A
1	1	E	2	1	A	2	1	A
1	1	A	1	2	B	1	1	C
1	2	E	1	1	C	2	1	C
1	2	B	2	1	C	1	1	E
2	1	A	1	1	E	1	2	B
2	1	C	1	2	E	1	2	E
2	2	E	2	2	E	2	2	E
3	2	E	3	2	E	3	2	E

Table 6.1: Example of reorderings with dimensions Day, Employee and Activity

Interesting enough, the different *wavelet trees* need not be just stacked, but can be combined in different ways according to the relevant queries in the domain. For instance, if we have a 4-dimensional sequence of events that are characterized by dimensions $A \times B \times C \times D$, we may first process the sequence of events attending their value in D, sorting them by that dimension. The result can then be used in two different *wavelet trees*, one that uses the values in dimension C as the vocabulary and another that uses the values in dimension B.

In this way, we shall be able to easily count occurrences for a given value d, a given pair $(c, d\star)$ or a given pair $(b, d\star)$, where $x\star$ denotes the ability to search for a specific value or a range of values. An alternative representation that builds the 3 *wavelet trees* in sequence, would be able to answer aggregation queries for given d, $(c, d\star)$ or $(b, c, d\star)$. In this way, the final representation can include as many combinations as needed to provide the necessary query capabilities, at a cost of extra space. Given a set of queries, the problem of computing the minimum number of necessary *wavelet trees* is an extension of the shortest common superstring problem [RU81], i.e. obtaining the shortest string that contains a set of given strings.

6.3 Reducing the space

In many real-world scenarios, sequences of events are highly repetitive. This effect, called *locality*, is increased with the granularity of the sequence. In the context described in Section 5.3, if activities are recorded every minute, the sequence will have more locality than if they are recorded every hour (i.e. it is more likely that an employee keeps performing the same task in the next minutes than in the next hours).

We introduce now an improvement over the *wavelet tree* composition proposed that takes advantage of this property, providing a solution that is insensitive to the granularity of the sequences. This is interesting because it makes them suitable to represent sequences with a high level of detail, i.e. resolution, in little space and supporting aggregation queries.

We assume that the original sequence is sorted in such a way that aforementioned locality exists (e.g. time or space depending on the domain). Then, locality produces runs of symbols in the sequence. An out-of-the-box solution to exploit the existence of these runs is to use run-length compressed *wavelet trees* [MN05]. We refer to this solution as *wtrle*. Runs on the sequence produce runs on the bitmaps of the *wavelet tree*, which can be represented in few space and still support rank/select operations [DRR06]. Overall, these *wavelet trees* require space proportional to the number of runs in the sequence, instead of to the length of the sequence itself. Counting and access queries can

be implemented as rank and access operations on the *wavelet tree*, respectively.

When queries involve more dimensions, an additional component is necessary. In the *wavelet tree* composition explained above, a query on a *wavelet tree* is refined on a second *wavelet tree*, and so on. To do that, we need to store for each leaf of the *wavelet tree* the number of elements that are lower, in the order defined by such *wavelet tree*, than its corresponding symbol. This component is used to restrict the query in the subsequent *wavelet tree* and it can be implemented as a plain array or as a sparse bitmap, depending on the size of the dimension.

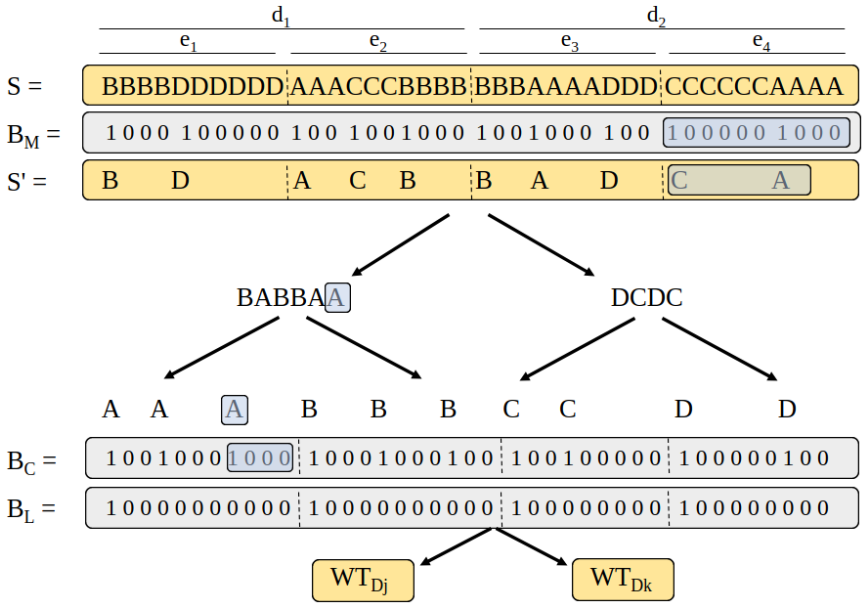


Figure 6.2: Schema of the *wtmap* solution for a sequence of 2 days, 2 employees, 4 activities and 10 time instants per day. The current WT orders the input attending activity lexicographical order. Highlighted elements are those visited to count the time-instants devoted by e_2 to activity A during d_2, which are 4.

This simple approach represents the runs at each level of the *wavelet tree*, which is somehow a waste of space. Thus, our second approach, *wtmap*, uses a technique to remove the runs from the original sequence.

We store a bitmap B_M, of length n, that marks the start of each run. Then, a *wavelet tree* is built over a sequence S' of length $n' < n$, in which each run of value v is represented as a single value v. As this sequence does not contain runs, the *wavelet tree* is built using plain bitmaps, which have the additional benefit of being faster for querying. A query on the original sequence can be easily mapped to a query on this new *wavelet tree* using rank operations on B_M. However, this *wavelet tree* does not contain information about repetitions of symbols, which is necessary to support counting operations. Hence, a second bitmap B_C is built, storing in unary the length of each run. This bitmap considers the runs in the reordering performed by the *wavelet tree*, i.e. it is aligned with the leaves of the *wavelet tree*. Rank/select support is also necessary on this bitmap to solve counting queries. The two bitmaps, B_M and B_C, are sparse, which is directly related with the existence of runs on the original sequence, and are implemented using Elias-Fano representation [OS07]. The same final component described for the *wtrle* is necessary here to do the composition between *wavelet trees*. This component is shown as bitmap B_L in Figure 6.2, which illustrates this approach.

6.4 Experimental evaluation

We designed a set of experiments focused on a realistic use case, involving the storage and access to employee tracking information. We assume that we have a set of entries storing the following information: day, employee, time instant and activity. We want to keep track of all the information of our employees, hence being able to recover the specific activity that was being performed at a given time instant by a specific employee. In addition, typical queries in this domain involve checking the amount of time devoted to a specific activity by an employee or by all employees. Hence we need to store the original sequence but also to efficiently answer aggregate queries. We shall consider for instance aggregations of time per activity/employee/day, and per activity/day.

6.4.1 Problem setup

In this kind of data, time is a natural ordering of entries, and also provides compressibility due to locality: consecutive entries for the same employee and day should usually return the same activity, especially if the measurement frequency (resolution) is high. In this sense, storing the sequence of activities sorted by employee and time should lead to long runs of similar activities. We will consider the sequence of activities as sorted first by day, and then by employee and time, since days are a natural way of grouping data, the locality of activities is not affected and it becomes easier to answer queries for a specific day. Even though not all employees work every day, we can ignore this problem by storing every possible combination of day, employee and time instant and using a special activity 0 to denote the fact that the employee was not working at that point. This increases the length of the sequence, but it makes computations simpler and our representations will compress these long runs of values in little space.

Again, the queries that we will consider in our experiments involve aggregation operations and retrieval of specific entries in the sequence. We name our counting queries C-xD-yE-zA, where D, E and A are days, employees and activities, and x, y, z can denote a single value (1), a range ("r") or all possible values ("a"). Hence, C-1D-1E-1A counts the time devoted on 1 day, by 1 employee, to a specific activity. A more complex aggregation is C-1D-aE-1A (1 day, all employees, 1 activity), that can be trivially extended to a range of days. We also test ranges of days with query C-rD-1E-1A. In addition to these aggregation queries, we also test the ability of our proposal to *access* specific positions of the original sequence: query *Acc* retrieves the activity being performed, given an employee, day and time instant.

Taking into account the characteristics of the dataset, we build our representation sorting entries by activity first, in a first *wavelet tree* decomposition; then we group again by employee. The first decomposition allows efficient counting operations by activity and day (or range of days); the second one works for queries on activity-employee-day. Notice also that since we actually store every possible combination of values, an *Acc* query can be easily translated into a position to extract in the sequence.

