Universidad Autónoma de Madrid

Escuela Politécnica Superior

22|23

Bachelor thesis

Time-aware flexible skyline queries

Jaime Pons Garrido

Escuela Politécnica Superior
Universidad Autónoma de Madrid
C\Francisco Tomás y Valiente nº 11

Campus Internacional excelencia UAM CSIC+

www.uam.es

UNIVERSIDAD AUTÓNOMA DE MADRID
ESCUELA POLITÉCNICA SUPERIOR

Bachelor as Ingeniería Informática

BACHELOR THESIS

Time-aware flexible skyline queries

Author: Jaime Pons Garrido
Advisor: Simone Santini

junio 2023

Francisco Tomás y Valiente, n^o 1
Madrid, 28049
Spain

Jaime Pons Garrido
Time-aware flexible skyline queries

Jaime Pons Garrido
C\ Francisco Tomás y Valiente N$^{\underline{o}}$ 11

PRINTED IN SPAIN

When you reach the end of your rope, tie a knot in it and hang on.

Franklin D. Roosevelt

Resumen

El uso de internet y de las nuevas tecnologías se ha incrementado significativamente durante los últimos años. En un mundo donde cada vez estamos más conectados a través de internet, se producen grandes cantidades de datos como una huella de nuestras actividades. Ha quedado patente que el análisis de estos es de vital importancia para todo tipo de actores sociales, desde empresas hasta gobiernos. En este contexto, surgen ciertos obstáculos a superar para procesar datos de forma eficiente. En primer lugar, el enorme volumen de datos, que dificulta la detección de tendencias e incrementa los costes computacionales. Por otra parte, los datos en sí mismos vienen dotados de un contexto que es mucho más difícil de procesar que la información explícita que contienen y un elemento clave dentro de dicho contexto es el tiempo.

Las conocidas como *skyline queries* han quedado contrastadas como un método eficiente para el filtrado de bases de datos de gran tamaño. No obstante, su uso ha caído porque carecen de características que son necesarias para el análisis de datos actual como la expresión de preferencias o el control del cardinal del set filtrado. Para solventar estos problemas, se han desarrollado operadores más complejos que combinan las *skyline queries* con otros métodos de filtrado como las *ranking queries*. A esta combinación se la denomina como *flexible skyline* y solventa muchos de los problemas presentados por el operador original, aunque sigue sin ser capaz de expresar el contexto temporal de la información.

El objetivo principal de este trabajo es realizar un estudio de las *flexible skylines* y proponer una modificación del operador que permita considerar el contexto temporal de la información. Esta solución no solo se proporcionará a nivel teórico, sino que se implementará en un sistema de procesamiento de datos. Este incorporará un modelo de detección de tópicos para procesar la información en bruto con el propósito de generar distribuciones de tópicos sobre las que realizar *queries* con filtrado temporal.

Palabras clave

Flexible Skylines, Ranking Queries, Time Series, Latend Dirichlet Allocation, Topics Over Time, Author-Topic Model

Abstract

The use of the Internet and new technologies has significantly increased in recent years. In a world where we are increasingly connected through the internet, large amounts of data are produced as a footprint of our activities. It has become clear that analyzing this data is of vital importance for all types of social actors, from companies to governments. In this context, certain obstacles arise to efficiently processing data. First, the enormous volume of produced data makes it difficult to detect trends and increases computational costs. On the other hand, the data itself is endowed with a context that is much more difficult to process than the explicit information that data contains, and a key element within the information context is time.

The so-called skyline queries have been characterized as an efficient method for filtering large databases. However, their use has declined because they lack features that are necessary for current data analysis, such as expression power or cardinality control. To solve these problems, more complex operators have been developed in recent years, combining skyline queries with other filtering methods, such as ranking queries. This combination is denominated as flexible skyline and solves many of the problems presented by the original operator. However, they still cannot express the temporal context of information.

The main objective of this thesis is to study flexible skylines as a data filtering method and propose a modification of the operator that allows to consider the temporal context of information. This solution will not only be provided at a theoretical level, but it will also be implemented in a data processing system that will incorporate a topic detection model to process raw information in order to produce topic distributions that will be used as data to generate time-aware skyline queries.

Keywords

Flexible Skylines, Ranking Queries, Time Series, Latend Dirichlet Allocation, Topics Over Time, Author-Topic Model

Table of Contents

Lists

List of algorithms

List of codes

List of equations

List of figures

List of tables

1

Introduction

Big data has become an undeniable trend in the last decade, and its efficient analysis is a key factor to succeed in the current market. In a context of information overload, it has become more necessary than ever to design methods that allow us to extract relevant information from large databases.

Methods such as skyline queries have traditionally been used to filter databases, looking for elements of potential interest [1]. Nonetheless, given some of the limitations of this method, new updated query operators, based on skyline queries, have been created; that is the case of flexible skylines [2]. Despite the last advancements, many querying methodologies do not contemplate time as a potential factor to take into account, and others that consider it can be computationally exhaustive.

As technological services advance, more sophisticated criteria are used to decide whether data is of interest, and time is one of the most complex ones. It is crucial to understand how this factor influences the decision of whether information passes a certain filter and is shown to a user or not. A clear example would be news filtering, in which the time factor is paramount to decide whether a news is relevant or not at a given point in time.

This project aims to design an algorithm based on flexible skylines capable of leveraging time-related data, such as time series, in order to perform time-aware queries. To pursue this goal, an end-to-end test bed system will be designed and implemented. This system will be able to perform topic detection over news published in social media and use the obtained topic distributions to generate time-aware flexible skyline queries.

1.1. Motivation

Data analysis is a field in continuous growth within data science, and it is tightly connected to decision making processes. However, the sheer amount of data generated every day is an obstacle to obtaining valuable insights due to the difficulty of filtering relevant information, as mentioned in [3]. Thus, the techniques used to make decisions regarding the potential value of data are crucial to succeed in the field.

In this context, it is advantageous to have systems that are able to generate outputs, not only taking into account a series of static parameters but considering more complex ones, such as time. Nowadays, it is common to see systems that allow one to select a time range to filter data, however, this approach can turn out to be inefficient since relevant data can be discarded for not falling into the selected time range.

Skyline queries are a consolidated method that allows defining dominance relationships over a set of data and returns those elements that are not dominated by any other one. But they have evolved into more complex versions that allow to introduce flexible parameters to the dominance relationships. The new characteristics of these query operators open the door to new possibilities, such as developing more sophisticated time filtering methodologies.

1.2. Objectives

This thesis has the following objectives:

- Investigate different approaches to carry out query filtering based on the skyline operator. Provide the theoretical background required to understand the filtering process.
- Present a modified flexible skyline operator that allows to perform time-aware queries using time series.
- Adapt an existing flexible skylines algorithm to implement the new time-aware operator while achieving an acceptable time complexity.
- Study different approaches to perform topic detection over texts extracted from social media news.
- Develop a test bed system capable of performing bulk topic detection over documents, store the results in a database and and generate time-aware flexible skyline queries over the results.

