ROMA TRE
Università degli Studi di Roma Tre
Dipartimento di Informatica e Automazione
Via della Vasca Navale, 79 – 00146 Roma, Italy

Tensor Calculus for RFID Data Management

Roberto De Virgilio and Franco Milicchio

RT-DIA-182 **March 2011**

Dipartimento di Informatica e Automazione
Università di Roma Tre
{devirgilio,milicchio}@dia.uniroma3.it

ABSTRACT

In current trends of consumer products market, there is a growing significance of the role of retailers in the governance of supply chains. RFID is a promising infrastructure-less technology, allowing to connect an object with its virtual counterpart, *i.e.*, its representation within information systems. However, the amount of RFID data in supply chain management is vast, posing significant challenges for attaining acceptable performance on their analysis. Current approaches provide hard-coded solutions, with high consumption of resources; moreover, these exhibit very limited flexibility dealing with multidimensional queries, at various levels of granularity and complexity.

In this paper we propose a general model for supply chain management based on the first principles of linear algebra, in particular on *tensorial calculus*.

Leveraging our abstract algebraic framework, our technique allows both quick decentralized on-line processing, and centralized off-line massive business logic analysis, according to needs and requirements of supply chain actors. Experimental results show that our approach, utilizing recent linear algebra techniques—tailored to performance and accuracy as required in applied mathematics and physics fields—can process analysis efficiently, when compared to recent approaches. In particular, we are able to carry out the required computations even in high memory constrained environments, such as on mobile devices. Moreover, when dealing with massive amounts of data, we are capable of exploiting recent parallel and distributed technologies, subdividing our tensor objects into sub-blocks, and processing them independently.

1 Introduction

A supply chain is a complex system for transferring products or services from a supplier to a final customer. In the management of a supply chain, main retailers are investing in new technologies in order to boost the information exchange. To this aim, RFID, the Radio-Frequency Identification, is a recent potential wireless technology, which enables a direct link of a product with a virtual one within information systems. Currently, RFID tags attached to products store an identification code named EPC[1], used as a key to retrieve relevant properties of an object from a database, usually within a networked infrastructure.

An RFID application usually generates a stream of tuples, usually called *raw data*, of the form of a triple (e, l, t), where e is an EPC, l represents the location where an RFID reader has scanned the e object, and t is the time when the reading took place. Other properties can be retrieved, *e.g.*, temperature, pressure, and humidity. A single tag may have multiple readings at the same location, thus potentially generating an immense amount of raw data. Therefore, a simple data cleaning technique consists in converting raw data in *stay records* of the form (e, l, t_i, t_o), where t_i and t_o are the time when an object enters or leaves a location l, respectively. In this paper, we will focus on stay records as the basic block to store RFID data.

In this manuscript, we address the challenging problem of manage efficiently the tera-scale amount of data per day, generated by RFID applications.

Recently, two models have been proposed: a *herd* and a *single* model. The former [9] proposes a warehousing model observing that usually, products move together in large groups at the beginning, and continue in small herds towards the end of the chain: this allows the aggregation and reduces significantly the size of the database. On the contrary, the latter [16] focuses on the movement of single, non grouped tags, defining query templates and path encoding schemes to process tracking and path oriented queries efficiently.

Both of these approaches present limited flexibility dealing multidimensional queries at varying levels of granularity and complexity. To exploit the compression mechanism, such proposals have to fix the dimensions of analysis in advance and implement *ad-hoc* data structures to be maintained. It is not known in advance whether objects are sorted and grouped, and therefore, it is problematic to support different kinds of high-level queries efficiently.

In this paper we propose a novel approach based on first principles derived from the linear algebra field. Matrix operations are invaluable tools in several fields, from engineering and design, graph theory, or networking; in particular streams and co-evolving sequences can also be envisioned as matrices: each data source, a sensor, may correspond to a row, while time-ticks to a column (cf. [18]).

Standard matrix approaches focus on a standard two-dimension space, while we extend the applicability of such techniques with the more general definition of *tensors*, a generalization of linear forms, usually represented by matrices. We may therefore take advantage of the vast literature, both theoretic and applied, regarding tensor calculus. Computational algebra is employed in critical applications, such as engineering and physics, and several libraries have been developed with two major goals: *efficiency* and *accuracy*.

[1] EPC stands for Electronic Product Code [8], a coding scheme for RFID tags, aimed at uniquely identify them.

Contribution Leveraging such background, this manuscript proposes a general model of supply chains, mirrored with a formal tensor representation and endowed with specific operators, allowing both quick decentralized on-line processing, and centralized off-line massive business logic analysis, according to needs and requirements of supply chain actors. Our model and operations, inherited by linear algebra and tensor calculus, is therefore theoretically sound, and its implementation may benefit of several numerical libraries developed in the past. Additionally, due to the properties of our tensorial model, we are able to attain two significative features: the possibility of conducting computations in memory-constrained environments such as on mobile devices, and exploiting modern parallel and distributed technologies, owing to the possibility for matrices, and therefore tensors, to be dissected into several chunks, and processed independently (i.e. also on-the-fly).

Outline Our manuscript is organized as follows. In section 2 we will briefly recall the available literature, while Section 3 will be devoted to the introduction of tensors and their associated operations. The general supply chain model, accompanied by a formal tensorial representation is supplied in Section 4, subsequently put into practice in Section 5, where we provide the reader a method of analyzing RFID data within our framework. We benchmark our approach with several test beds, and supply the results in Section 6. Finally, Section 7 sketches conclusion and future work.

2 Related Work

In a real scenario, great lapse and huge amounts of data are generated. To this aim, knowledge representation techniques focus on operating deep analysis in such systems. There exists two main approaches to the management of RFID data. The former is based on processing data streams at run-time [2, 15, 14, 19]. In the latter, the processing is performed off-line, once RFID data are aggregated, compressed and stored [9, 16].

If we consider RFID data as a stream, the main issues are event processing and data cleaning. Wang et al. have proposed a conceptualization of RFID events based on an extension of the ER model [19]. Bai et al. have studied the limitations of using SQL to detect temporal events and have presented an SQL-like language to query such events in an efficient way [2]. The inaccuracy of RFID tags readings causes irregularities in the processing of data. Therefore cleaning techniques need to be applied to RFID data streams as proposed in [15].

An alternative approach consists in warehousing RFID data and performing multidimensional analyses on the warehouse. The focus here is on data compression techniques and on storage models with the goal of achieving a more expressive and effective representation of RFID data. A straightforward method is to provide a support to path queries (e.g., find the average time for products to go from factories to stores in Seattle) by collecting RFID tag movements along the supply chain. Usually, the tag identifier, the location and the time of each RFID reading is gathered and stored in a huge relational table. Gonzalez et al. [9] have presented a new storage model, based on a data warehousing approach, in which items moving together are grouped and data analysis is preformed in a multidimensional fashion, as it happens in a typical data warehouse. Finally, Lee et al. have proposed an effective path encoding approach to represent the data flow representing the movements of products [16]. In a path, a prime number is assigned to each node and

a path is encoded as the product of the number associated with nodes. Mathematical properties of prime numbers guarantee the efficient access to paths.