6.4.2 Baseline representation

As a baseline for comparison we build a simpler representation that uses the same ordering of the information and takes advantage of it to answer the same relevant queries with additional data structures. The baseline consists of two separate components: a representation of the original sequence, which can be used to access the original data, and an additional data structure for aggregation queries. This baseline thus follows the usual approach of separating the original data and precomputing aggregated values in a separate structure.

The sequence of tuples is stored using two arrays and two additional bitmaps. Initially, the sequence is "compressed" removing repeated entries: each group of consecutive entries with the same values is replaced with a single entry storing the activity and the amount of time devoted to it. The compact representation works with this shorter sequence. A bitmap B_D marks the positions in the sequence where a change of day occurs; a second bitmap B_E is used to locate positions where each employee sequence begins. Using these two bitmaps, we only need to store activities and times for each position: two arrays store the activities and time values for each entry. With this representation the original contents can be recovered using the bitmaps to locate the appropriate run and processing the sequence of times to locate the desired position. Operations that require counting (time spent in an activity by an employee or by all employees in a day) can also be computed using this sequence, but may require the traversal of several different regions of the array to compute the overall result.

To provide efficient counting operations, the baseline also stores a data structure specifically designed to compute aggregated values, a promising bet towards compact data at development time. This structure is called *CMHD* (see Section 2.4.4) and it is a compact representation designed for aggregation queries in hierarchical domains such as OLAP systems. This representation essentially decomposes the n-dimensional datasets, according to predefined hierarchies and is able to store accumulated values at each level of decomposition (for instance, given products and places, it can store in the same representation accumulated values by category and country, and also by individual products and places). In our domain no natural hierarchies appear apart from days and time.

Nevertheless, we can build fictitious hierarchy levels to efficiently answer queries: decomposing first some dimensions we can obtain a first level in the *CMHD* that stores cumulative values for all employees, 1 activity and 1 day; then, decomposition can continue in a second level to store cumulative values for 1 employee, 1 activity, 1 day. In this way, queries that involve a single value or all values can be answered efficiently, as long as a hierarchy is appropriately built in the *CMHD*.

6.4.3 Experiments and results

We built several synthetic datasets to test non-biased data with different granularities following the expected distribution for the domain. All our datasets have similar realistic characteristics but different size in different dimensions, to measure the evolution of the proposals. The changing parameters in the datasets are the number of activities A, the number of employees E and the number of time instants per day, or resolution, R. The number of days is set to a fixed value (500), since it has little effect on query times and only changes the length of the sequence. Datasets *T-rrr* (where *rrr* is the resolution) are built with $E = 50$, $A = 16$ and $R \in \{720, 1120, 5760, 11520\}$. Datasets *Dat-ee-aa* are built with $R = 5760$, $E \in \{20, 50, 100\}$ and $A \in \{16, 32, 64, 256\}$. In all the datasets, we consider that employees work on shifts (50% of the time each working day), and have free days (only 80% of the employees work each day). For each dataset and query type we generated sets of 1 million queries, selecting random values and intervals.

In this section we compare our two proposals, *wtrle* and *wtmap*, with the alternative *baseline* representation, for all the test datasets. All the representations are implemented in C++ and compiled with g++ with full optimization enabled. All the experiments in this section were performed in an Intel Core i7-3770@3.60GHz and 16GB of RAM, running Ubuntu 16.04.4.

Table 6.2 shows a comparison between the different spatial efficiencies of each approach. For each dataset, the input size is computed considering a completely plain representation of all the tuples in the sequence, with each value stored using 32 bits, so this size increases linearly with the number of entries. All the compressed representations are much smaller than the original sequence, thanks to the efficient

compression of the runs of repeated values. Both of our representations are in all cases considerably smaller (5 to 10 times smaller) than the baseline. The most efficient representation is *wtmap*, since it removes runs of repeated values in a single step, while *wtrle* has a small overhead to compress the runs in every level of the *wavelet tree*.

Dataset	Input size	*baseline*	*wtrle*	*wtmap*
T-720	274.66	30.92	3.09	**2.98**
T-2880	1,098.63	30.73	3.58	**3.33**
T-5760	2.197.27	30.89	3.84	**3.53**
T-11520	4,394.53	30.85	4.09	**3.70**
Dat-20-16	878.91	12.41	1.50	**1.35**
Dat-20-32	878.91	12.79	1.78	**1.43**
Dat-20-64	878.91	13.20	2.04	**1.51**
Dat-20-256	878.91	13.74	2.35	**1.65**
Dat-50-16	2,197.27	30.65	3.82	**3.51**
Dat-50-32	2.197.27	32.11	4.63	**3.76**
Dat-50-64	2,197.27	33.14	5.35	**3.99**
Dat-50-256	2,197.27	34.43	6.20	**4.35**
Dat-100-16	4,394.53	61.76	7.72	**7.27**
Dat-100-32	4.394.53	64.02	9.35	**7.76**
Dat-100-64	4,394.53	65.78	10.78	**8.15**
Dat-100-256	4,394.53	67.96	12.45	**8.84**

Table 6.2: Space required by all the datasets (sizes in MB)

Next we compare our proposals with the baseline for the different queries described for this domain. Table 6.3 reflects the diverse query times between approaches. The first query we study is the *access* query *Acc*, that recovers random positions in the sequence. In this query both of our proposals are faster than the baseline in all cases and again *wtmap* is the fastest of our alternatives.

Next, also in Table 6.3, we study aggregation queries. For C-1D-1E-1A we show two different times for the baseline: the first time is obtained querying the *CMHD* representation, and the time in parentheses is obtained traversing the sequence representation. In this query, the naive approach that traverses the sequence obtains in practice the best results. This is due to the relatively small number of activities per day in this domain. Note however that the cost of the naive sequence traversal is linear on the number of entries to search, so it may be slower than any of the indexed proposals even for simple queries if the number of entries to search is relatively large. In any case, our representations are still

Dataset	Access			C-1D-1E-1A			C-1D-aE-1A			C-rD-1E-1A		
	baseline	*wtrle*	*wtmap*	*baseline*	*wtrle*	*wtmap*	*baseline*	*wtrle*	*wtmap*	*baseline*	*wtrle*	*wtmap*
T-720	0.54	0.38	**0.31**	1.57 (**0.49***)	0.91	0.68	**1.46**	2.33	1.69	—	5.66	**3.06**
T-2880	0.54	**0.29**	**0.29**	1.57 (**0.48***)	1.15	0.94	**1.40**	2.48	1.78	—	6.71	**3.45**
T-5760	0.54	**0.29**	0.40	1.55 (**0.48***)	1.18	0.92	**1.44**	2.35	1.80	—	7.86	**4.24**
T-11520	0.55	**0.34**	0.43	1.54 (**0.48***)	1.50	1.08	**1.42**	3.15	1.96	—	8.58	**4.29**
Dat-20-16	0.51	0.36	**0.23**	1.46 (**0.45***)	1.12	0.74	**1.35**	2.21	1.67	—	7.25	**3.49**
Dat-20-32	0.51	**0.39**	**0.39**	1.50 (**0.50***)	1.16	1.07	**1.38**	2.30	1.86	—	7.39	**3.93**
Dat-20-64	0.51	0.38	**0.35**	1.52 (**0.54***)	1.42	1.35	**1.41**	2.50	1.52	—	7.89	**3.83**
Dat-20-256	0.51	**0.41**	0.44	1.43 (**0.59***)	1.64	1.42	**1.30**	3.18	2.29	—	8.98	**4.25**
Dat-50-16	0.55	0.34	**0.29**	1.50 (**0.48***)	1.51	1.13	**1.36**	2.66	2.08	—	8.53	**4.46**
Dat-50-32	0.55	**0.37**	**0.37**	1.53 (**0.54***)	1.41	0.97	**1.40**	2.66	1.89	—	8.63	**4.39**
Dat-50-64	0.56	0.53	**0.32**	1.57 (**0.58***)	1.58	1.15	**1.38**	3.30	2.28	—	8.60	**4.65**
Dat-50-256	0.56	**0.40**	0.47	1.46 (**0.64***)	2.03	1.56	**1.32**	3.57	2.47	—	10.18	**4.74**
Dat-100-16	0.59	0.34	**0.33**	1.61 (**0.53***)	1.49	1.16	**1.41**	3.07	2.05	—	8.19	**4.35**
Dat-100-32	0.59	**0.36**	0.37	1.6 (**0.58***)	1.38	1.42	**1.4**	3.28	2.59	—	9.68	**4.25**
Dat-100-64	0.60	0.43	**0.30**	1.6 (**0.62***)	1.67	1.22	**1.4**	4.16	3.13	—	9.43	**5.19**
Dat-100-256	0.61	0.56	**0.46**	1.50 (**0.66***)	1.97	1.57	**1.31**	4.50	2.97	—	11.78	**7.19**

* Values in parentheses are performed counting sequentially in the baseline sequence representation

Table 6.3: Query times for access (Acc) and counting queries. Times in μs/query

competitive in query times and use up to 10 times less space. Moreover, our proposals are faster than the indexed *CMHD* representation in many cases.