2

State of the art

This section focuses on the various theoretical concepts required to understand skyline queries and their evolution to more complex operators. Different cases of tuple dominance relationships will be explained, and the concept of flexible skyline queries will be introduced. In addition, two different approaches to performing skyline queries using probability distributions and time series will be introduced and compared.

An overview of different methodologies to perform topic detection over a collection of documents will be provided. This theoretical background will be used to design the topic detection module of the test bed system.

2.1. Skyline queries

Skyline queries are a relatively new concept in the field of database filtering. Proposed by Stephan Borzsony in 2001 [4], they constitute a powerful filtering tool, that allows one to filter large data sets using multiple factors obtaining the set of all the potentially interesting elements for the user. Excluding only those that are worse than other elements in all the factors considered.

The skyline operator is designed to return the tuples that are “better” than all the others in one attribute and “not worse” in the rest. Moving forward, smaller values will be assumed to be better. Please note that this is an arbitrary assumption and the opposite could be used as well.

Consider the following example: A travel agency is requested to find a hotel based on two criteria. First, the hotel must be located close to the beach, and second, it must be cheap. Note that these are usually conflicting requests. The hotel database is unable to return a single element since there are always tradeoffs, thus it is the client who should decide. However, it is possible to return a set of hotels that are not worse than any other in any of the two criteria, this is commonly known as the *Skyline*.

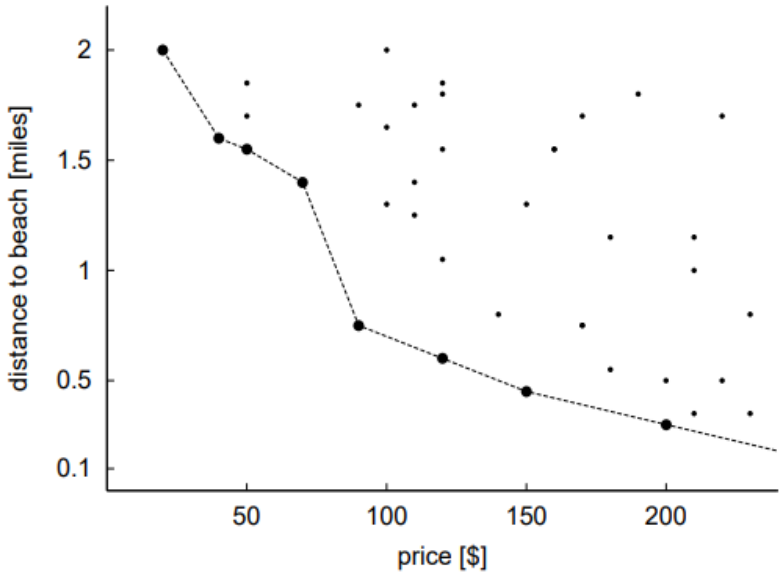


Figure 2.1: Example of skyline set, obtained from [4].

Definition 1. *Let $t_1, t_2 \in \mathcal{DB}$ be two tuples in the database. Each tuple in the database has the following attributes $\{a_1, \ldots, a_n\}$. Then t_1 is said to be dominated by t_2 ($t_2 \prec t_1$) if:*

$$\exists a_i \;\; s.a. \;\; t_2[a_i] < t_1[a_i] \wedge \forall a_j \;\; t_2[a_j] \leq t_1[a_j].$$

Given this definition of dominance, the skyline set is defined as follows:

Definition 2. *Let $\mathcal{DB}$ be a set of tuples in a database. Then $\mathcal{SKY}$ is defined as:*

$$\mathcal{SKY} = \{t_i \in \mathcal{DB} \mid \nexists\, t_j \in \mathcal{DB} : \;\; t_j \prec t_i\}. \tag{2.1}$$

From now on, we will refer to the set of non-dominated elements or *skyline* as $\mathcal{SKY}$. An interesting property of the skyline operator is that it is a monotone operator, which means that for any monotone scoring function $f : \mathcal{DB} \Rightarrow \mathbb{R}$, where $\mathcal{DB}$ is the set of tuples in the database, if $p \in \mathcal{DB}$ is the global maximum of f, then $p \in \mathcal{SKY}$. So, it does not really matter which tradeoffs one wants to make regarding personal preferences, the preferred option will always be in the $\mathcal{SKY}$ set.

Even though the skyline operator was implemented into databases, it had been previously studied as an abstract problem in the field of optimization. The problem of finding the maximum vector in a set, discussed in [5].

2.1.1. Properties of the skyline operator

Skyline queries have become a de facto solution for many data filtering problems due to some convenient properties [6]:

- **Reusability:** Once a skyline set has been obtained, the addition of new elements to the database turns out to be a reduced problem since they must only be matched against the elements in the skyline.
- $O(1)$ **additions:** Note that the $\mathcal{SKY}$ set is composed by non-dominated tuples that do not maintain any relationships between them, thus, there is no possible ordering, which makes additions to the set an $O(1)$ operation.

Finally, it is important to note that the skyline operator presents some undesirable properties.

- **Query cardinality:** Since the only condition required to belong to $\mathcal{SKY}$ is non-domination, the cardinality of the skyline set can be very high, which can lead to an increased computational cost.
- **Limited expression power:** It was previously mentioned that it does not matter the tradeoff chosen, the skyline will always contain the best option for each one. This implies that there is no way to tailor the query's output given a specific set of preferences.

In the following section, some modifications to the skyline operator will be presented, potentially solving the aforementioned problems.

2.1.2. Flexible skyline queries

As it was mentioned before, skyline queries present some tradeoffs that make them unsuitable for some use cases. In recent years, different approaches have been taken to overcome these problems. There are other types of operators capable of performing queries to solve similar issues, but that present other characteristics, two of the most relevant are:

- **Ranking operator (top-k):** [7] The ranking operator uses a weighted scoring function and returns the top k elements classified by their score. This reduces the multi-criteria problem to a single objective one since those weights in the scoring function represent the importance of each factor considered to rank the tuples. Including ordering in the set of results allows to impose cardinal control.
- **Lexicographic operator:** [8] This operator offers a more radical approach where criteria are in a fixed order of priority, and data is classified according to the most important criteria. If a tie is found, the next criterion is used and so on. Notice that even the slightest difference

in a parameter's value can lead to a different classification, which produces high levels of volatility in the results.

The second of the mentioned operators has very limited use. However, when looking at the first one, it can be noticed that it presents almost complementary characteristics to the skyline. A new operator, combining top-k and skyline queries was defined in [6] [2] [9]. This "mixed" operator aims to remove some of the tradeoffs imposed by the previous ones and has been named as **flexible skyline operator**, generating the so-called **flexible skylines ($\mathcal{F}-$Skylines)**.

Taking the skyline operator as a reference, the flexible skyline operator is again based on the concept of dominance. It defines a subset of weighted monotone scoring functions $\mathcal{F} \subset \mathcal{M}$, where $\mathcal{M}$ is the infinite set of all monotone scoring functions.

Definition 3. ***Monotone Scoring Function:*** *A monotone scoring function is a function $f : [0,1] \Rightarrow \mathbb{R}^+$ that assigns scores to tuples $t = (a_1, ..., a_d)$ and that satisfies the following property:*

$$\forall i \in \{1, ..., d\};\ t[A_i] \leq s[A_i] \Rightarrow f(t) \leq f(s). \tag{2.2}$$

As it will be seen later in this section, the flexible skyline operator is based on the use of monotone scoring functions to define dominance. Therefore, it is important to define $\mathcal{M}$ as the infinite space of all possible monotone scoring functions.