As we have noticed in the Introduction, a major limitation of the majority of these approaches have to fix the dimensions of analysis in advance to exploit *ad-hoc* data structures to be maintained. It follows that, in many application scenario the compression loses its effectiveness and the size of tables does not be reduced significantly. Moreover, most of the approaches presented in this section present a limited flexibility when multidimensional queries, at varying levels of granularity and complexity, need to be performed.

3 Preliminary Issues

The following paragraphs will be devoted to a brief recall of tensor representations, which is preliminary to the introduction of our model in the ensuing sections.

We take for granted notions and notations of *groups*, *vector spaces*, *matrix rings*, and refer the reader to [12].

Tensors Tensors arise as a natural extension to linear forms. Linear forms on a vector space $\mathbb{V}$ defined over a field $\mathbb{F}$, and belonging to its *dual space* $\mathbb{V}^*$, are maps $\phi : \mathbb{V} \to \mathbb{F}$ that associate to each pair $\phi \in \mathbb{V}^*$, and $v \in \mathbb{V}$, an element $e \in \mathbb{F}$. This is usually denoted by the *pairing* $\langle \phi, v \rangle = e$, or by definition, with a *functional* notation, *i.e.*, $\phi(v) = e$. Such mapping exhibits the linearity property, *i.e.*, $\langle \phi, \alpha\, v + \beta w \,\rangle = \alpha\langle \phi, v\rangle + \beta\langle \phi, w\rangle$.

Generalizing the concept of linearity, we hence may introduce the following:

Definition 1 (Tensor) *A* tensor *is a multilinear form ϕ on a vector space $\mathbb{V}$,* i.e.*, a mapping*

$$\phi : \underbrace{\mathbb{V} \times \ldots \times \mathbb{V}}_{k-\text{times}} \longrightarrow \mathbb{F}\,, \tag{1}$$

the form ϕ assumes also the name of tensor of rank k *(or rank-k tensor).*

The property of *multilinearity* mirrors the previous definition of linearity: ϕ is, in fact, linear in each of its k domains.

As with linear forms, one possible representation of a tensor is with elements of the *matrix ring*. Therefore, a rank-k tensor may be represented by $M \in \mathbb{M}_{i_1 i_2 \ldots i_k}(\mathbb{F})$, that is a k-dimensional matrix with elements in $\mathbb{F}$.

A comprehensive description of tensors and their associated properties is beyond the scope of this manuscript; for a thorough review, see [1, 11], and [13].

Operations Vectors and forms, both belonging to a *vector space*, provide different operations: from basic sum and product of a vector (form) with a scalar to more complex operations that may be defined on vector spaces. In particular, let us consider the following:

Definition 2 (Hadamard Product) *The* Hadamard product *of two vectors $v, w \in \mathbb{V}$ is the* entry-wise *product of their components,* i.e.*,*

$$v \circ w = r, \quad \textit{such that} \quad r_i := v_i w_i\,. \tag{2}$$

To clarify the above definition, we provide the reader with the following example:

Example 1 *Let us consider two vectors* $v, w \in \mathbb{R}^3$*, with* $v = \begin{pmatrix} 1 & 4 & 0 \end{pmatrix}^t$*, and* $w = \begin{pmatrix} 2 & 3 & 9 \end{pmatrix}^t$*, with* $()^t$ *denoting the* transposed *vector. The Hadamard product will be*

$$v \circ w = \begin{pmatrix} 1 \cdot 2 & 4 \cdot 3 & 0 \cdot 9 \end{pmatrix}^t = \begin{pmatrix} 2 & 12 & 0 \end{pmatrix}^t .$$

Since a tensor of rank k will henceforth be denoted by a multidimensional matrix, we shall simplify notations by a variant of *Einstein's notation*. Such variant will be employed wherever a summation occurs: where we encounter two identical indices in a given expression, a summation on that index is implicitly defined.

Let us consider for example, a notable form, the *inner product* $(\cdot)$ between two vectors, known also as *scalar product*. We mention in passing that an inner product arises in defining a particular class of rank-2 tensors: symmetric (or Hermitian) positive-definite forms. If we consider a standard cartesian metrics, *i.e.*, an orthonormal basis [12], the inner product notation can be written leveraging the summation notation described above:

$$v \cdot w = \sum_j v_j w_j =: v_j w_j .$$

As with linear applications, represented by matrices, tensors may be applied to elements of vector spaces, giving rise the following:

Definition 3 (Application) *The* application *of a rank k tensor ϕ, represented by the matrix $M \in \mathbb{M}_{i_1 i_2 \dots i_k}(\mathbb{F})$, to a vector $v \in \mathbb{V}$ is a rank $k-1$ tensor represented by the matrix $\widetilde{M}$:*

$$\widetilde{M}_{i_1 \dots i_{j-1} i_{j+1} \dots i_k} := M_{i_1 \dots i_j \dots i_k} v_{i_j} . \tag{3}$$

The reader should notice that the previous definition of *application* is a generalization of the common matrix-vector product. In other terms, applying a tensor to a vector simply "eats" one dimension, as evidenced by the indices in the previous definition.

Following a standard algebraic practice, indices may be rendered more intelligible when dealing with a tiny number of dimensions, *e.g.*, do not exceed 4 dimensions: in this case, instead of using $M_{i_1 i_2 i_3 i_4}$, we will employ the easier notation of M_{ijkl}.

Example 2 *Let us consider a rank-3 tensor ϕ represented by the matrix $M \in \mathbb{M}_{333}(\mathbb{R})$ and a vector $v \in \mathbb{R}^3$:*

$$M = \begin{pmatrix} \begin{pmatrix} 1 \\ 1 \\ 1 \end{pmatrix} & \begin{pmatrix} 1 \\ 1 \\ 1 \end{pmatrix} & \begin{pmatrix} 1 \\ 1 \\ 1 \end{pmatrix} \\ \begin{pmatrix} 2 \\ 2 \\ 2 \end{pmatrix} & \begin{pmatrix} 2 \\ 2 \\ 2 \end{pmatrix} & \begin{pmatrix} 2 \\ 2 \\ 2 \end{pmatrix} \\ \begin{pmatrix} 3 \\ 3 \\ 3 \end{pmatrix} & \begin{pmatrix} 3 \\ 3 \\ 3 \end{pmatrix} & \begin{pmatrix} 3 \\ 3 \\ 3 \end{pmatrix} \end{pmatrix}, \; v = \begin{pmatrix} 0 \\ 1 \\ 0 \end{pmatrix} .$$

The result of the application of ϕ to v is a rank-2 tensor

$$\langle \phi, v \rangle = M_{ijk} v_i = \begin{pmatrix} 2 & 2 & 2 \\ 2 & 2 & 2 \\ 2 & 2 & 2 \end{pmatrix} .$$

We mention in passing that applications of tensors span beyond simple vectors, with roots in Grassmann algebra, *i.e.*, the algebra of *multidimensional vectors* (cf. [13]).