In the last two queries a naive traversal of the sequence in the baseline becomes too slow (5 to 20 times slower than our proposals) so we only compare with the *CMHD*. In query C-1D-aE-1A the *CMHD* can be faster by storing cumulative values for the corresponding query, but our proposals are still competitive. In addition, our proposals can answer a generalization of the query to a range of days (C-rD-aE-1A) in roughly the same time, whereas the *CMHD* would have to resort to summing up multiple entries, becoming much slower. This is also shown in query C-rD-1E-1A: our proposals are a bit slower in this query, but query times are not too high in comparison with previous queries; however, the *CMHD* would have to obtain and sum cumulative values for each day, becoming too slow in practice, with times again comparable to a naive traversal of the sequence. A partial improvement on the *CMHD* could be obtained by generating fixed divisions as time hierarchies (by week, by month), reducing the number of sums to perform, but this requires previous knowledge of the desired ranges, and unless queries

involve ranges exactly matched with hierarchical values, the *CMHD* is expected to be much slower in these queries. While the *CMHD* is much more dependent on a previous definition of hierarchies of interest, our proposals are more flexible and can efficiently answer range queries on the different dimensions.

6.5 Conclusions

This chapter has introduced two different compact representations for multidimensional sequences with support for aggregation queries. Both are based on *stacked wavelet trees*, ordering each tree the information attending a different criteria. The first proposal uses plain *wavelet trees* as ordering tools while the improved version takes advantage of data locality avoiding repetitions in the branches storing very large datasets in reduced space. The experiments show that our proposals are smaller, and faster to access, than simpler representations of multidimensional sequences. Also, and even if storing much more information, our proposals are competitive on aggregation queries with state-of-the-art data structures designed specifically for those queries.

Chapter 7

Conclusions and future work

This thesis has tackled the challenges associated with the management of event sequences, from space usage reduction during storage to solving aggregated queries efficiently. One of the main issues treated in this work has been the multiple reorganizations that an event sequence can acquire depending on the exploitation results we want to achieve. Again, if we need to retrieve theatre plays written by an author from a library it would be really helpful if the books are sorted by literary genre but searching by another dimension such as author could be easier. Probably the best option would always be an order hierarchy: first by genre, then by author and last by publication date. The methods and structures we have proposed to tackle this and other related problems are summarized in the next subsection.

7.1 Conclusions

Along the previous chapters we have focused on three main issues of general sequences and, particularly, event sequences. Before going deeper into our work it may be necessary to refresh the definition of a sequence as an ordered list of events of the form:

$$E^1_{a,i,\ldots x}; E^2_{b,j,\ldots y}; E^3_{a,k,\ldots z}; E^4_{c,i,\ldots y}; \ldots$$

Where the event identifier is depicted as a superscript and the subscripts identify different characteristics of the event ($dimension_1$

could be a, b, c; $dimension_2$ could be i, j, k; etc.).

With this basic arrangement in mind, it is not difficult to think about the three main problems encompassed in this manuscript: dynamic compression, aggregated queries and multiple indexing. The former concerns about the space usage on dynamic sequences, if a sequence is generated on the fly (e.g. text messages) it is necessary to have an efficient method able to compress it as it is created. Also, it could be interesting to extract aggregated data of a sequence (e.g. most purchased product of the month) without jeopardize the space consumption. Last, as discussed above, each element on a sequence may have some dimensions that characterize it somehow; exploiting this associated information could lead to the acquisition of some hidden knowledge in the original sequence (e.g. pattern searching) in addition to search time improvements as in the library example. We have been capable of overcoming this challenges with our new brand general-purpose proposals, namely:

- **D-V2V.** This structure is born as a combination of the two main families of compression algorithms: dictionary-based compressors (variable-length-symbols-to-fixed-length-codes) and statistical compressors (fixed-length-symbols-to-variable-length-codes). Our variable-to-variable compression technique merges the best characteristics of both worlds achieving searcheable compressed sequences with better compression times than the former and better compressed ratios than the latter. Besides, *D-V2V* is conceived to perform in critical transmission scenarios where it is not possible to preprocess the sequence as semi-static solutions do. Therefore, our algorithm works dynamically building coding rules from the already processed input data and reordering them by frequency as the sequence keeps coming to maintain an optimal encoding.

  To verify the good performance of our structure we have tested it in some well-known natural language datasets and compared the effectiveness against other state of the art compressors from both families as well as the semistatic version of our proposal. *D-V2V* yields considerably better compression ratios than its semi-static and dynamic predecessors, *V2Vc* and *DETC* respectively, while achieving competitive results against popular compressors such as

p7zip or *lzma*. However, repetitive datasets show different outcomes as our proposal increases the improvement over *DETDC* and the semi-static *V2Vc* but the distance from the standard solutions also increases. On the other hand, time measurements show a total opposite result. *D-V2V* is faster than *RePair* and *lzma* but is outperformed by its semi-static version and, particularly, by (the much simpler) *DETDC*. Finally, the main drawback of our proposal is the high memory usage required to hold the already-known sequences while reading a text, which exceeds space requirements of most of our competitors.

- **T-Matrices.** As a generalization of a computer graphics technique, we have extended and improved this solution with the aim of creating a general-purpose structure capable of solving aggregation queries efficiently. The main idea of this contribution is to store precaculated accumulative matrices of data in order to solve aggregated queries (e.g. compute how many passengers were on a bus) as quickly as possible. Two improvements regarding space footprint were introduced and tested in different real contexts as well. Concerning the experiments, we have detailed the perks of using our structure in diverse contexts along with the positive results obtained, highlighting the compression ratios.

  We also presented *semantrix*, a structure that combines *T-Matrices* with other simpler structures in order to solve a wider range of queries skillfully. It is important to highlight that our proposal outperformed the naive approach, successfully solving aggregations straightway but, in exchange, requiring more space. This is the reason why improved flavours of our proposal were proposed in order to reduce the space footprint.

- **Stacked Wavelet Trees.** We have employed the well-known structure known as *wavelet tree* to build order hierarchies above sequences favoring dynamic configurations that suit the possible shifting targets. Thus, each tree orders the sequence following a particular criterion and this new ordered sequence is the input of the next tree (with a different order criterion). This composed structure is not only useful for sorting the information easing the

searching process but also for extracting relevant hidden pattern information. The main advantage of our proposal is that it is able to solve aggregated queries efficiently without explicitly storing accumulative data as other classical aggregated solutions.

An improved compact version of our original proposal was also explained, avoiding unnecessary redundancy and boosting the compression ratio. We have detailed the compressed solutions used as benchmark and how we tested our structures over a wide range of datasets in order to analyze how much does data granularity affect the behaviour of our proposal. Both of our solutions, *wtrle* and the compressed *wtmap*, occupy considerable less space in memory than the baseline while outperforming it at non-consecutive aggregation and access queries.

7.2 Future work

It is not trivial to set the final milestone on a research, it does not matter how efficient our structures are, they can always be improved. This section gathers the ideas, twists and advances that we were not able to fit in this manuscript due to time constraints:

- **D-V2V.** One of the most interesting future lines would be to implement direct searching within the compressed stream. Note that looking for the occurrences of a given word P would be possible by counting the number of escape codewords until the first occurrence of P (that counter indicates the initial entry of the *codebook* where we add P). From there on, by simulating the decompressing process we need to track the occurrences of the codeword corresponding to the terminal P and those codewords corresponding to all the non-terminals which include P.

  Also, based on the ideas of [NR04], it could be possible to build a phrase/word searcher. The main idea would be to implement a complex system of prefixes and suffixes able to count not only the occurences of each word as in the previous idea, but also the occurrence of a sought phrase even if it is divided in different compressed text chunks.