Flexible skylines have a particular definition of the $\mathcal{SKY}$ set that adapts the usual dominance concept of the skyline operator to the use of monotone scoring functions.

Definition 4. *Let $\mathcal{DB}$ be a set of tuples in a database. Then $\mathcal{SKY}$ is defined as:*

$$\mathcal{SKY} = \{t_i \in \mathcal{DB} \mid \exists f \in \mathcal{M}\ \forall s \in DB\ \ s \neq t \Rightarrow f(t) < f(s)\}. \tag{2.3}$$

Now that we have redefined the notion of skyline we will provide the definition of $\mathcal{F}-$dominance, which is the relationship used to perform flexible skyline queries. But first, it is important to define the subset $\mathcal{F}$.

Definition 5. *Let $\mathcal{F} \subset \mathcal{M}$ be a subset of monotone scoring functions, $f : [0,1] \longrightarrow \mathbb{R}^+$, then $\mathcal{F}$ is tuple distinguishing if:*

$$\forall t, s \in [0,1]^d.\ \ t \neq s \Rightarrow \exists f \in \mathcal{F}\ \ f(t) \neq f(s). \tag{2.4}$$

There are many possible infinite sets of functions $\mathcal{M}$ that can be used to generate $\mathcal{F}$, however, to ease computations, weighted functions are the most common. Given a database where each tuple has d attributes $\{a_1, ..., a_d\}$, $\mathcal{M}$ can be defined as:

$$\mathcal{M} = \{w_1a_1 + w_2a_2 + ... + w_da_d\},\ \text{where } w_i \in [0,1]\ \ i : 1, ..., d. \tag{2.5}$$

Definition 6. *F-dominance: Let $\mathcal{F}$ be a set of monotone scoring functions. A tuple s is $\mathcal{F}-$dominated by a tuple t ($t \prec_{\mathcal{F}} s$) if and only if:*

$$\forall f \in \mathcal{F} \;\; f(t) \leq f(s). \tag{2.6}$$

This operator outputs two different subsets obtained from the database. The first one is the $\mathcal{ND}$ subset which corresponds to the non-dominated tuples. The second one is the $\mathcal{PO}$ subset that includes the potentially optimal tuples which are the ones that are optimal according to a specific function in $\mathcal{F}$.

Definition 7. *$\mathcal{ND}$: Let $\mathcal{F}$ be a set of monotone scoring functions. The non-dominated subset $\mathcal{ND}$ is the set of tuples defined as the following:*

$$\mathcal{ND}(\mathcal{DB}, \mathcal{F}) = \{t \in DB \mid \nexists s \in DB, s \prec_{\mathcal{F}} t\}.$$

Definition 8. *$\mathcal{PO}$: Let $\mathcal{F}$ be a set of monotone scoring functions. The potentially optimal subset $\mathcal{PO}$ is the set of tuples defined as the following:*

$$\mathcal{PO}(\mathcal{DB}, \mathcal{F}) = \{t \in DB \mid \exists f \in \mathcal{F} : \forall s \neq t \in DB \Rightarrow f(t) < f(s)\}.$$

CarID	Price ($\times 10^3$)	Mileage ($\times 10^3$)
C1	10	35
C2	18	25
C3	20	30
C4	20	15
C5	25	20
C6	35	10
C7	40	5

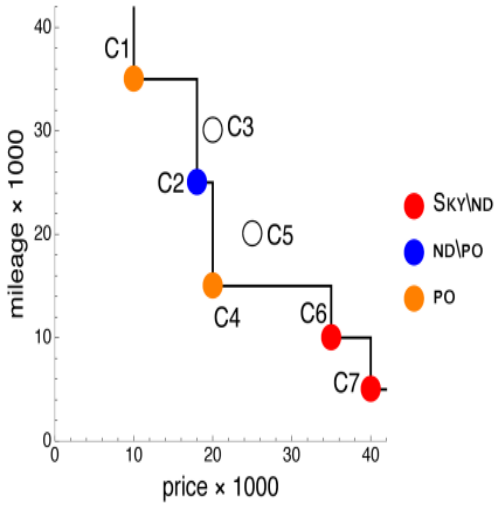


Figure 2.2: Example of $\mathcal{F}$-skyline operator over a car dataset with scoring function $\mathcal{F} = \{w_p Price + w_m Mileage | w_p \geq w_M\}$. Obtained from [2].

After defining the two output subsets, it is convenient to introduce the concept of dominance region, which consists of all the tuples that are dominated by a given one.

Definition 9. *The $\mathcal{F}$-dominance region of a tuple t is defined as the set of $[0,1]^d$ points, where d is the number of parameters per tuple, that are dominated by t:*

$$\mathcal{DR}(t,\mathcal{F}) = \{s \in [0,1]^d | t \prec s\}. \tag{2.7}$$

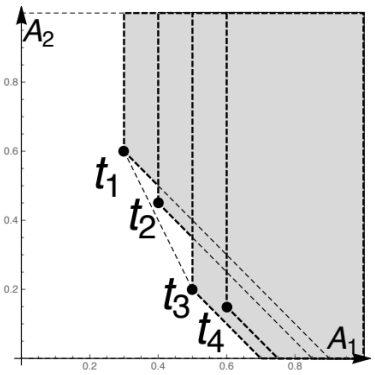


Figure 2.3: Example of dominance regions, areas highlighted in gray, note that $\mathcal{ND}$ points are never located within any dominance region. Obtained from [9].

To calculate whether a tuple belongs to another's dominance region, the following inequalities proposed in [9] are checked:

Let $\mathcal{W}$ be the set of all possible weight vectors and let $C = \{C_1, ..., C_c\}$ be a set of linear constraints on $\mathcal{W}$ weights. Let $V = \{W^1, ..., W^q\}$ be the set of vertices of $W(C) = \{W \in \mathcal{W} : C(\mathcal{W}) = true\}$. Then the dominance region of a tuple t using functions in $\mathcal{L}_p^C$ ($DR(t; \mathcal{L}_p^C)$) is the set of points defined by the inequalities:

$$\sum_{i=1}^{d} w_i^l s[A_i]^p \geq \sum_{i=1}^{d} w_i^l t[A_i]^p \qquad l \in 1, ..., q. \tag{2.8}$$

To finish the introduction to flexible skyline queries, an algorithm to generate the $\mathcal{ND}$ subset will be provided. In the following chapter, we will see how this algorithm can be modified to work with time series, adding context to skyline queries. Some relevant subprocesses included in the algorithm will be further explained, and practical implementations of them will be presented.

input : Relation of tuples r, constrains C, family $\mathcal{F} = L_p^C$
output: $\mathcal{ND}(r, \mathcal{F})$

1 $\{w_1, ..., w_q\} \leftarrow$ `VerticesOf(` $\mathcal{W}(\mathcal{C})$ `);`
2 $Centroid(\mathcal{W}(\mathcal{C})) \leftarrow$ `CentroidOf(` $\{w_1, ..., w_q\}$ `);`
3 $\mathcal{ND} := \emptyset$;
4 $r \leftarrow$ `Order(` r, $Centroid(\mathcal{W}(\mathcal{C}))$ `);`
5 **foreach** *tuple* s *of* r **do**
6 compute left hand side of inequalities in 2.8;
7 **foreach** *tuple* t *of* $\mathcal{ND}$ **do**
8 **if** $t \prec s \vee t \prec_{\mathcal{F}} s$ **then**
9 move to line 5;
10 **end**
11 **end**
12 $\mathcal{ND} := \mathcal{ND} \bigcup s$;
13 **end**
14 **return** $\mathcal{ND}$;

Algorithm 2.1: SVE1F algorithm to compute $\mathcal{ND}$, published in [6].