Definition 4 (Kroneker Tensor) *The rank-k tensor $\delta_{i_1 i_2 \ldots i_k}$, whose matricial representation is $M_{i_1 i_2 \ldots i_k} = 1$ iff $i_1 = i_2 = \ldots = i_k$, is called* Kroneker tensor.

Kroneker tensor is well known in diverse fields ranging from computer science to economics, and it is commonly known as *Kroneker delta*. Moreover, with the canonical duality between forms and vectors [12], it is common to employ the very same symbol for a vector specification, *e.g.*, $v \in \mathbb{R}^3$, with $v = \delta_2 = (0\ 1\ 0)^t$. Note as the notation δ_2 does not contain any reference to the dimension, due to any lack of ambiguity, since here $\delta_2 \in \mathbb{R}^3$; this notation means that each component in a position different from 2 has value 0, while the component in position 2 has value 1.

The last definition we are going to introduce is a particular restriction of the general concept of maps between spaces, and a common notation in functional programming.

Definition 5 (Map) *The* map *is a function with domain a pair constituted by a function and a vector space, and codomain a second vector space:*

$$\mathrm{map} : \mathbb{V}^* \times \mathbb{V} \to \mathbb{W}\,, \tag{4}$$

which associates to each component of a vector, the result of the application of a given function to the component itself:

$$\mathrm{map}(f, v) = w, \quad w_i := f(v_i)\,,$$

with $f : \mathbb{V} \to \mathbb{W}$, $v \in \mathbb{V}$, and $w \in \mathbb{W}$.

The *map* definition will be exemplified by the following:

Example 3 *Let us consider a vectors $v \in \mathbb{R}^3$, with $v = (0\ \ 1/2\,\pi\ \ \pi)^t$. The* map *of the* sin *function on v results in*

$$\mathrm{map}(\sin, v) = (\sin(0)\ \ \sin(1/2\,\pi)\ \ \sin(\pi))^t = \begin{pmatrix} 0 & 1 & 0 \end{pmatrix}^t\,.$$

4 RFID data Modeling

This section is devoted to the definition of a general model capable of representing all aspects of a given supply chain.

Our overall objective is to give a rigorous definition of a supply chain, along with few significant properties, and show how such representation is mapped within a standard tensorial framework.

4.1 A General Model

Let us define the set $\mathcal{E}$ as the set of all EPCs, with $\mathcal{E}$ being finite. A *property* of an EPC is defined as an application $\pi : \mathcal{E} \to \Pi$, where Π represents a suitable property codomain.

Therefore, we define the application of a property $\pi(e) := \langle \pi, e \rangle$, *i.e.*, a property related to an EPC $e \in \mathcal{E}$ is defined by means of the pairing EPC-property; a property is a surjective mapping between an EPC and its corresponding property value.

A *supply chain* is defined as the product set of all EPCs, and all the associated properties. Formally, let us introduce the family of properties π_i, $i = 1, \ldots, k + d < \infty$, and their corresponding sets Π_i; we may therefore model a supply chain as the product set

$$\mathcal{S} = \mathcal{E} \times \Pi_1 \times \ldots \times \Pi_{k-1} \times \Pi_k \times \ldots \times \Pi_{k+d} \,. \tag{5}$$

We highlight the indices employed in the definition of the supply chain. The reader should notice as we divided explicitly the first k spaces $\mathcal{E}, \Pi_1, \ldots, \Pi_{k-1}$, from the remaining ones. As a matter of fact, properties may be split into two different categories: *countable* and *uncountable* ones.

By definition, a set A is *countable* if there exists a function $f : A \to \mathbb{N}$, with f being injective; for example, every subset of natural numbers $U \subseteq \mathbb{N}$ is countable (possibly infinite), and relative and rational sets, $\mathbb{Z}$ and $\mathbb{Q}$, respectively, are countable.

4.2 Properties

In the following we will focus on some of the properties related to EPCs, *i.e.*, we will model some codomains Π and their associated features.

Location Let us briefly model the *location* associated to an EPC. It is common to employ a GPS system in order to track the position on earth, however, any fine-grained space is sufficient to our purposes. In particular, being the earth homeomorphic to a 3-sphere, any ordered triple of real numbers suffices, leading us to the following definition:

Definition 6 (Location) *Let $\mathcal{L}$ be the set of ordered tuples $\ell := (\ell_1, \ell_2, \ell_3)$, with $\ell_1, \ell_2, \ell_3 \in \mathbb{R}$. We name $\mathcal{L}$ as* location set*, ℓ_1, ℓ_2 and ℓ_3 as* location coordinates*.*

A commonly employed coordinate system is the GPS location, *i.e.*, latitude and longitude, with an additional altitude coordinate. However, in many cases the altitude has no influence from an applicative point of view, thus leading the location to be defined by a couple of real numbers, or better, $\ell = (\ell_{lat}, \ell_{long}, \ell_{alt} \equiv 0)$.

Some standard representation of locations abstract their geographical information, preferring to employ a simple mapping between locations and a subset of natural numbers, representing the *location identifier*. Such expression is in contrast with the previous definition with respect to the *countability* property: employing a coordinate system will make location an uncountable property, while electing an identifier lets the property set fall in the countable category.

Which representation is more suitable to an application, is a matter of choice with respect to the domain of the problem.

Time In order to model a temporal interval relative to a product (EPC), we resort to an ordered couple of elements from the ring of real numbers. This suffices to specify, *e.g.*, the *entry* and *exit* time of a product from a given location. With this point of view, we model time as follows:

Definition 7 (Time) *Let $\mathcal{T}$ be the set of ordered couples $\tau := (t_i, t_o)$, with $t_i, t_o \in \mathbb{R}$. We name $\mathcal{T}$ as* time set*, t_i and t_o as* incoming *and* outcoming timestamps*, respectively.*

We highlight the fact that time spaces need not to be modeled as real numbers: in fact, natural numbers may also be suitable within a particular context, *e.g.*, employing UNIX timestamps. However, our model aims at generality, and therefore the real ring is the most appropriate choice, being the natural numbers set a proper subset of the real numbers ring.