On the other hand, the main drawback of *D-V2V* is that it needs lots of memory at compression to handle the subsequences (non-terminals) in the tree. As future work, we will try to improve the current implementation of the tree to reduce memory requirements. We also want to apply the ideas in [BFNP10] to create an asymmetric lightweight version of *D-V2V*. This should reduce the work done by the receiver and its memory usage. In addition, the codeword associated to a given symbol S_i would not vary so often, which could allow us to implement efficient direct searches for a pattern within the compressed text.

- **T-Matrices.** Regarding future work, the first step will be to increase the scope of this work in order to represent somehow geometries in a compact way inside our structures. This idea opens a wide new field of possibilities to perform queries combining spatial, temporal and semantic constraints.

  Also, as *semantrix* has demonstrated it still needs auxiliary structures to solve efficiently non-aggregated queries. It would be really handful to create an extension of this project overcoming this drawback.

- **Stacked Wavelet Trees.** An interesting line of work is to explore the effect of the encoding of the symbols in real contexts. In our running example, some activities may be more likely to be performed after a specific one than others. For example, "take-a-nap" activity it is more likely to be performed after "lunch" rather than after "breakfast". By encoding *similar* activities with closer symbols, these regularities are automatically exploded by the *wtrle*, but not for any other of the solutions.

Appendix A

Publications and other research results

Publications

Journals

- Nieves R. Brisaboa, Antonio Fariña, Miguel R. Luaces, Daniil Galaktionov, and Tirso V. Rodeiro. **Trippy: a GIS to visualize aggregated users' trips data built on a compact representation.** *Draft to submit.*
- Nieves R. Brisaboa, Antonio Fariña, Daniil Galaktionov, Tirso V. Rodeiro, and M. Andrea Rodríguez. **Improved structures to solve aggregated queries for trips over public transportation networks.** *Submitted to Information Sciences.*

International conferences

- Nieves R. Brisaboa, Antonio Fariña, Gonzalo Navarro, and Tirso V. Rodeiro. **Semantrix: A Compressed Semantic Matrix.** *In Proceedings of the 2020 Data Compression Conference (DCC 2020) IEEE Computer Society, Snowbird, Utah (United States), pp. 113-122, 2020.*

- Nieves R. Brisaboa, Antonio Fariña, Adrián Gómez-Brandón, Gonzalo Navarro, and Tirso V. Rodeiro. **Dv2v: A Dynamic Variable-to-Variable Compressor.** *In Proceedings of the 2019 Data Compression Conference (DCC 2019), IEEE Computer Society, Snowbird, Utah (United States), pp. 83-92, 2019.*

- Nieves R. Brisaboa, Guillermo de Bernardo, Gonzalo Navarro, Tirso V. Rodeiro, and Diego Seco. **Compact Representations of Event Sequences.** *In Proceedings of the 2018 Data Compression Conference (DCC 2018), IEEE Computer Society, Snowbird, Utah (United States), pp. 237-246, 2018.*

- Nieves R. Brisaboa, Antonio Fariña, Daniil Galaktionov, Tirso V. Rodeiro, and M. Andrea Rodríguez. **New Structures to Solve Aggregated Queries for Trips over Public Transportation Networks.** *In Proceddings of the 25th International Symposium on String Processing and Information Retrieval (SPIRE 2018) - LNCS 11147, Springer, Lima (Perú), pp. 88-101, 2018.*

- Alejandro Cortiñas, Miguel R. Luaces, and Tirso V. Rodeiro. **A Case Study on Visualizing Large Spatial Datasets in a Web-Based Map Viewer.** *In Proceddings of Web Engineering - 18th International Conference (ICWE 2018), LNCS 10845, Springer, Cáceres (Spain), pp. 296-303, 2018.*

- Alejandro Cortiñas, Miguel R. Luaces, and Tirso V. Rodeiro. **Storing and Clustering Large Spatial Datasets Using Big Data Technologies.** I*n Proceddings of Web and Wireless Geographical Information Systems - 16th International Symposium (W2GIS 2018) - LNCS 10819, Springer , A Coruña (Spain), pp. 15-24, 2018.*

International research stays

- *10th April, 2018 - 11th July, 2018.* Research stay at Universidad de Concepción, Departamento de Informática y Ciencias de la Computación (Concepción, Chile).
- *3rd January, 2019 - 1st March, 2019.* Research stay at University of Melbourne, School of Computing and Information Systems (Melbourne, Australia).
- *11th October, 2019 - 12th December, 2019.* Research stay at University of Kyushu, Department of Informatics (Kyushu, Japan).

Appendix B

Resumen del trabajo realizado

B.1 Introducción

Hace muchos, muchos años, mucho antes de la llegada de los Kindles y los iPads, la humanidad consiguió doblegar el espacio y el tiempo con la invención de las tablillas de barro. Este pequeño artilugio permitía encerrar un pensamiento abstracto en un medio físico y fácilmente transportable. Todos conocemos, o podemos imaginar, el impacto que esto causó en las sociedades de nuestros antepasados: educación, tratados, registros de contabilidad, cartas, dibujos, leyendas... Como es lógico, con el paso del tiempo las tablillas fueron incorporando mejoras, desde el uso de diversos materiales (madera, metal, marfil...) hasta la inclusión en el estilo[1] de una espátula para borrar de manera sencilla. Sin embargo, este gran invento presentaba una limitación fundamental: el restringido espacio de escritura. Ni siquiera aunque se escribiera por los dos lados cabían textos extensos. Por tanto, cuanto más largo fuese el texto a escribir, más tablillas eran necesarias, con las correspondientes complicaciones que eso conlleva (peso, precio, mayor posibilidad de perder o de que se humedeciera alguna de las tablillas y dejar un texto inconexo...). Habría que esperar siglos para la invención de un nuevo mecanismo que hiciese frente a estas trabas: el papiro.

[1] Aguja con la que se escribía sobre la tablilla.

Estas construcciones de fibras vegetales eran finas, flexibles, ligeras y, a pesar de solo tener una cara aprovechable, un solo rollo de dimensiones habituales podía contener una tragedia griega completa o un evangelio entero. Además, el papiro se puede enrrollar, almacenando una gran cantidad de texto en muy poco espacio. Igual que la escritura evolucionó para almacenar más información en menos espacio, las propuestas de esta tesis continúan combatiendo las limitaciones espaciales a través de "papiros virtuales" que almacenan los datos de una forma flexible y compacta.

La incorporación de los papiros a la sociedad trajo también nuevos dilemas que resolver; por ejemplo, en la biblioteca de Alejandría era tal la cantidad de papiros almacenados que era imposible encontrar una obra concreta. Para terminar con este problema era necesario establecer un orden que permitiese organizar la bilblioteca. Zenódoto, primer director de la biblioteca de Alejandría, contribuyó a la resolución de este problema usando por primera vez el orden alfabético para separar en cantos los largos poemas de Homero. Poco más tarde, Calímaco, sucesor de Zenódoto, inventaría los primeros catálogos de libros bautizándolos como "Pinakes". El sistema de Calímaco dividía las obras en seis géneros literarios y en cinco tipos de prosa y, dentro de cada categoría, las obras estaban ordenadas alfabéticamente por autor. El impacto de esta pequeña reorganización fue de tal envergadura que se siguieron usando variaciones de los "Pinakes" originales hasta el siglo XIX. Siguiendo estas ideas del poder de la reorganización, esta tesis usa una aproximación similar para mejorar la accesibilidad y los tiempos de búsqueda. Igual que las obras de Alejandría, si la información está organizada por las dimensiones adecuadas será más sencillo encontrar lo que buscamos.