2.2. Context modelling in skyline queries

Introducing the notion of context combined with the tools for generating queries provided by the skyline operator is one of the main goals of this project. This implies researching and evaluating different approaches to uncertainty and context modelling adapted to skyline queries.

In this section, two main approaches to context modelling will be presented. The first one is based on the concept of probability distributions as an expression of uncertainty, and the second one is based on the concept of time series as a way to model the evolution of context. Both approaches will be provided combined with the skyline operator.

2.2.1. Probability in skyline queries

The previous concepts related to skyline queries were presented relying on deterministic data modelling. Thus, the skyline operator only considered results obtained from comparisons between tuples containing numerical data. But it is a natural concern to ask, what is going to be the case if this notion is extended to tuples containing probabilistic data; such as density distributions. An approach to this situation is given in [10].

Let $\mathcal{DB}$ be a database where each tuple is a multivariate probabilistic distribution $t = f(\overrightarrow{x}) \in \mathcal{DB}$ and has an associated vector $\overrightarrow{x} = (x_1, x_2, ..., x_d)$ which is an stochastic variable that follows $f(\overrightarrow{x})$ distribution. Within this context, the previous definitions of dominance applied to skyline queries cannot be trivially extended.

Definition 10. ***Probabilistic dominance*** *is defined as the probability that* $\overrightarrow{x}$ *generated by* $f(\overrightarrow{x})$ *dominates* $\overrightarrow{y}$ *generated by* $g(\overrightarrow{y})$*:*

$$P(\overrightarrow{x} \prec \overrightarrow{y}) = \int_{\mathbb{R}^d} \int_{\mathbb{R}^d} f(\overrightarrow{x}) * g(\overrightarrow{y}) * \mathbf{1}_{\overrightarrow{x} \prec \overrightarrow{y}} \, d\overrightarrow{x} \, d\overrightarrow{y}. \tag{2.9}$$

Using this definition of probabilistic dominance, it is possible to present the concept of skyline probability.

Definition 11. ***Skyline Probability*** *of a tuple* $f(\overrightarrow{x}) \in \mathcal{DB}$ *is the probability that an* $\overrightarrow{x}$ *generated by* $f(\overrightarrow{x})$ *is not dominated by any other* $\overrightarrow{y}$ *generated by* $g(\overrightarrow{y})$*, where* $g(\overrightarrow{y}) \in DB/f(\overrightarrow{x})$*. This is formulated as a probability intersection.*

$$P_S(f(\overrightarrow{x})) = P\left(\bigcap_{g(\overrightarrow{y}) \in DB/f(\overrightarrow{x})} g(\overrightarrow{y}) \not\prec f(\overrightarrow{x}) \right). \tag{2.10}$$

These concepts allow us to define the skyline operator in a probabilistic context. However, it is important to note that there are some issues that should be addressed. If common probabilistic distributions, such as the Gaussian, are used to model the data, the skyline query will likely be composed by the whole database. This is a natural consequence of the extreme values that distributions are able to reach, despite having very low probabilities. To avoid this situation, it is convenient to set a threshold for the skyline probability in order to filter out those extreme cases that are highly improbable.

This approach, despite providing a nice way to operate with probabilistic data in skyline queries, does not fit well with the purpose of this project. Since the topic modelling system will already return fixed probabilities for each tuple, thus there is no need to model probabilistic distributions within the skyline context. Besides, [10] mentions that, despite being an interesting theoretical concept, the actual implementation would be extremely demanding in computational terms, since probabilistic dominance requires evaluating $P(\overrightarrow{x} \prec \overrightarrow{y})$ for every possible vector generated by both distributions.

2.2.2. Time series modelling

A time series is defined as a sequence of data points indexed in time, usually in equal intervals, which translates to a list of discrete data. The main challenge posed by this type of structure when applied to databases is its high dimensionality, produced by the large volume of time series stored [11] [12]. This increases the cost of performing queries based on similarity since the distance between two time series is usually expressed as the sum of the euclidean distance between each pair of their points. To overcome this issue, several techniques have been developed with the goal of performing dimensionality reduction on data, for example Singular Value Decomposition (SVD), Discrete Fourier Transform (DFT) or Discrete Waveler Transform (DWT). However, more efficient methods have appeared, such as Adaptive Piecewise Constant Approximation(APCA) [11].

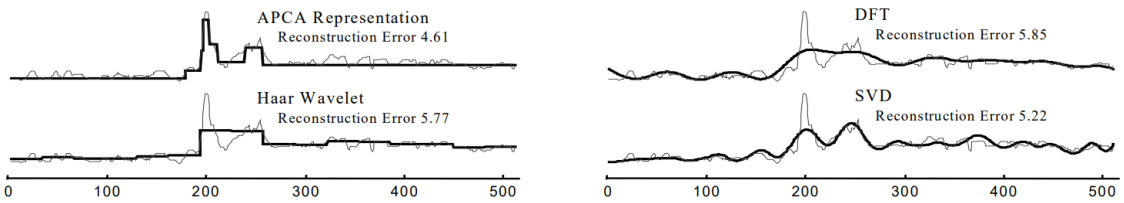


Figure 2.4: Comparison between APCA and other previous dimensionality reduction techniques. Obtained from [11].

Adaptive Piecewise Constant Approximation(APCA)

APCA is a technique used to approximate a time series with a piecewise constant function. This means that, each time, a series is approximated by a set of constant value segments of varying length that minimize the time series reconstruction error.

The main advantage of this technique is that the loss in precision due to the approximation error is generously compensated by a large speedup produced by the possibility of indexing the approximations using R-trees [11].

The formal description of the APCA is as follows: Given a time series $T = (t_1, t_2, ..., t_n)$, there is an associated APCA representation $C = \{< tv_1, tr_1 >, ..., < tv_M, tr_M >\}$ where tv_i is the mean value of data points within the i-th segment and tr_i is the right endpoint of the i-th segment. Note that the segment length is not explicitly represented to ease indexing in the database [11]. To visualize this concept, consider the following example:

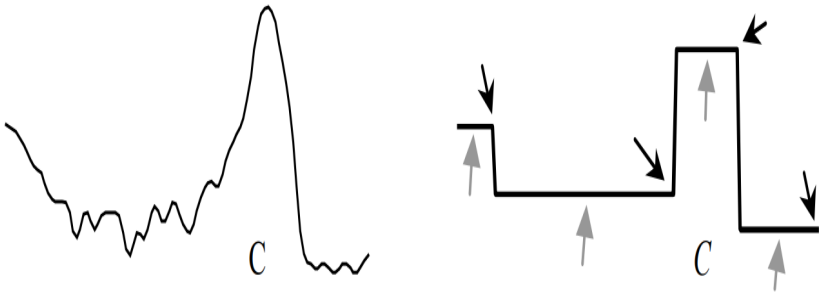


Figure 2.5: A time series C and its APCA representation C, M = 4. Obtained from [11].