Definition 8 (Inner sum) *Let us define the* inner sum *of two time elements $\tau_1 = (t_i^1, t_o^1)$, $\tau_2 = (t_i^2, t_o^2)$ as the operator $\oplus : \mathcal{T} \times \mathcal{T} \to \mathcal{T}$:*

$$\tau_1 \oplus \tau_2 := \left(\min(t_i^1,\, t_i^2), \max(t_o^1,\, t_o^2) \right) .$$

With the above operator, we have that $(\mathcal{T}, \oplus)$ assumes the algebraic structure of an abelian group. Such operation allows us to rigorously model the "addition of products": the overall timeframe of two products is, in fact,

Definition 9 (Lifetime) *We define as* lifetime *the linear form $\lambda : \mathcal{T} \to \mathbb{R}$ defined as follows:*

$$\langle \lambda,\, \tau \rangle := t_o - t_i \,, \quad \tau = (t_i, t_o) \in \mathcal{T} \,.$$

Due to the linearity, we are allowed to construct equivalence classes in $\mathcal{T}$ as follows:

$$[\widetilde{\tau}] := \{ \tau \in \mathcal{T} : \langle \lambda,\, \tau \rangle = \langle \lambda,\, \widetilde{\tau} \rangle \} \,. \tag{6}$$

The *canonical representative elements* of the above equivalence classes are defined as $(0, t_o)$, with $t_o \in \mathbb{R}$.

Definition 10 (Admissible time) *Given a time element $\tau \in \mathcal{T}$, we say that $\tau = (t_i, t_o)$ is* admissible *iff*

$$\langle \lambda,\, \tau \rangle > 0, \;\; \textit{with } t_i, t_o > 0 \,.$$

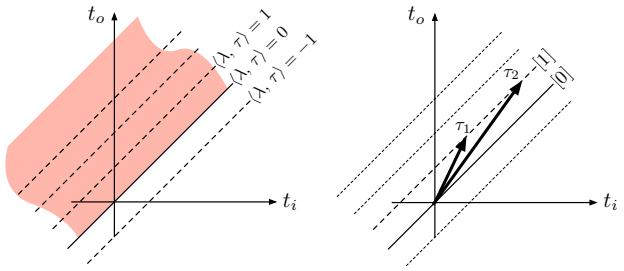
Both admissibility subspace, as well as few equivalence classes, are pictured in Figure 1.

4.3 Tensorial Representation

Let us now introduce a formal *tensorial framework* capable of grasping all properties related to a supply chain, as proposed in Section 4.1.

As previously outlined, we divide properties into two categories, *countable* and *uncountable* spaces. This separation allows us to represent countable spaces with natural numbers, therefore mapping their product space to $\mathbb{N}^k$, while leaving the product space of all uncountable properties into a collective space $\mathbb{U}$:

$$\mathcal{S} = \underbrace{\mathcal{E} \times \Pi_1 \times \ldots \times \Pi_{k-1}}_{\mathbb{N}^k} \times \underbrace{\Pi_k \times \ldots \times \Pi_{k+d}}_{\mathbb{U}} \,. \tag{7}$$



(a) (b)

Figure 1: The admissible time $\langle \lambda, \tau \rangle > 0$ region pictured in red is shown in (a); in (b) the two times τ_1 and τ_2 belong to the same equivalence class, *i.e.*, [1].

Such mapping will therefore introduce a family of injective functions called *indexes*, defined as:

$$\mathrm{idx}_i : \Pi_i \longrightarrow \mathbb{N}\,, \qquad i = 1, \ldots, k-1\,. \tag{8}$$

When considering the set $\mathcal{E}$, we additionally define a supplemental index, the *EPC index function* $\mathrm{idx}_0 : \mathcal{E} \to \mathbb{N}$, consequently completing the map of all countable sets of a supply chain $\mathcal{S}$ to natural numbers.

It should be hence straightforward to recognize the tensorial representation of a supply chain:

Definition 11 (Tensorial Representation) *The* tensorial representation *of a supply chain $\mathcal{S}$, as introduced in equation* (5)*, with countability mapping as in* (7) *is a multilinear form*

$$\Sigma : \mathbb{N}^k \longrightarrow \mathbb{U}\,. \tag{9}$$

A supply chain can be therefore rigorously denoted as a rank-k tensor with values in $\mathbb{U}$, mapping countable to uncountable product space.

4.4 Implementation

Preliminary to describing an implementation of our supply chain model, based on tensorial algebra, we pose our attention on the practical nature of a supply chain.

Our treatment is general, representing a supply chain with a tensor, *i.e.*, with a multidimensional matrix. Several other approaches are available in literature on the same topic, with a graph being the preferred supply chain model [6]. These models are thoroughly represented by our tensorial approach, due to the well known mapping between graphs and matrices (cf. [3], [4], and §1.2 of [7]).

However, a matrix-based representation need not to be *complete*. Supply chain rarely exhibit completion: practical evidence [6] suggests that products, identified by their EPC, for example, seldom present themselves in every location. The same consideration applies also to other properties, in particular, to countable properties.

Hence, our matrix effectively requires to store only the information regarding connected nodes in the graph: as a consequence, we are considering sparse matrices [5], *i.e.*, matrices storing only non-zero elements.

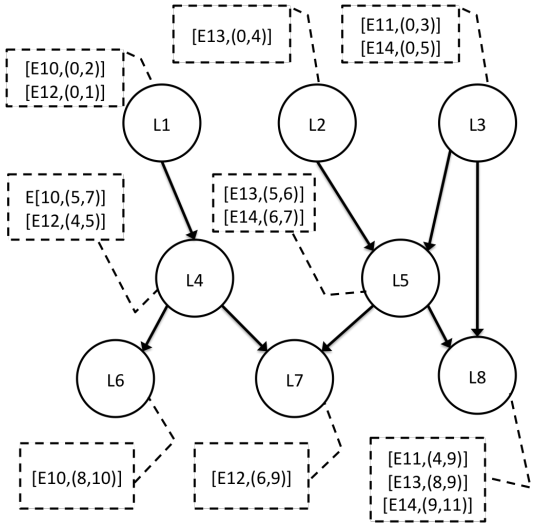


Figure 2: An example supply chain represented by a directed graph. Callouts represent lists of tuple constituted by an EPC with the associated time (t_i, t_o).

Notation A sparse matrix may be indicated with different notations (cf. [17] and [5]), however, for simplicity's sake, we adopt the *tuple notation*. This particular notation declares the components of a tensor $M_{i_1 i_2 \ldots i_k}(\mathbb{U})$ in the form of

$$\mathcal{M} = \Big\{ \{i_1 i_2 \ldots i_k\} \to u \neq 0 \, , u \in \mathbb{U} \Big\} , \tag{10}$$

where we implicitly intended $u \neq 0 \equiv 0_{\mathbb{U}}$. As a clarifying example, consider a vector $\delta_4 \in \mathbb{R}^5$: its sparse representation will be therefore constituted by a single tuple of one component with value 1, *i.e.*, $\{ \{4\} \to 1 \}$.