Siglos más tarde, durante el otoño de 1848, se produjo una epidemia de cólera en Inglaterra, causando gran mortalidad. Para aquel entonces, no se conocía con certeza el modo de transmisión de esta enfermedad, enfrentándose dos corrientes teóricas. Por un lado estaban los "contagionistas", quienes sostenían que el cólera se adquiría por el contacto con el enfermo o con sus vestidos y pertenencias. Por otro lado, estaban los que apoyaban la teoría "miasmática". Esta teoría postulaba que ciertas condiciones atmosféricas, en especial los vientos, transmitían de un lugar a otro los "miasmas": vapores tóxicos emitidos

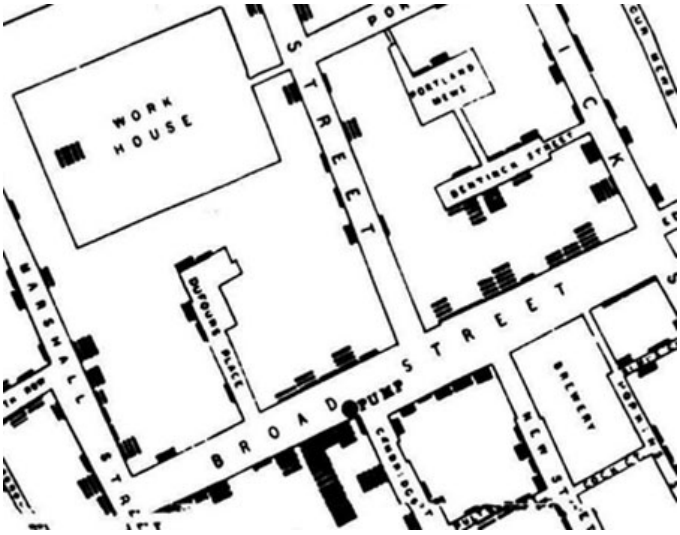


Figure B.1: Mapa de las muertes por cólera del doctor Snow. La bomba de agua está localizada en la intersección de Broad Street con Cambridge Street. Las barras negras reflejan el número de muertos en cada zona. 1854. (Fuente: https://johnsnow.matrix.msu.edu/book_images12.php)

por materia en descomposición, los cuales "transportaban" de un lugar a otro el cólera. El doctor John Snow, que no confiaba en ninguna de estas teorías, se propuso encontrar la causa real de los contagios. Fue así como surgió la idea del popularmente conocido como "mapa del cólera" (Figura B.1). Siguiendo la pista de un severo brote en el sur de la ciudad, el doctor Snow confeccionó un mapa del sector, en el cual marcó los puntos correspondientes a defunciones por cólera y las distintas bombas de agua potable existentes, demostrando gráficamente la relación espacial entre las muertes por cólera y la bomba de Broad Street. Finalmente, el estudio de la bomba demostró que una tubería de alcantarillado pasaba a escasa distancia de la fuente de agua de la bomba, existiendo filtraciones entre ambos cursos de agua. Siguiendo la estela del doctor Snow, hoy en día son comunes los sistemas encargados de recolectar y agregar datos para acceder a información oculta en los eventos individuales y esta tesis aporta una nueva contribución en este frente.

B.2 Motivación

Como ha sugerido el apartado anterior, esta tesis estudia tres de los problemas más habituales en la explotación de secuencias en general y secuencias de ventos en particular: compresión dinámica, indexación múltiple y explotación de datos acumulados. Una secuencia de eventos es conceptualmente una serie de elementos ordenados unidimensionalmente representando cada uno de los elementos un evento que ocurre en un tiempo concreto. Además del tiempo, habitualmente se pueden encontrar otras dimensiones, características o atributos donde cada evento tiene un valor determinado.

Consideremos por ejemplo la publicación de trabajos literarios del Siglo de Oro español. La publicación de cada obra es un evento donde existen, además de la dimensión temporal, otras características relevantes como el autor o el género literario al que pertenece. Esta secuencia de eventos no tiene elementos repetidos, pero podría ocurrir. En el caso de una secuenca de palabras en un texto a transmitir, cada palabra puede ser vista como un evento con un tiempo de transmisión además de otras dimensiones como el tamaño o la caterogría sintáctica. Generalizando, una secuencia de eventos es una lista ordenada de elementos de la forma:

$$E^1_{a,i,\ldots x}; E^2_{b,j,\ldots y}; E^3_{a,k,\ldots z}; E^4_{c,i,\ldots y}; \ldots$$

Siendo los superíndices el identificador de evento y los subíndices *a, b, c* los valores de la característica 1, los valores *i, j, k* los valores de la característica 2 y *x, y, z* los de la característica n. No obstante, la mayor diferencia al comparar la lista de obras literarias con la secuencia de palabras es el grado de repetición, que puede ser explotado por técnicas de compresión.

Aunque el orden cronológico siempre es posible en las secuencias de eventos, a veces una organización alternativa puede resultar más beneficiosa. Volviendo al ejemplo de las publicaciones del Siglo de Oro, el orden temporal es el más obvio pero también sería posible ordenarlas por autor, por género literario o incluso por una combinación de ambas. Esta combinación de dimensiones permitiría crear jerarquías de orden. Por ejemplo, sería posible clasificar las obras por género (primero novelas,

después obras teatrales y por último poemarios), dentro de cada género ordenarlas por orden alfabético según el nombre del autor y, finalmente, un orden de tercer nivel por fecha de publicación. Pero tal vez otras combinaciones jerárquicas serían más útiles para explotar distintintos tipos de consultas: por autor en el primer nivel y por género en el segundo; primero por fecha, luego por autor y después por género, etc. Un sistema capaz de gestionar las diferentes configuraciones sería realmente útil.

Además, hay que tener en cuenta que algunos eventos tienen entre sus características una de tipo espacial. Por ejemplo, consideremos la secuencia de tramos de calle[2] recorridas cada dia por una flota de taxis. En este caso un evento sería que un taxi recorriera un tramo de calle y las características que definirían el evento serían el tiempo, el identificador del taxista y, por supuesto, el identificador del propio tramo de calle. Obsérvese que de nuevo aquí son aplicables las consideraciones sobre las necesidades de compresión de la secuencia, su reordenación y su acumulación para resolver consultas. Así por ejemplo, obsérvese que la ordenación de los eventos de forma puramente temporal podría ser poco últil, por el contrario sería más interesante reordenar los eventos por taxista y dentro de cada taxista por tiempo. Esto nos permitiría ver las trayectorias habituales de cada persona cada día e incluso su evolución en el tiempo. Pero también podría ser útil hacer una ordenación por días y, dentro de cada día, por taxista y, a su vez, dentro de cada taxista ordenar los tramos por hora de modo que nos facilitaría reconstruír las trayectorias diarias de cada conductor.

B.3 Contribuciones

Esta tesis estudia los retos asociados a las secuencias de eventos, desde las restricciones espaciales durante su almacenamiento hasta la velocidad de búsqueda en consultas agregadas. Para conseguir esto, se han presentado y analizado tres estructuras básicas que se ocupan de los aspectos fundamentales de las secuencias en general y de las secuencias de eventos en particular:

[2]Entendiendo un tramo de calle como el que no tiene entradas ni salidas, es decir, la sección entre dos cruces.

- **DV2V.** Esta propuesta nació de la combinación de las dos familias de compresión de textos más importantes: compresores estadísticos (identifican símbolos de longitud fija con códigos de longitud variable) y compresores basados en diccionarios (identifican símbolos de longitud variable con códigos de longitud fija). Nuestra propuesta fusiona características de ambos mundos identificando símbolos de longitud variable con códigos de longitud variable siendo capaz así de obtener secuencias comprimidas buscables, mejores ratios de compresión que los compresores estadísticos y mejores tiempos de compresión que los compresores basados en diccionario. Además, *D-V2V* ha sido pensado para actuar en escenarios críticos donde no es posible preprocesar la secuencia como hacen las soluciones estáticas. Así, nuestro algoritmo trabaja dinámicamente construyendo códigos y reordenándolos por frecuencia al mismo tiempo que va llegando la secuencia.

  Para verificar el buen rendimiento de nuestra estructura, la hemos comparado con compresores conocidos de ambas familias así como a su propio ancestro estático. *D-V2V* se ha comportado según lo esperado en cuanto a tiempos de compresión y descompresión además de obtener unos resultados competitivos en cuanto a ratios de compresión.

- **T-Matrices.** Partiendo de una técnica para renderizar gráficos de forma eficiente, se ha presentado una generalización que extiende y mejora la ténica original con el fin de crear una estructura de carácter general capaz de resolver consultas agregadas eficazmente. La idea principal de esta contribución es almacenar matrices de datos acumulados para resolver consultas agregadas (p. ej. calcular cuántos pasajeros había en un autobús) lo más rápidamente posible. Se han propuesto dos mejoras en cuanto al uso de espacio y se ha testeado esta estructura en diferentes contextos de aplicación. En cuanto a los experimentos, se han detallado las ventajas de usar nuestra propuesta en diversos contextos así como los resultados positivos obtenidos, destacando los ratios de compresión.

  Hemos presentado también *semantrix*, una estructura que combina las *T-Matrices* con otras estructuras más simples para poder resolver un mayor abanico de consultas hábilmente.

- **Wavelet trees apilados.** Hemos utilizado la estructra conocida como *wavelet tree* para construir jerarquías de orden sobre secuencias con el fin de permitir configuraciones dinámicas que se adapten a los distintos objetivos. Así, cada árbol ordena la secuencia siguiendo un criterio particular y esta nueva secuencia ordenada es la entrada para el siguiente árbol (que ordenará la secuencia siguiendo otro criterio distinto). Esta estructura compuesta no solo es útil para ordenar la información facilitando las búsquedas sino que también permite extraer patrones ocultos en la información. La principal ventaja de nuestra propuesta es que es capaz de resolver consultas agregadas rápidamente sin almacenar explícitamente la información acumulada como hacen las soluciones agregadas clásicas.