In terms of computational cost, the process of finding the optimal piecewise representation of a time series has a complexity of $O(Nn^2)$, where N is the number of points in the time series [13]. However, since most tasks do not require fully optimal representations, greedy suboptimal algorithms are commonly used to reduce complexity [14]. In the original APCA paper, an original algorithm was used to achieve complexity $O(n \log(n))$ [11].

Time series applied to skyline queries

In [12] an approach to compute skyline queries over time series is presented. This new perspective is based on the idea of using the APCA technique to estimate a skyline region named **Skyline Bounding Region (SBR)**, which is constructed from several time series contained in a database. After identifying the SBR, the APCA method is used to compute an estimation of the SBR.

As a core concept in the system, the Skyline Bounding Region (SBR) over a time period $[0, T]$ is defined over a group of time series $S = s_1, ..., s_n$ of length l, like the region of the space delimited by the top and bottom skylines and two vertical lines connecting both of them at times 0 and T. Top and bottom skylines are defined as follows:

$$\begin{aligned} TSky = \{ts_1, ..., ts_l\}, \qquad & BSky = \{bs_1, ..., bs_l\}; \\ ts_i = max\{s_1[i], ..., s_n[i]\}, \qquad & bs_i = min\{s_1[i], ..., s_n[i]\}. \end{aligned} \tag{2.11}$$

Using the previous definitions, the SBR can be defined:

$$SBR = \{< t, ts_i, bs_i > | 0 \leq t \leq T, 1 \leq i \leq l\}. \tag{2.12}$$

The region, as shown in Figure 2.6, can be defined in a continuous sense, but in practical terms must be estimated as mentioned before in this section. The authors of the text compared this methodology

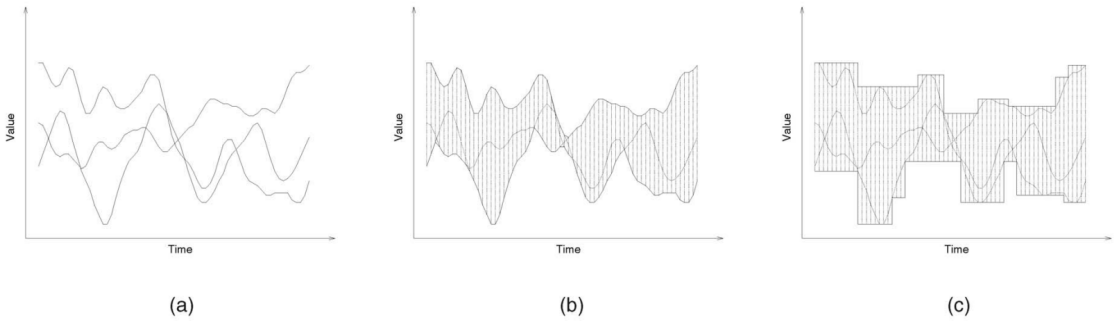


Figure 2.6: Example of SBR construction. (a) Three time series (b) Continous SBR (c) Approximate SBR. Obtained from [12].

of indexing with the one proposed in [11] using only APCA. For clarity, the formulas used to calculate accuracy are shown below:

$$\begin{aligned} Q_{SKY}(I_R) &= \frac{Area(ApproximateSBR)}{Area(DataBoundingRegion)}, \\ Q_{APCA}(I_R) &= \frac{Area(\bigcup_{i=1}^{M}(R_i))}{Area(DataBoundingRegion)}; R_1, ..., R_M = R. \end{aligned} \tag{2.13}$$

Note that R is the set of rectangles defined by SBR for node I_R using APCA indexing. The comparison's results can be seen in Table 2.1.

Segments	$Q_{APCA}(I_R)$	$Q_{SKY}(I_R)$
8	1.673	1.326
16	1.7	1.345
32	1.7	1.333

Table 2.1: Bounding regions error. Data obtained from [12].

Results show that using the skyline based approach reduces the error in the bounding region by roughly 25 %. Besides that, the graphical visualization of SBR is more intuitive than the one of APCA since it is only one region in the space not a set composed by overlapping rectangles like R is in APCA.

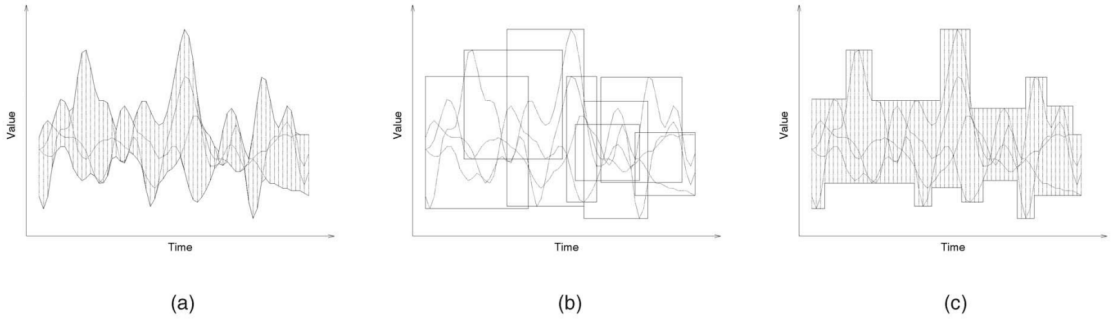


Figure 2.7: Index bounding regions of APCA and SBR compared to data bounding region. Obtained from [12].

2.3. Topic detection systems

Topic detection systems are based on a variety of machine learning methods that are capable of analyzing and organizing large volumes of text into topics. It is important to remark that the term “topic” in this specific context is not used to represent a general category such as economy or sports. Instead, topics are groups representing particular trends, and topic labels within topic detection systems are usually updated according to the most recent documents added to the training set. For example, in [15], a modification in the model used to perform topic detection changed the label “sports” to “american sports”, while using the same training set.

To generate topics and classify documents, systems rely on a variety of methodologies such as clustering or statistical modelling, that combined with natural language processing techniques, allow them to associate words or text structures to a given topic.

These systems can be classified according to the approach they take to detect topics embedded in text [16]. In this section, we will present the most common approaches to topic detection; which include clustering approaches as well as generative approaches that use statistical distributions and Bayesian theory to generate topic distributions [17].

2.3.1. Clustering models

Clustering methods are commonly used for topic detection since they allow to group together text elements that are alike, producing subgroups or clusters with greater homogeneity. The inferences made by the process can be particularly helpful to detect topics in a collection of documents [18]. Two main types of clustering methods will be mentioned in this section: hierarchical clustering and iterative clustering.