Example 4 *Let us consider the supply chain pictured in Figure 2, described tensorially by* $\Sigma : \mathbb{N}^2 \to \mathcal{T}$*, whose representative matrix is as follows:*

$$\begin{pmatrix} (0,2) & \cdot & \cdot & (5,7) & \cdot & (8,10) & \cdot & \cdot \\ \cdot & \cdot & (0,3) & \cdot & \cdot & \cdot & \cdot & (4,9) \\ (0,1) & \cdot & \cdot & (4,5) & \cdot & \cdot & (6,9) & \cdot \\ \cdot & (0,4) & \cdot & \cdot & (5,6) & \cdot & \cdot & (8,9) \\ \cdot & \cdot & (0,5) & \cdot & (6,7) & \cdot & \cdot & (9,11) \end{pmatrix}$$

where, for typographical simplicity, we omitted $0_{\mathcal{T}} = (0,0)$*, denoted with a dot.*

For clarity's sake, we outline the fact that Σ *is a rank-2 tensor with dimensions 5 (i.e. the rows), 8 (i.e. the columns). In fact, the EPC and locations sets are, respectively*

$$\begin{aligned} \mathcal{E} &= \{E10, E11, E12, E13, E14\} \, , \\ \Pi_1 &= \{L1, L2, L3, L4, L5, L6, L7, L8\} \, . \end{aligned}$$

where $\mathrm{idx}_0(E10) = 1$*,* $\mathrm{idx}_0(E11) = 2$*, ...,* $\mathrm{idx}_0(E14) = 5$*, and similarly* $\mathrm{idx}_1(L1) = 1$*, ...,* $\mathrm{idx}_1(L8) = 8$*. In the sparse representation we have* $\{\{1,1\} \to (0,2), \{1,4\} \to (5,7), \ldots, \{5,8\} \to (9,11)\}$.

Remarks The reader noticed that all properties regarding spaces and their correlated operations, as defined in Sections 3 and 4, exhibit fundamental algebraic properties: *associativity*, *commutativity*, and *distributivity*. These characteristics help us in clarifying one implementative aspect regarding our sparse approach: memory constraints.

At first sight, in order to perform operations such as tensor application, we may need to load the whole matrix and afterwards execute the required computation. Due to the distributivity of products with respect to summations, we actually may *split* a matrix in sub-matrices, carry out the application and finally, thanks to the associativity, add the partial results obtaining the correct solution.

For example, if $A \in \mathbb{M}_{22}(\mathbb{R})$, and $v \in \mathbb{R}^2$ being a suitable vector, the matrix can be divided in two (or more) sub-matrices, thus giving the ability to perform computations independently:

$$Av = (A' + A'')v = A'v + A''v\,. \tag{11}$$

The above equation shows how the *sparsity* of a matrix—*i.e.*, the ratio between the overall dimensions and the actual non-zero stored elements—alleviates problems caused by the finiteness of computer memories. When performing $A'v$, we actually do not need any information regarding the subsequent A'' matrix and vice versa, and as a result, we may not store its elements in memory. Additionally, multiplication of a sparse matrix with a vector has a computational complexity that depends on the sparsity of the input: generally, very sparse matrices as the ones resulting from supply chains, yield *sub-quadratic* (cf. [5, 7]), and even *quasi-linear* asymptotic complexity [10].

On a final note, due to the above mentioned properties, we are able to carry out the required computations with high memory constraints, such as on mobile devices. Moreover, when dealing with massive amounts of data, we are capable of exploit recent parallel and distributed technologies, remembering that tensor applications may be subdivided into sub-blocks, and processed independently.

5 RFID data Analysis

Based on our conceptual framework, in this section we will provide a method to analyze RFID data represented by a tensor.

With reference to the example provided in Section 4, in the following we simplify our notation, employing two countable sets, EPC $\mathcal{E}$ and location $\mathcal{L}$ indexes, in that order, and the uncountable time set $\mathcal{T}$, *i.e.*, a matrix M_{ij}: i referring to EPCs, and j being the index of locations.

Such choice of attributes introduce no limitations on the generality of our approach, due to the fact that, in essence, applying a tensor to a vector means summing all the components sharing an index, regardless of ranks and dimensions. We will show how our approach allows to process both real-time and off-line analysis in a uniform way.

5.1 Monitoring Query

Usually real-time analysis on a supply chain should monitor the (correct) traversing of products. For instance, we would control if RFID readers are producing incorrect data, *e.g.*, $t_i > t_o$, that is if an object leaves a location before entering in it, or for example, if

the transport of products has an unexpected delay, or any property exceeds a threshold value. We call such analysis a *monitoring query*.

In this case we have to perform quick decentralized on-line processing, *i.e.*, different data sources send local elaborations to be integrated. As we discussed in Section 4, and subsequently proved in Section 6, our framework is able to quickly import RFID data, perform monitoring query efficiently, and finally, thanks to the associativity of the operations, add the partial results obtaining the correct solution.

In this paper we provide two typical examples of monitoring query on the time dimension. First of all, we introduce the *admissibility test*, selecting all EPCs whose time is not admissible (cf. Definition 10 in Section 4):

$$\mathrm{map}(\, \langle \lambda, \cdot \rangle \, < 0, M_{ij}) \,. \tag{12}$$

The reader should notice a shorthand notation for an implicit boolean function: as for all descendant of the C programming language, such definition yields 1 if the condition is met, *i.e.*, the time associated to an element of the tensor is not valid, 0 otherwise. Furthermore, we recall the fact that being M_{ij} represented as a sparse matrix, the application of our condition is well defined, being applicable to the stored values, *i.e.*, to the time couple $(t_i, t_o) \in \mathcal{T}$.

A second analysis is the *delay test*, selecting all EPCs whose time surpasses an expected interval. In our framework, given two times τ_1 and τ_2, *i.e.*the bounds of the interval, we would return all EPCs whose time τ excesses the inner sum $\tau_3 = \tau_1 \oplus \tau_2$ (cf. Definition 8 in Section 4):

$$\mathrm{map}(\, (t_i(\cdot) > t_i(\tau_3)\,) \wedge (t_o(\cdot) < t_o(\tau_3)\,)\,, M_{ij}) \,. \tag{13}$$

In the same way, it is straightforward to extend similar analysis on other properties of a product.

5.2 Tracking Query

A *tracking query* finds the movement history for a given tag identifier $e \in \mathcal{E}$. We can comfortably perform the query efficiently, using the model described in Section 4, by applying the tensor application.