  Nuestra investigación también incluye una versión mejorada y compacta de nuestra propuesta original, evitando redundancia inncesaria para mejorar considerablemente el ratio de compresión. En esta tesis se han detallado las soluciones comprimidas contra las que se ha comparado nuestra propuesta y cómo se han hecho pruebas sobre una amplia gama de conjuntos de datos con el objetivo de analizar cuánta influencia tiene el nivel de granularidad de la información en el comportamiento de nuestra estructura.

Además de proponer soluciones para la compresión, la agregación y el indexado de las secuencias de eventos, en esta tesis se han afrontado problemas reales que se pueden solucionar con estas técnicas. Por ello, una contribución adicional de esta investigación es el modelado de problemas de logística como secuencias de eventos y su solución utilizando las técnicas propuestas en este texto.

B.4 Trabajo futuro

No es trivial establecer el punto final en una investigación, por muy compactas y eficientes que sean las contribuciones siempre se podrán mejorar. Esta sección resume las ideas y mejoras que no se han incluído en este manuscrito debido a las restricciones temporales:

- **D-V2V.** Una de las líneas futuras más interesantes sería la capacidad de realizar búsquedas directas. Hay que tener en cuenta que buscar cuántas veces aparece en un texto la palabra P sería posible contando el número de códigos de escape hasta la primera ocurrencia de esa palabra. De ahí en adelante habría que simular el proceso de descompresión para seguir la pista de todas las ocurrencias de P, incluyendo los nodos no terminales donde P llega como parte de una frase más larga.

  Por otro lado, uno de los mayores inconvenientes del *D-V2V* es la cantidad de memoria que necesita para hacerse cargo de los nodos no terminales del árbol. Un trabajo futuro sería mejorar esta impementación del árbol intentando reducir los requisitos de espacio. También se ha pensado en aplicar las ideas de [BFNP10] para construir una versión asimétrica y más ligera de nuestra propuesta. Esto debería aliviar la carga de trabajo realizada por el receptor y el espacio usado. Además, el código asociado a un símbolo concreto S_i no variaría tan a menudo, lo que permitiría implementar búsquedas directas de patrones en el texto comprimido.

- **T-Matrices.** En cuanto a trabajo futuro, el primer paso sería aumentar el alcance de esta contribución para incluír de alguna manera las geometrías asociadas a las dimensiones de interés de una forma compacta. Esta idea abre un nuevo campo de posiblidades para realizar consultas mezclando información espacial, temporal y semántica.

  Además, como ha quedado patente en *semantrix*, todavía son necesarias estructuras auxiliares para resolver eficientemente consultas no agregadas. Sería de mucha ayuda crear una extensión de esta contribución unificando todo el proceso en una sola estructura de una forma flexible y eficaz.

- **Wavelet trees apilados.** Una línea interesante de trabajo sería explorar los efectos de la codificación de símbolos en dominios reales. En nuestro contexto conreto, seguramente algunas actividades son más propensas a ser realizadas después de otras. Por ejemplo, la actividad “dormir la siesta” es más propensa a ocurrir después de la actividad “comer” que después de la actividad

"desayunar". Asignando códigos cercanos a actividades similares estas regularidades se explotarían automáticamente con nuestra propuesta *wtrle*, al contrario que con el resto de soluciones.

Bibliography

[AAMF16] Azalden Alsger, Behrang Assemi, Mahmoud Mesbah, and Luis Ferreira. Validating and improving public transport origin–destination estimation algorithm using smart card fare data. *Transportation Research Part C: Emerging Technologies*, 68:490–506, 2016.

[Bac66] Charles W. Bachman. On a generalized language for file organization and manipulation. *Communications of the ACM*, 9(3):225–226, 1966.

[BCN10] Nieves R. Brisaboa, Ana Cerdeira-Pena, and Gonzalo Navarro. A compressed self-indexed representation of XML documents. In Ernest Teniente and Silvia Abrahão, editors, *XV Jornadas de Ingeniería del Software y Bases de Datos (JISBD)*, pages 199–199, 2010.

[BCPL+16] Nieves R. Brisaboa, Ana Cerdeira-Pena, Narciso Lopez Lopez, Gonzalo Navarro, Miguel R. Penabad, and Fernando Silva-Coira. Efficient representation of multidimensional data over hierarchical domains. In *Proceedings of the 23rd International Symposium on String Processing and Information Retrieval*, pages 191–203, 2016.

[BFG+18] Nieves R. Brisaboa, Antonio Fariña, Daniil Galaktionov, Tirso V. Rodeiro, and M. Andrea Rodríguez. New structures to solve aggregated queries for trips over public transportation networks. In *Proceedings of the 25th String*

Processing and Information Retrieval (SPIRE), pages 88–101, 2018.

[BFL+03] Nieves R. Brisaboa, Antonio Fariña, Miguel R. Luaces, José R. Paramá, Miguel R. Penabad, Ángeles S. Places, and José R. R. Viqueira. Using geographical information systems to browse touristic information. *Journal of Information Technology & Tourism*, 6(1):31–46, 2003.

[BFL+10] Nieves R. Brisaboa, Antonio Fariña, Juan-Ramón López, Gonzalo Navarro, and Eduardo R. López. A new searchable variable-to-variable compressor. In *Proceedings of the Data Compression Conference (DCC)*, pages 199–208, 2010.

[BFNP07] Nieves R. Brisaboa, Antonio Fariña, Gonzalo Navarro, and José Paramá. Lightweight natural language text compression. *Information Retrieval*, 10(1):1–33, 2007.

[BFNP08] Nieves R. Brisaboa, Antonio Fariña, Gonzalo Navarro, and José Paramá. New adaptive compressors for natural language text. *Software: Practice and Experience*, 38(13):1429–1450, 2008.

[BFNP10] Nieves R. Brisaboa, Antonio Fariña, Gonzalo Navarro, and José R. Paramá. Dynamic lightweight text compression. *ACM Transactions on Information Systems (TOIS)*, pages 1–32, 2010.

[BHL+00] Dimitri P. Bertsekas, Mark L. Homer, David A. Logan, Stephen D. Patek, and Nils R. Sandell. Missile defense and interceptor allocation by neuro-dynamic programming. *IEEE Transactions on Systems, Man, and Cybernetics*, 30(1):42–51, 2000.

[BK15] Maria Börjesson and Ida Kristoffersson. The Gothenburg congestion charge. Effects, design and politics. *Transportation Research Part A: Policy and Practice*, 75(C):134–146, 2015.

[BLPP17] Nieves R. Brisaboa, Miguel R. Luaces, Cristina M. Pérez, and Ángeles S. Places. Semantic trajectories in mobile workforce management applications. In *Proceedings of the 15th International Symposium on Web and Wireless Geographical Information Systems, W2GIS*, volume 10181 of *Lecture Notes in Computer Science*, pages 100–115, 2017.

[Bly00] Phil T. Blythe. Transforming access to and payment for transport services through the use of smart cards. *Journal Intelligent Transport Systems*, 6(1):45–68, 2000.

[BM77] Robert S. Boyer and J. Strother Moore. A fast string searching algorithm. *Communications of the ACM*, 20(10):762–772, 1977.

[BvDD09] Jaap Bloema, Menno van Doorn, and Sander Duivestein. *Me the Media - Rise of the Conversation Society*. Sogeti - Vint Editions, 2009.

[BW94] Michael Burrows and David J. Wheeler. A block-sorting lossless data compression algorithm. Technical report, Digital Equipment Corporation, 1994.

[BYRN08] Ricardo Baeza-Yates and Berthier Ribeiro-Neto. *Modern Information Retrieval: The Concepts and Technology Behind Search*. Pearson Education Ltd., 2nd edition, 2008.

[Cle67] Cyril Cleverdon. The cranfield tests on index language devices. In *Aslib Proceedings, Vol. 19 No. 6*, ISSN 0001-253X, pages 173–194, 1967.

[CN09] Francisco Claude and Gonzalo Navarro. Practical rank/select queries over arbitrary sequences. In *Proceedings of the 15th International Symposium on String Processing and Information Retrieval*, SPIRE '08, pages 176–187, 2009.

[Cod70] Edgar F. Codd. A relational model of data for large shared data banks. *Communications of the ACM*, 13(6):377–387, 1970.