Hierarchical clustering

Hierarchical agglomerative clustering: It is a clustering method that transforms a *proximity matrix* into a sequence of nested partitions in which each partition is nested into the next one in the sequence. This is an agglomerative method since all of the structures in the corpus will finish the process nested into a single cluster [18].

The process starts with each element being contained in a single cluster. In each iteration, the two closest elements in the *proximity matrix* will join in one cluster and this process will continue until all elements are contained in one. The result of this process is a hierarchical structure represented by a *dendrogram*.

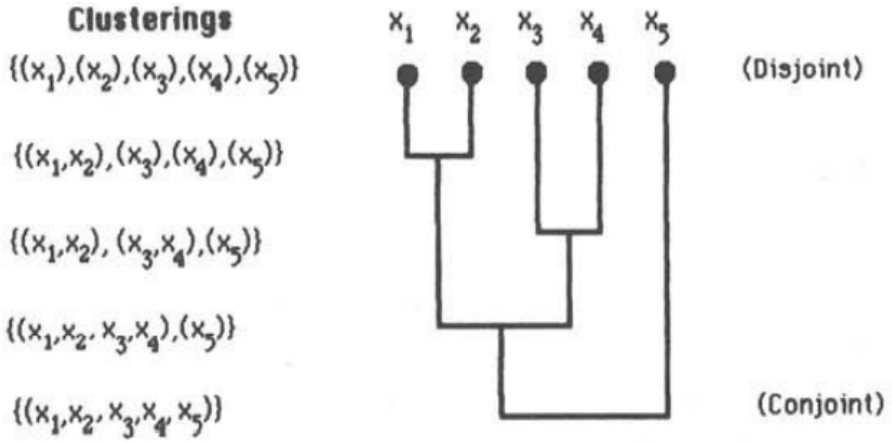


Figure 2.8: Example of *dendrogram*, where x_i represents each observed element to be clustered. Obtained from [19].

There are three different ways to build a *dendrogram*, which translates in three different clustering algorithms depending on how elements are clustered together:

- **Single-link:** Distance between two clusters is defined as the distance between the closest elements of each cluster.
- **Complete-link:** Distance between two clusters is defined as the distance between the farthest elements of each cluster.

- **Average-link:** Distance between two clusters is defined as the average distance between all the elements of each cluster.

Principal Direction Divisive Partitioning (PDDP): This approach consists of creating a root tree holding all the elements within the corpus and recursively partitioning them following a given criteria. A common approach, described in [18], is to partition the cluster if the mean square distance in it is greater than some specified threshold. In this case, the mean square distance is the scatter function and keeps the tree balanced. Once the criterion is satisfied in all the leafs, the algorithm terminates.

Iterative clustering

Iterative clustering methods are based on the idea of function optimization, this function may be defined either globally or locally [20]. In this section, two of the most popular versions of this approach will be presented: K-means and Expectation Maximization.

K-means: It is a centroid-based clustering algorithm that creates a set of k clusters and relocates elements within them minimizing the distance between elements and the centroid of the cluster [19].

- **Initialization:** Once picked the number of clusters k, generate a centroid vector $\mu = (\mu_1, ..., \mu_k)$, usually, this vector is randomly generated.
- **Assignment:** Each element to be classified is assigned to the closest μ_i.
- **Update:** For each $\mu_i \in \mu$, update its value to the mean of its assigned elements. This is achieved by locally minimizing the following function, where $x_j \in S_i$ represents the elements assigned to μ_i:

$$min_S(\mu_i) = min_S \sum_{i=1}^{k} \sum_{x_j \in S_i} \|x_j - \mu_i\|^2 . \tag{2.14}$$

Repeat *assignment* and *update* steps until the centroid vector does not change or the change performed is under a given threshold.

This method is usually employed to identify unlabeled groups within a dataset, however, the problem's optimality is NP-hard and thus, it is necessary to find an approximation to the solution using heuristics that converge to a local optimum in an iterative fashion. There are some variations of this algorithm such as K-medoids, K-means++ or K-medians.

DBSCAN or Density-based spatial clustering of applications with noise, is one of the most common algorithms for what is called "density-based clustering". The fundamental idea behind this method is fairly simple. Each point in the dataset is mapped in space using a distance metric. Then, those points with many close neighbors are tagged as generators and grouped into the same cluster along with their neighbors. The points that do not have any generators within a specified radius are considered outliers and are not assigned to any cluster.

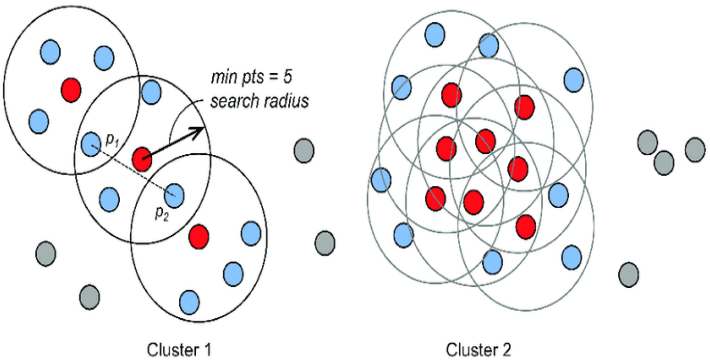


Figure 2.9: The DBSCAN algorithm and two generated clusters. Red points are classified as cluster generators; blue points are not cluster generators but fall within a generator's radius; grey points are labelled as outliers. Obtained from [21].

2.3.2. Statistical models

There are a variety of statistical models that can be used to detect topics, some of the most common ones being those based on Expectation Maximization (EM), [22] or on Latent Dirichlet Allocation [23], which will be the main focus of this section.

Expectation Maximization is a statistical approach to obtaining maximum likelihood estimates of parameters in probabilistic models. The algorithm is best suited for situations where there are missing data or when there is a portion of the data that is not observed (latent data) [22]. This algorithm has been used extensively in different fields such as data mining, machine learning, and Bayesian statistics.

The goal of the algorithm is to maximize the likelihood function $L(\theta, X)$. θ is a vector of unknown model parameters that need to be estimated and X is the set of observed data generated by the statistical model. The likelihood function is defined as follows:

$$L(\theta, X) = \int_y p(X, y|\theta)dy. \tag{2.15}$$

The expectation maximization estimate is modified by performing iterations, each one of them is composed by two steps [24]. The E-step computes the expectation of the data likelihood function $L(\theta, X)$. The M-step is used to maximize θ with respect to the expectation computed in the E-step. The algorithm continues iterating until the sequence of computed θ converges, usually to a local maximum, [22]. If that is the case, and through iterative runs of the algorithm several local maximums were to be found, then the higher one would be chosen.

Latent Dirichlet Allocation (LDA) is a generative probabilistic model that acts on discrete data collections. In this model, Blei, Ng and Jordan [25], propose the analysis of collections composed of discrete elements, usually words, where each one of them is described by a finite distribution over a set of topics and each topic is described by an infinite distribution over a set of probabilities.