Therefore, given $i = \mathrm{idx}(e)$, we build a Kroneker vector as a vector δ_i, with $|\delta_i| = |\mathcal{E}|$, and finally apply of the rank-2 tensor represented by M_{ij} to δ_i, *i.e.*:

$$r = M_{ij}\delta_i \,.$$

For instance, referring to the example pictured in Figure 2, let us consider the tag $E13$, we have $i = \mathrm{idx}(E13) = 4$, and therefore our vector will be $\delta_4 = \{\{4\} \to 1\}$. Consequently, the resulting vector will be $r = M_{ij}\delta_4 = \{\{2\} \to (0,4),\ \{5\} \to (5,6),\ \{8\} \to (8,9)\}$, or in another notation, $L2 \to L5 \to L8$

5.3 Path Oriented Query

A *path oriented query* returns the set of tag identifiers that satisfy different conditions. Following the query templates given in [16], we subdivide path oriented queries into two main categories: *path oriented retrieval* and *path oriented aggregate* queries.

The former returns all tags covering a path satisfying given conditions, while the latter computes an aggregate value. It is possible to formulate a grammar for these queries, similar to XPath expressions; in particular, path oriented retrieval and path oriented aggregate queries are respectively indicated as follows:

$$L_1[\text{cond}_1]//L_2[\text{cond}_2]//\ldots//L_n[\text{cond}_n]\,, \tag{14a}$$

$$L_1[\text{cond}_1]/L_2[\text{cond}_2]/\ldots/L_n[\text{cond}_n]\,, \tag{14b}$$

where $L_1, \ldots L_n$ is the *path condition*, *i.e.*, the sequence of locations covered by the tag, and cond_i is the *info condition*; for more information about such expressions, we refer the reader to [16]. A path condition expresses *parenthood* between locations, indicated as L_i/L_j, or *ancestry* with $L_i//L_j$; an info condition, on the other hand, indicates conditions on tag properties, *e.g.*, $StartTime$ and $EndTime$.

Path Oriented Retrieval In our framework, a path oriented retrieval query as in (14a) is easily performed by exploiting the tensor application coupled with the Hadamard product. Given the location set $\mathcal{L}$, for each $z = \text{idx}(L_k)$, with $k = 1, \ldots, n$, we create a Kroneker vector δ_z and subsequently apply the rank-2 tensor, resulting in a set of vectors $r_z = M_{ij}\delta_j$, where for typographical reasons, we dropped the subscript intending $\delta \equiv \delta_z$.

Finally, we apply the condition function to each r_z employing the map operator. This yields a set of vectors

$$\widetilde{r_z} = \text{map}(\text{cond}_z, r_z)\,, \qquad \text{cond}_z : \mathbb{N}^k \times \mathbb{U} \longrightarrow \mathbb{F}\,,$$

whose Hadamard multiplication generates the final result:

$$\bar{r} = \widetilde{r_1} \circ \ldots \circ \widetilde{r_n}\,,$$

reminding the reader that only non-zero values are stored, and therefore given as a result of a computation. The cond functions, as indicated above, are maps between properties of supply chains and a suitable space $\mathbb{F}$, *e.g.*, natural numbers for a boolean result.

For an example, referring to Figure 2, let us consider the query

$$L3[StartTime > 0]//L8[EndTime - StartTime < 4]\,.$$

In this case, given $\text{idx}(L3) = 3$ and $\text{idx}(L8) = 8$, we build δ_3 and δ_8, and generate the partial results $r_3 = M_{ij}\delta_3 = \{\{2\} \to (0,3), \{5\} \to (0,5)\}$ and $r_8 = M_{ij}\delta_8 = \{\{2\} \to (4,9), \{4\} \to (8,9), \{5\} \to (9,11)\}$, where evidently the application was performed along the second dimension, *i.e.*, for each $\delta \in \{\delta_3, \delta_8\}$, we compute $M_{ij}\delta_j$. Finally, subsequent to mapping conditions on the results, in this case $\text{cond}_3 = t_i(\cdot) > 0$ and $\text{cond}_8 = \langle\lambda, \cdot\rangle < 4$, we obtain the correct outcome $\bar{r} = \widetilde{r_3} \circ \widetilde{r_8} = \{\{5\} \to (9,11)\}$, *i.e.*, $E14$.

Considering parenthood as in (14b) instead of ancestry, *i.e.*, L_i/L_j, we briefly sketch the fact that such query does not, in fact, differ from the above, except for one particular: each resulting EPC, when subject to a tracking query, produces a sparse vector whose *length*, *i.e.*, the number of non-zero stored elements, is exactly equal to the number of locations under analysis.

Path Oriented Aggregate Path oriented aggregate queries may be represented as $\langle f, Q\rangle =: f(Q)$ where f is an *aggregate function*, *e.g.*, average or minimum, and Q is the result of a path oriented retrieval query.

Therefore, let $\bar{r_Q}$ be the result of Q, and let f be a function defined on vectors of supply chain elements, we simply have that a path aggregate may be expressed as $f(\bar{r_Q})$.

Referring to Figure 2, let us consider the query expressed in the grammar of [16]

$$\langle AVG[L8.StartTime], //L8\rangle\,,$$

it is easily performed by applying the function $f := \text{average}(t_i)$ to the outcomes of the path oriented retrieval query on $L8$, resulting in $\bar{r} = \text{average}(\{4, 8, 9\}) = 7$.

6 Experiments

We performed a series of experiments aimed at evaluating the performance of our approach, reporting the main results in the present section.

Environment Our benchmarking system is a dual core 2.66GHz Intel with 2 GB of main memory running on Linux, where we implemented our framework in C++ within the Mathematica 8.0[2] computational environment. Our results have been compared to the ones from the approach in [16], tested against a generated synthetic RFID data in terms of stay records, and considering products moving together in small groups or individually, a behavior called *IData* in [16]: data production followed the same guidelines on a supply chain of 100 locations. The complete data set comprises 10^5, $5 \cdot 10^5$, 10^6, $5 \cdot 10^6$, and 10^7 stay records. In the following, we will denote our tensorial approach T, while the proposed one in [16] with P.

Results Performances have been measured with respect to data loading and querying. Referring to the former, the main advantage of our approach is that we are able to perform loading without any particular relational schema, when compared to P, where a schema coupled with appropriate indexes have to be maintained by the system. In this case, loading execution times are 0.9, 11, and 113 seconds, for sets of 10^5, 10^6, and 10^7 stay records, respectively; on the contrary, P timings were in order of minutes and hours. Another significant advantage of T relies in memory consumption: we need 13, 184, and 1450 MB to import the above mentioned sets; as a side-note, we highlight the fact that the 10^7 set required a division in smaller blocks, *e.g.*, 10^6, due to the limited memory at disposal.

With respect to query execution, T presents a similar behavior and advantages with respect to P, for both time and memory consumption. We performed *cold-cache* experiments, *i.e.*, dropping all file-system caches before restarting the systems and running the queries, and repeated all the tests three times, reporting the average execution time. As in [16], we formulated 12 queries to test the two systems as reported in Table 1 in the Appendix, due to space constraints. In brief, Q1 is a tracking query, Q2 to Q5 are path oriented retrieval queries, while Q6 to Q12 are path oriented aggregate queries. Due to the nature of P, we were able to perform a comparison only on centralized off-line massive analysis.