[Cod79] Edgar F. Codd. Extending the database relational model to capture more meaning. *ACM Transactions on Database Systems*, 4(4):397–434, 1979.

[Cro84] Franklin C. Crow. Summed-area tables for texture mapping. *ACM SIGGRAPH Computer Graphics*, 18(3):207–212, 1984.

[CS93] Edgar F. Codd and C. Salley. Providing olap to user-analysts: An it mandate. 1993.

[CW84] John Cleary and Ian Witten. Data compression using adaptive coding and partial string matching. *IEEE Transactions on Communications*, 32(4):396–402, 1984.

[DBS04] Eduardo Dias, Euro Beinat, and Henk J. Scholten. Effects of mobile information sharing in natural parks. In *Sh@ring: 18th International Conference "Informatics for Environmental Protection", Part 2*, pages 11–25, 2004.

[dMNZB00] Edleno Silva de Moura, Gonzalo Navarro, Nivio Ziviani, and Ricardo A. Baeza-Yates. Fast and flexible word searching on compressed text. *ACM Transactions onn Information Systems*, 18(2):113–139, 2000.

[dMNZBY98] Edleno de Moura, Gonzalo Navarro, Nivio Ziviani, and Ricardo Baeza-Yates. Fast searching on compressed text allowing errors. In *Proceedings of the 21st Annual Int. ACM SIGIR Conference on Research and Development in Information Retrieval (SIGIR'98)*, pages 298–306, 1998.

[DRR06] O'Neil Delpratt, Naila Rahman, and Rajeev Raman. Engineering the LOUDS succinct tree representation. In *Proceedings of the 5th Int. Workshop on Experimental Algorithms*, pages 134–145, 2006.

[Edm01] Ian R. Edmonds. The use of latent semantic indexing to identify evolutionary trajectories in behaviour space. In *Advances in Artificial Life, 6th European Conference, ECAL Proceedings*, volume 2159 of *Lecture Notes in Computer Science*, pages 613–622, 2001.

[EGSV99] Martin Erwig, Ralf H. Güting, Markus Schneider, and Michalis Vazirgiannis. Spatio-temporal data types: An approach to modeling and querying moving objects in databases. *GeoInformatica*, 3(3):269–296, 1999.

[Fal73] Newton Faller. An adaptive system for data compression. In *Record of the 7th Asilomar Conference on Circuits, Systems, and Computers*, pages 593–597, 1973.

[Far05] Antonio Fariña. *New compression codes for text databases.* PhD thesis, Universidade da Coruña, 2005.

[FCF92] Andrew U. Frank, Irene Campari, and Ubaldo Formentini. Theories and methods of spatio-temporal reasoning in geographic space. In *Proceedings of International Conference GIS - From Space to Territory: Theories and Methods of Spatio-Temporal Reasoning*, volume 639 of *Lecture Notes in Computer Science*, 1992.

[FGG+99] Andrew U. Frank, Stéphane Grumbach, Ralf H. Güting, Christian S. Jensen, Manolis Koubarakis, Nikos A. Lorentzos, Yannis Manolopoulos, Enrico Nardelli, Barbara Pernici, Hans-Jörg Schek, Michel Scholl, Timos K. Sellis, Babis Theodoulidis, and Peter Widmayer. Chorochronos: A research network for spatiotemporal database systems. *Special Interest Group on Management of Data Record*, 28(3):12–21, 1999.

[FGNV08] Paolo Ferragina, Rodrigo González, Gonzalo Navarro, and Rossano Venturini. Compressed text indexes: From theory to practice. *ACM Journal of Experimental Algorithmics*, 13, 2008.

[FM00] Paolo Ferragina and Giovanni Manzini. Opportunistic data structures with applications. In *Proceedings of the 41st Annual Symposium on Foundations of Computer Science (FOCS)*, pages 390–398, 2000.

[FM01] Paolo Ferragina and Giovanni Manzini. An experimental study of a compressed index. *Information Sciences*, 135(1-2):13–28, 2001.

[FM05] Paolo Ferragina and Giovanni Manzini. Indexing compressed text. *Journal of the ACM*, 52(4):552–581, 2005.

[FMMN07] Paolo Ferragina, Giovanni Manzini, Veli Mäkinen, and Gonzalo Navarro. Compressed representations of sequences and full-text indexes. *ACM Transactions on Algorithms*, 3(2):20, 2007.

[Gal78] Robert Gallager. Variations on a theme by huffman. *IEEE Transactions on Information Theory*, 24(6):668–674, 1978.

[GBE+00] Ralf H. Güting, Michael H. Böhlen, Martin Erwig, Christian S. Jensen, Nikos A. Lorentzos, Markus Schneider, and Michalis Vazirgiannis. A foundation for representing and querying moving objects. *ACM Transactions on Database Systems*, 25(1):1–42, 2000.

[GBLP96] Jim Gray, Adam Bosworth, Andrew Layman, and Hamid Pirahesh. Data cube: A relational aggregation operator generalizing group-by, cross-tab, and sub-total. In *Proceedings of the Twelfth International Conference on Data Engineering*, pages 152–159, 1996.

[GGG+07] Alexander Golynski, Roberto Grossi, Ankur Gupta, Rajeev Raman, and Satti Srinivasa Rao. On the size of succinct indices. In *Proceedings of the 15th Annual European Symposium on Algorithms (ESA)*, LNCS 4698, pages 371–382, 2007.

[Góm20] Adrián Gómez-Brandón. Bitvectors with runs and the successor/predecessor problem. In *Proceedings of the Data Compression Conference (DCC)*, pages 133–142, 2020.

[Hor80] Nigel Horspool. Practical fast searching in strings. *Software: Practice and Experience*, 10(6):501–506, 1980.

[Huf52] David A. Huffman. A method for the construction of minimum-redundancy codes. *Proceedings of the I.R.E.*, 40(9):1098–1101, 1952.

[IB86] William H. Inmon and Thomas J. Bird. *The dynamics of data base*. Prentice Hall, 1986. Includes index.

[Inm86] William H. Inmon. *Information Systems Architecture: A System Developer's Primer*. Prentice-Hall, Inc., USA, 1986.

[Inm11] William H. Inmon. A tale of two architectures (1). *Database Magazine*, 1:28–31, 2011.

[Jac89] Guy Jacobson. Space-efficient static trees and graphs. In *Proceedings of the 30th IEEE Symposium on Foundations of Computer Science (FOCS)*, pages 549–554, 1989.

[KGT99] George Kollios, Dimitrios Gunopulos, and Vassilis J. Tsotras. On indexing mobile objects. In *Proceedings of the Eighteenth ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems*, 1999.

[KR02] Ralph Kimball and Margy Ross. *The data warehouse toolkit: the complete guide to dimensional modeling, 2nd Edition*. Wiley, 2002.

[KRS04] Vijay Khatri, Sudha Ram, and Richard T. Snodgrass. Augmenting a conceptual model with geospatiotemporal annotations. *IEEE Transactions on Knowledge and Data Engineering*, 16(11):1324–1338, 2004.

[KSF+03] Manolis Koubarakis, Timos K. Sellis, Andrew U. Frank, Stéphane Grumbach, Ralf H. Güting, Christian S.

Jensen, Nikos A. Lorentzos, Yannis Manolopoulos, Enrico Nardelli, Barbara Pernici, Hans-Jörg Schek, Michel Scholl, Babis Theodoulidis, and Nectaria Tryfona, editors. *Spatio-Temporal Databases: The CHOROCHRONOS Approach*, volume 2520 of *Lecture Notes in Computer Science*, 2003.

[Kwa20] Michael M. Kwakye. Conceptual model and design of semantic trajectory data warehouse. *International Journal of Data Warehousing and Mining (IJDWM)*, 16(3):108–131, 2020.

[Lan89] Gail Langran. A review of temporal database research and its use in GIS applications. *International Journal of Geographic Information Science*, 3(3):215–232, 1989.

[Lan93] Gail Langran. Issues of implementing a spatiotemporal system. *International Journal in Geographic Information Science*, 7(4):305–314, 1993.

[LJZL17] Zhidan Liu, Shiqi Jiang, Pengfei Zhou, and Mo Li. A participatory urban traffic monitoring system: The power of bus riders. *IEEE Transactions on Intelligent Transportation Systems*, 18(10):2851–2864, 2017.

[LM99] N. Jesper Larsson and Alistair Moffat. Offline dictionary-based compression. In *Proceedings of the Data Compression Conference (DCC)*, pages 296–305, 1999.

[Luh58] Hans P. Luhn. A business intelligence system. *IBM Journal of Research & Development*, 2(4):314–319, 1958.