This model is considered the fundamental basis of many other more advanced models [26], such as the Correlated Topic Model (CTM) [27] .The following table provides a basic description of the variables involved in LDA:

Variable	Description
M	Number of documents.
N_i	Number of words in document i.
K	Number of topics.
V	Number of words in vocabulary.
α	Vector of length K, contains prior topic weight in a document.
β	Vector of length V, contains prior word weight in a topic.
θ_i	Distribution of words in topic i.
ϕ_i	Distribution of topics in document i.

Table 2.2: LDA model variables. Data obtained from [25].

Documents are represented by distributions of latent topics, each topic is represented as a distribution over all the analyzed words. The generative process is as follows:

- For each document: $\theta_i \;\; i : 1, ..., M$; is chosen from a Dirichlet distribution with parameter α.
- For each topic: $\phi_j \;\; j : 1, ..., N$; is chosen from a Dirichlet distribution with parameter β.
- For each word in each document $(i, j), i : 1, ..., M; j : 1, ..., N_i$:
  - A topic $z_{i,j}$ is chosen from a multinomial distribution with parameter θ_i ($z_i \sim M(\theta_i)$).
  - A word $w_{i,j}$ is chosen from a multinomial distribution with parameter $\phi_{z_i,j}$ ($w_{i,j} \sim M(\phi_{z_i,j})$).

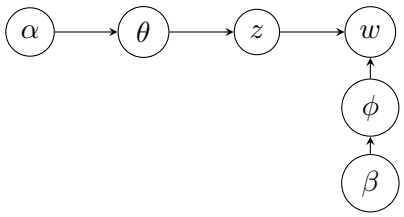


Figure 2.10: LDA basic structure model.

LDA models generate words as a two step process. Words are generated from topics and topics are generated from documents, thus the probability of a word w_i being in a document d is:

$$p(w_i|d) = \sum_{j=1}^{K} p(w_i|z_i)p(z_i|d). \tag{2.16}$$

The two distributions used to calculate the probability of a word being in a document in Equation 2.16 are learned in an entirely unsupervised manner. To do so, it is necessary to use an inference technique, one of the most popular is Gibbs sampling, see [28]. To conclude the presentation of the model, the following equation will show the complete LDA probabilistic distribution:

$$P(w_{1:M}, z_{1:M}, \theta_{1:M}, \phi_{1:K}) = \prod_{i=1}^{M} P(\theta_i|\alpha) \prod_{j=1}^{K} P(\phi_j|\beta) \prod_{i=1}^{M} \prod_{j=1}^{N_i} P(z_{i,j}|\theta_i) P(w_{i,j}|\phi_{z_{i,j}}). \tag{2.17}$$

Correlated Topic Model (CTM): This model was presented by Blei and Lafferty in 2006 [29]. It is based on LDA and aims to eliminate one of its main limitations, the inability to consider the correlation between topics.

To do this, a normalized logistic distribution is incorporated into the system to model the relationships between topics. This method was demonstrated to be able to produce a fit with a lower error than basic LDA after being trained with a dataset that included about 5.7 million words.

Biterm Topic Model (BTM): This model was presented in 2013 [27] as a fundamentally different alternative to the LDA model. Its goal was to obtain better performance in the analysis of short texts where the co-occurrence ratio of terms is very low, and therefore, LDA-based methods are inefficient.

The idea behind BTM is to use biterms, unordered pairs of words that appear simultaneously in a short text, for example: *science* $\sim$ *biology*. Then, topics are defined as distributions over these biterms, not on the individual occurrence of each word.

3

Time-aware skyline queries

In the previous chapter, several new approaches to leverage temporal or probabilistic variables in skyline queries were presented. Nevertheless, neither of them addressed the specific problem of query generation in a time-aware context. The dominance relationship between tuples defined by the flexible skyline operator, see Equation 2.6, is based on a set of scoring functions that are useful to compare numeric values, but fail to score time-related variables, for example, the date of publication of a news article.

This section will present a theoretical modification of the flexible skyline operator that will allow the use of time-related variables, not only aiming to adapt them to be computed by the scoring functions, but as distributions that will affect all the attributes involved in the dominance relationships. Moreover, the SVE1F algorithm, proposed in [6], will be adapted to implement the modification and generate time-aware skyline queries that will produce different sets of non-dominated tuples ($\mathcal{ND}$ sets) depending on time.

Moving forward we will consider a database $\mathcal{DB}$ where each tuple $s \in \mathcal{DB}$ is defined as a set of attributes $s = \{a_1, a_2, \ldots, a_n \ t_s\}$, where a_i is the value of the i-th attribute and t_s is a timestamp associated to the tuple.

The first step to adapt the flexible skyline operator to a time-aware context is to normalize the time attributes. To do so, consider the following function: $\mathcal{N}(t_s) : \{t_s : s \in \mathcal{DB}\} \longrightarrow [-1, 1]$, where t_{max} and t_{min} are the maximum and minimum values of the time attribute in the database, respectively.

$$\mathcal{N}(t_s) = \frac{(t_s - t_{min}) \cdot 2}{t_{max} - t_{min}} - 1. \tag{3.1}$$

3.1. Generation of time distributions

The concept of time within skyline queries is especially difficult to deal with for two main reasons. Since time is a context variable, it may be potentially influential on other attributes' relevance. For

example, given that hotel prices are usually higher during the summer; a lower price becomes a high priority during that specific time interval, thus increasing the "price" attribute relevance.

Second, time has a relative value depending on the context. When looking for sport news articles, one may be interested in the most recent ones, but when looking for historical articles, the most recent ones may not be the most relevant. This implies that there is not a proper way to globally define what time values are "better" than others, which means that even normalizing them is not enough to compare them. Time-aware flexible skylines will use time distributions to overcome the aforementioned problems.

Definition 12. ***Time distribution:*** *Let $\mathcal{DB}$ be a database where each tuple $s \in \mathcal{DB}$ has n attributes. A time distribution $\mathcal{TD}$ of $\mathcal{DB}$ is defined as:*

$$\mathcal{TD} = \{f_1, f_2, \ldots, f_n \mid f_i : [-1, 1] \longrightarrow \mathbb{R}^+\}. \tag{3.2}$$

This vector of functions intends to increase the query's expressiveness by defining time influence over each individual attribute in the database. For example, given a tuple containing three attributes, where the first one is unrelated to time, the second one becomes relevant as time passes and the third one losses value for extremely old or new time values, the following time distribution could be defined:

$$\mathcal{TD} = \{f_1, f_2, f_3\} = \{1, x/2 + 1, \phi(0, \frac{1}{2}) + \frac{1}{2}\}.$$

Where $\phi(a, b)$ is a Gaussian function with form $\phi(a, b) = e^{-\frac{(x-a)^2}{2b^2}}$.