[2] `http://www.wolfram.com/mathematica/`

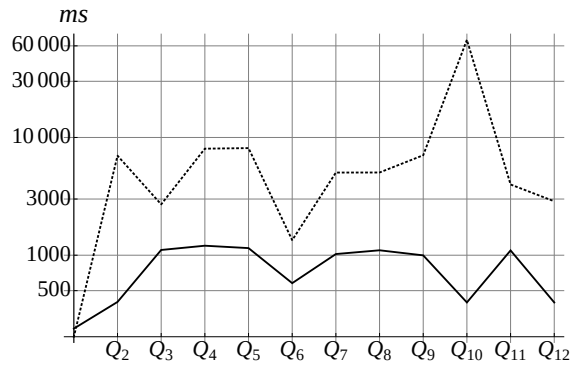


Figure 3: Query execution time in logarithmic scale for 10^7 stay records: the solid line refers to T, while dotted to P.

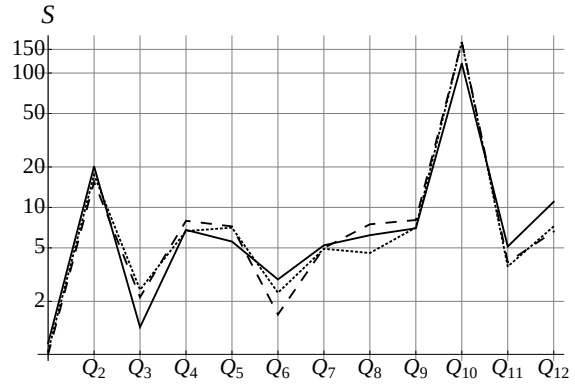


Figure 4: Speed-up logarithmic graph for all queries with dataset sizes of 10^5 (solid), 10^6 (dashed), and 10^7 (dotted).

Figure 3 shows the query performance times in milliseconds with 10^7 stay records; the Q1 label is omitted due to typographical reasons. Both T and P present similar performance to execute tracking queries, *i.e.*, Q1 in the query set. With path oriented queries, as indicated in the plot, T is faster than P: while T executes such queries again as tensor applications, P has to perform SQL queries with different complexities, such as *join operations* for info conditions or aggregations; on average, T executes in the range $[500, 1000]\, msec$, while P in $[3000, 10000]\, msec$. In the Appendix, the reader finds detailed diagrams, showing query performances according to the number of stay records for each of the twelve queries.

A more significant result is the *speed-up* between the two approaches, as shown in Figure 4; again, Q1 label is dropped due to typographical causes. We computed the speed-up for all data sets as the ratio between the execution time of P, and that of our approach T, or briefly $S = {}^{t_P}/_{t_T}$. In general, T performs very well with respect to P in any dataset, particularly for queries related to the object transition, *e.g.*, Q2, Q4, Q5, Q10. The query performance of T is on the average 19 times better than that of P, 150 times on the maximum, *i.e.*, Q10.

Another strong point of T is a very low consumption of memory, due to the *sparse matrix* representation of tensors and vectors. Figure 5 illustrates the main memory consumption of each query with respect to 10^5, 10^6, and 10^7 stay records. On the average, tracking queries require very few bytes of memory for any dataset, path oriented queries

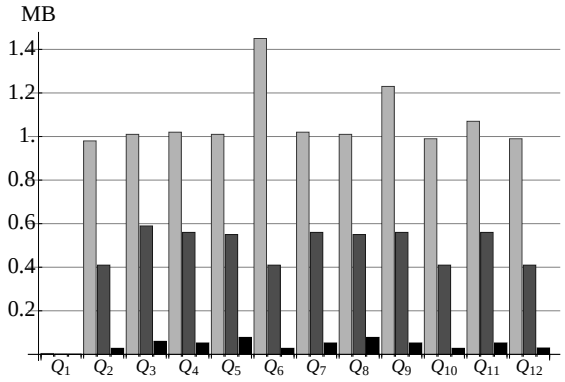


Figure 5: Main memory consumption for each query: black bars refer to 10^5, dark gray to 10^6, and light gray to 10^7 data size.

few KBytes, topping 1 MB for 10^7. Results demonstrate how our approach can be used in a wide range of applications, where devices with limited calculus resources may process large amount of data in an efficient and effective way.

7 Conclusion and Future Work

We have presented an abstract algebraic framework for the efficient and effective analysis of RFID data in supply chain management. Our approach leverages tensorial calculus, proposing a general model that exhibits a great flexibility with multidimensional queries, at diverse granularity and complexity levels. Experimental results proved our method efficient when compared to recent approaches, yielding the requested outcomes in memory constrained architectures. For future developments we are investigating the introduction of reasoning capabilities, along with a thorough deployment in highly distributed Grid environments. In addition, we are about to test our model on mobile devices, comprising more complex properties and queries.

References

[1] R. Abraham, J. E. Marsden, and T. Ratiu. *Manifolds, Tensor Analysis, and Applications*. Springer, 1988.

[2] Y. Bai, F. Wang, P. Liu, C. Zaniolo, and S. Liu. Rfid data processing with a data stream query language. In *Proceedings of the 23rd Int. Conf. on Data Engineering (ICDE)*, pages 1184–1193, 2007.

[3] A. Bondy and U. S. R. Murty. *Graph Theory*. Graduate Texts in Mathematics. Springer, 2010.

[4] G. Chartrand. *Introductory Graph Theory*. Dover Publications, unabridged edition edition, 1984.

[5] T. A. Davis. *Direct Methods for Sparse Linear Systems*. SIAM, 2006.

[6] R. Derakhshan, M. E. Orlowska, and X. Li. Rfid data management: Challenges and opportunities. In *Proc. of the IEEE Int. Conf. on RFID (RFID)*, pages 175–182, 2007.

[7] I. S. Duff, A. M. Erisman, and J. K. Reid. *Direct Methods for Sparse Matrices*. Numerical Mathematics and Scientific Computation. Oxford Univ. Press, 1989.

[8] EPC Global. http://www.epcglobalinc.org/home.

[9] H. Gonzalez, J. Han, X. Li, and D. Klabjan. Warehousing and analyzing massive rfid data sets. In *Proceedings of the 22nd Int. Conf. on Data Engineering (ICDE)*, page 83, 2006.

[10] F. G. Gustavson. Two fast algorithms for sparse matrices: Multiplication and permuted transposition. *ACM Trans. Math. Softw.*, 4:250–269, 1978.

[11] J. H. Heinbockel. *Introduction to Tensor Calculus and Continuum Mechanics*. Trafford Publishing, 2001.

[12] K. M. Hoffman and R. Kunze. *Linear Algebra*. Prentice Hall, 1971.