[LVBD17] Juan M. L. Varela, Maria Börjesson, and Andrew Daly. Public transport: one mode or several? Working papers in Transport Economics 2017:6, CTS - Centre for Transport Studies Stockholm (KTH and VTI), 2017.

[MFD03] Ginger Myles, Adrian Friday, and Nigel Davies. Preserving privacy in environments with location-based applications. *IEEE Pervasive Comput.*, 2(1):56–64, 2003.

[MK90] Yannis Manolopoulos and G. Kapetanakis. Overlapping b+trees for temporal data. In *Proceedings of the 5th Jerusalem Conference on Information Technology*, pages 491–498, 1990.

[MM93] Udi Manber and Eugene W. Myers. Suffix arrays: A new method for on-line string searches. *SIAM Journal on Computing*, 22(5):935–948, 1993.

[MM98] François Michaud and Maja J. Mataric. Learning from history for behavior-based mobile robots in non-stationary conditions. *Autonomous Robots*, 5(3-4):335–354, 1998.

[MN05] Veli Mäkinen and Gonzalo Navarro. Succinct suffix arrays based on run-length encoding. In *Proceedings of the 16th Annual Symposium on Combinatorial Pattern Matching*, pages 45–56, 2005.

[Mof89] Alistair Moffat. Word-based text compression. *Software: Practice and Experience*, 19(2):185–198, 1989.

[Mor68] Donald R. Morrison. PATRICIA - practical algorithm to retrieve information coded in alphanumeric. *Journal of the ACM*, 15(4):514–534, 1968.

[MP12] Marcela A. Munizaga and Carolina 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.

[MR01] David Mountain and Jonathan Raper. Modelling human spatio-temporal behaviour: a challenge for location based services. In *Proceedings of the 6th International Conference on GeoComputation*, pages 65–74, 2001.

[Mun96] Ian Munro. Tables. In *Proceedings of the 16th Conference on Foundations of Software Technology and Theoretical Computer Science (FSTTCS)*, LNCS 1180, pages 37–42, 1996.

[Nav16] Gonzalo Navarro. *Compact Data Structures: A Practical Approach*. Cambridge University Press., 2016.

[NdMN+00] Gonzalo Navarro, Edleno de Moura, M. Neubert, Nivio Ziviani, and Ricardo Baeza-Yates. Adding compression to block addr. inverted indexes. *Information Retrieval*, 3(1):49–77, 2000.

[NR04] Gonzalo Navarro and Mathieu Raffinot. Practical and flexible pattern matching over ziv-lempel compressed text. *Journal of Discrete Algorithms*, 2(3):347–371, 2004.

[OA16] Wided Oueslati and Jalel Akaichi. Querying a multi-version trajectory data warehouse. *International Journal of Business Information Systems*, 21(4):403–417, 2016.

[OOR+07] Salvatore Orlando, Renzo Orsini, Alessandra Raffaetà, Alessandro Roncato, and Claudio Silvestri. Trajectory data warehouses: Design and implementation issues. *JCSE*, 1(2):211–232, 2007.

[OS07] Daisuke Okanohara and Kunihiko Sadakane. Practical entropy-compressed rank/select dictionary. In *Proceedings of the 9th Workshop on Algorithm Engineering and Experiments*, 2007.

[Pou19] Jacob Poushter. Pew research center: Smartphone ownership and internet usage continues to climb in emerging economies. `https://www.pewresearch.org/`, 2019.

[PRD+08] Nikos Pelekis, Alessandra Raffaetà, Maria Luisa Damiani, Christelle Vangenot, Gerasimos Marketos, Elias Frentzos, Irene Ntoutsi, and Yannis Theodoridis. Towards trajectory data warehouses. In *Mobility, Data Mining and Privacy - Geographic Knowledge Discovery*, pages 189–211. Springer, 2008.

[PSR+13] Christine Parent, Stefano Spaccapietra, Chiara Renso, Gennady L. Andrienko, Natalia V. Andrienko, Vania

Bogorny, Maria Luisa Damiani, Aris Gkoulalas-Divanis, José A. Fernandes, Nikos Pelekis, Yannis Theodoridis, and Zhixian Yan. Semantic trajectories modeling and analysis. *ACM Computing Surveys*, 45(4):42:1–42:32, 2013.

[RDW+02] Jonathan Raper, Jason Dykes, Jo Wood, David M. Mountain, Anton Krause, and David Rhind. A framework for evaluating geographical information. *Journal in Information Sciences*, 28(1):39–50, 2002.

[RRR02] Rajeev Raman, Venkatesh Raman, and S. Srinivasa Rao. Succinct indexable dictionaries with applications to encoding k-ary trees and multisets. In *Proceedings of the 13th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 233–242, 2002.

[RRS07] Rajeev Raman, Venkatesh Raman, and Srinivasa Rao Satti. Succinct indexable dictionaries with applications to encoding k-ary trees, prefix sums and multisets. *ACM Transactions on Algorithms*, 3(4), 2007.

[RU81] Kari-Jouko Räihä and Esko Ukkonen. The shortest common supersequence problem over binary alphabet is np-complete. *Theoretical Computer Science*, 16:187–198, 1981.

[Sal07] David Salomon. *Variable-length Codes for Data Compression*. Springer-Verlag, 2007.

[Shk02] Dmitry Shkarin. Ppm: One step to practicality. In *Proceedings of the Data Compression Conference (DCC)*, page 202, 2002.

[SIKH82] David C. Smith, Charles H. Irby, Ralph Kimball, and Eric Harslem. The star user interface: an overview. In *Proceedings of the American Federation of Information Processing Societies: 1982 National Computer Conference*, volume 51, pages 515–528, 1982.

[SL65] Gerard Salton and Michael E. Lesk. The SMART automatic document retrieval systems - an illustration. *Communications of the ACM*, 8(6):391–398, 1965.

[TTFR15] M. Carolina Torres, Valéria C. Times, José A. Fernandes, and Chiara Renso. SWOT: A conceptual data warehouse model for semantic trajectories. In *Proceedings of the ACM Eighteenth International Workshop on Data Warehousing and OLAP, DOLAP*, pages 11–14, 2015.

[VW01] Michalis Vazirgiannis and Ouri Wolfson. A spatiotemporal model and language for moving objects on road networks. In *Proceedings of the 7th International Symposium in Advances in Spatial and Temporal Databases*, volume 2121, pages 20–35, 2001.

[VZ13] Alejandro A. Vaisman and Esteban Zimányi. Trajectory data warehouses. In *Mobility Data: Modeling, Management, and Understanding*, pages 62–82. Cambridge University Press, 2013.

[Wan11] Wenpeng Wang. Review on hybrid flow shop scheduling. In *2011 International Conference of Information Technology, Computer Engineering and Management Sciences*, volume 4, pages 7–10. IEEE, 2011.

[Wel84] Terry A. Welch. A technique for high-performance data compression. *Computer*, 17(6):8–19, 1984.

[WFR+13] Ricardo Wagner, José A. Fernandes, Alessandra Raffaetà, Chiara Renso, Alessandro Roncato, and Roberto Trasarti. Mob-warehouse: A semantic approach for mobility analysis with a trajectory data warehouse. In *Advances in Conceptual Modeling - ER 2013 Workshops, LSAWM, MoBiD, RIGiM, SeCoGIS, WISM, DaSeM, SCME, and PhD Symposium*, volume 8697, pages 127–136, 2013.

[WL64] Frederick W. Lancaster. Mechanized document control: A review of some recent research. In *Aslib Proceedings, Vol. 16 No. 4*, pages 132–152, 1964.

[WMB94] Ian H. Witten, Alistair Moffat, and Timothy C. Bell. *Managing Gigabytes: Compressing and Indexing Documents and Images*. Van Nostrand Reinhold, 1994.

[WMB99] Ian Witten, Alistair Moffat, and Timothy Bell. *Managing Gigabytes: Compressing and Indexing Documents and Images*. Morgan Kaufmann Publishers, 1999.

[ZL77] Jacob Ziv and Abraham Lempel. A universal algorithm for sequential data compression. *IEEE Transactions on Information Theory*, 23(3):337–343, 1977.

[ZL78] Jacob Ziv and Abraham Lempel. Compression of individual sequences via variable-rate coding. *IEEE Transactions on Information Theory*, 24(5):530–536, 1978.

[ZM01] Alexander Zipf and Rainer Malaka. Developing location based services for tourism. the service providers' view. In *Information and Communication Technologies in Tourism, ENTER*, pages 83–92, 2001.