[13] B. Jancewicz. The extended grassmann algebra of $\mathbb{R}^3$. In W. E. Baylis, editor, *Clifford (geometric) algebras with applications to physics, mathematics, and engineering*. Birkhäuser Boston, 1996.

[14] S. R. Jeffery, G. Alonso, M. J. Franklin, W. Hong, and J. Widom. A pipelined framework for online cleaning of sensor data streams. In *Proceedings of the 22nd Int. Conf. on Data Engineering (ICDE)*, page 140, 2006.

[15] S. R. Jeffery, M. N. Garofalakis, and M. J. Franklin. Adaptive cleaning for rfid data streams. In *Proceedings of the 32nd Int. Conf. on Very Large Data Bases (VLDB)*, pages 163–174, 2006.

[16] C.-H. Lee and C.-W. Chung. Efficient storage scheme and query processing for supply chain management using rfid. In *Proc. of the Int. Conf. on Management of Data (SIGMOD)*, pages 291–302, 2008.

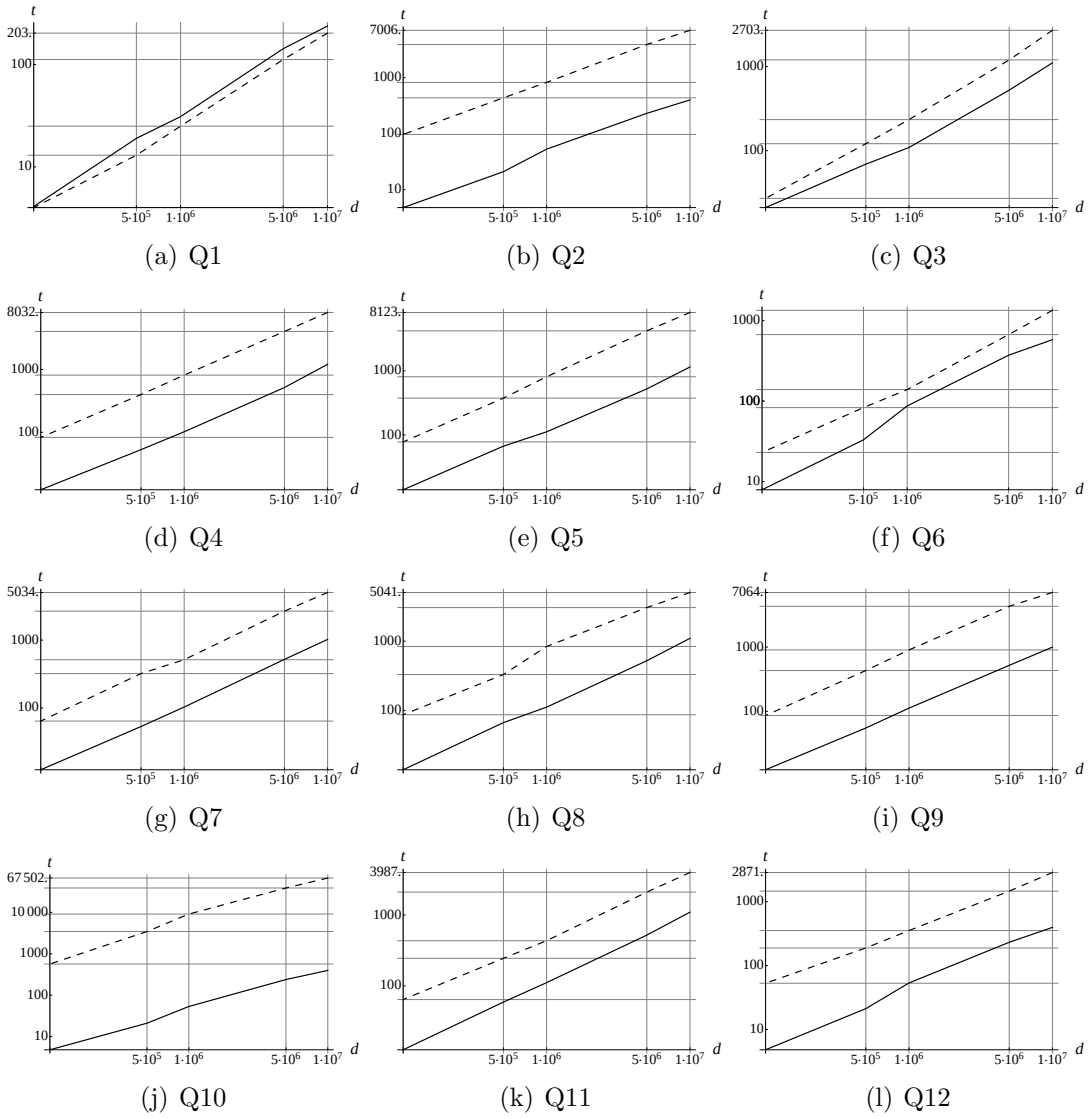
[17] O. Osterby and Z. Zlatev. *Direct Methods for Sparse Matrices*. Lecture Notes in Comp. Sci. Springer, 1983.

[18] S. Papadimitriou, J. Sun, and C. Faloutsos. Streaming pattern discovery in multiple time-series. In *Proceedings of the 31st Int. Conf. on Very Large Data Bases (VLDB)*, pages 697–708, 2005.

[19] F. Wang, S. Liu, P. Liu, and Y. Bai. Bridging physical and virtual worlds: Complex event processing for rfid data streams. In *10th Int. Conf. on Extending Database Technology (EDBT)*, pages 588–607, 2006.

A Experiments

Using the XPath-like syntax of [16], we formulated 12 queries as shown in Table 1. Figure 6 reports all timing benchmarks for queries Q1 to Q12, for all test data sets. For typographical reasons, the first dataset label 10^5 is omitted.



(a) Q1 (b) Q2 (c) Q3

(d) Q4 (e) Q5 (f) Q6

(g) Q7 (h) Q8 (i) Q9

(j) Q10 (k) Q11 (l) Q12

Figure 6: Query performances for all data sets as indicated in Section 6 for Q1 (a) to Q12 (l); query execution time (t) in *msec* is plotted, in logarithmic scale, against data set size (d). Dashed graphs refers to P, while solid ones to T.

Table 1: Test Queries

Q1	EPC = 17281
Q2	//L628
Q3	/L03/L16/L29/L321/L422
Q4	//L16//L322//L628
Q5	//L16//L322[StartTime<200]//L628
Q6	COUNT(), //L628
Q7	COUNT(), //L16//L322//L628
Q8	COUNT(), //L16//L322[StartTime<200]//L628
Q9	AVG(L16.EndTime - L16.StartTime), //L16//L322//L628
Q10	AVG(L628.StartTime), //L628
Q11	MIN(L16.EndTime - L16.StartTime), //L16//L322//L628
Q12	MIN(L628.StartTime), //L628