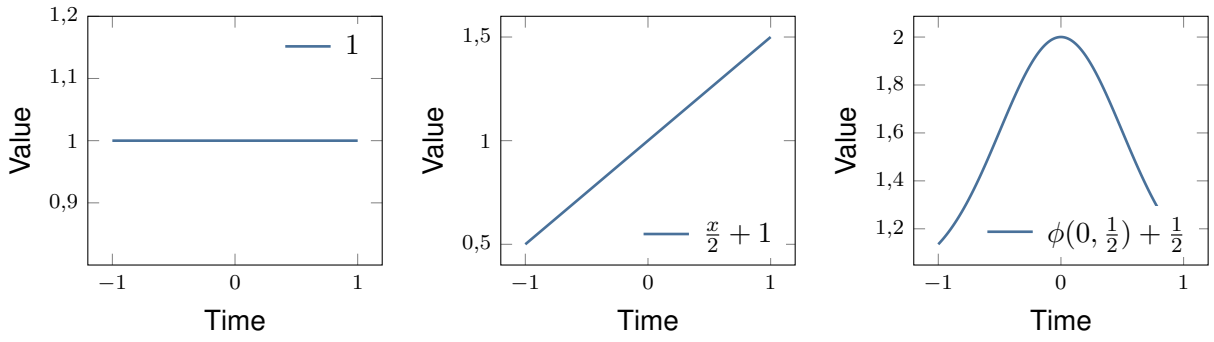


Figure 3.1: Time distribution functions of $\{1, x/2 + 1, \phi(0, \frac{1}{2}) + \frac{1}{2}\}$.

3.2. Modification of $\mathcal{F}$ subset

Time distributions allow the expression of individual time behaviors for each analyzed attribute. However, it is still unclear how they fit into the flexible skyline operator querying process. To understand their role, it is necessary to recall that the $\mathcal{F}$ dominance set of scoring functions is defined as a subset of an infinite set $\mathcal{M}$, which contains all the possible scoring functions that can be defined over the

database. These concepts will be used to include time distributions in the flexible skyline operator with the objective of generating the non-dominated $\mathcal{ND}$ set. In [6], a theorem key to understanding the difference between the different output subsets of skyline queries is presented but not proven. This result is important to understand the relationship between the output sets and $\mathcal{F}$ as a subset of $\mathcal{M}$, a proof will be provided in this section.

Theorem 1. *$\mathcal{PO} \subseteq \mathcal{ND} \subseteq \mathcal{SKY}$. In particular, if $\mathcal{F} = \mathcal{M}$: $\mathcal{PO} = \mathcal{ND} = \mathcal{SKY}$.*

Proof. Given a tuple $t \in \mathcal{PO}$, by definition: $\exists f \in \mathcal{F}\ s.a.\ \forall s \neq t\ f(t) < f(s)$. Using a proof by contradiction, suppose: $\exists s\ s.a.\ s \prec_{\mathcal{F}} t \Rightarrow \forall f \in \mathcal{F}\ f(s) \leq f(t)$, which is a contradiction. Therefore $t \in \mathcal{ND}$.

The second part of the expression, $\mathcal{ND} \subseteq \mathcal{SKY}$ requires an slightly more complex proof:

$$t \in \mathcal{ND} \Rightarrow \nexists s \in DB\ \ s.a.\ s \prec_{\mathcal{F}} t \Rightarrow \forall s \in DB\ \ \exists f \in \mathcal{F},\ \ f(t) \leq f(s).$$

Observe that f is not necessarily the same for all $s \in DB$, thus, consider S as the set composed by the functions that allow t not to be dominated by any tuple. Define the function $g \in \mathcal{M}$ as:

$$g(s) = \begin{cases} min_{f \in S}(f(s)) & \text{if } s \neq t, \\ max_{f \in S}(f(s)) & \text{otherwise.} \end{cases}$$

Recalling that, for each tuple $s \neq t$, $\exists f \in S$ such that $f(t) \leq f(s)$, it follows that g is a monotone scoring function that fulfills the $\mathcal{SKY}$ property, Equation 2.3, for t.

Note that a key element in the proof is the fact that g is defined over $\mathcal{M}$, if $\mathcal{F} = \mathcal{M}$ then it becomes clear that $\mathcal{PO} = \mathcal{ND} = \mathcal{SKY}$. □

In the previous chapter, a common way to define $\mathcal{M}$ was introduced, see Equation 2.5, where $\mathcal{M}$ is defined as the set of all possible weighted sums of a tuple's attributes, this definition will be the one used from now on. $\mathcal{F}$ will be the result of imposing certain conditions over $\mathcal{M}$, which will restrict the set of possible weights. This environment will be used to define the time-aware dominance set $\mathcal{F}_{TD}$.

Definition 13. *Let $\mathcal{F}$ be a subset of $\mathcal{M}$ weighted scoring functions that fulfills C conditions:*

$$\mathcal{F} = \{f(s) = w_1 s_1 + ... + w_n s_n \in \mathcal{M} \mid C(f) = true\}, \tag{3.3}$$

and let $\mathcal{TD} = \{f_1, f_2, ..., f_n\}$ be a time distribution. Then the time-aware dominance set $\mathcal{F}_{TD}$ is defined as:

$$\mathcal{F}_{TD} = \left\{ f(s) = \hat{W}_1 s_1 + ... + \hat{W}_n s_n \in \mathcal{M} \mid \hat{W}_i = \frac{\vec{W}_i}{|W|} = \frac{w_i \cdot f_i(s_t)}{\sum_{j=1}^{n} w_j \cdot f_j(s_t)} \right\}. \tag{3.4}$$

This new subset of scoring functions is generated by normalized weights that are composed by a constant w_i which expresses a preference over the attribute s_i and a function $f_i(s_t) \in TD$ which expresses its temporal relevance, thus modifying its weight depending on the time value s_t of the tuple s.

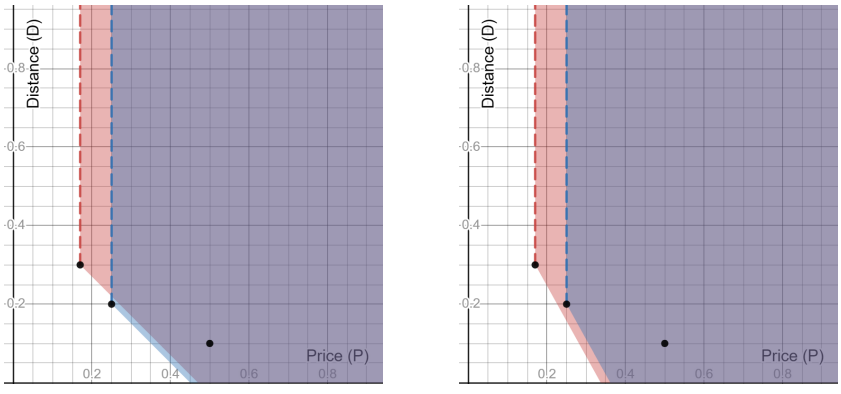
Example: Consider a variation of the classic example provided in *The Skyline Operator: A New Operator for Data* [4]. A travel agency wants to offer the most interesting hotels to its customers based on the following criteria: price and distance to the city center. However, clients concerned about the increment in prices during the summer want to prioritize that parameter during that period. These tuples are the ones included in the database; consider that the time interval [-1,1] is equivalent to a year from January to December:

Hotel	Price(P)	Distance(D)	Time (normalized [-1,1])
Hotel 1 (H1)	0.17	0.3	0
Hotel 2 (H2)	0.25	0.20	0
Hotel 3 (H3)	0.5	0.1	0

Table 3.1: Example 1: Hotel database tuples.

Lets consider a particular case in which the agency never considers any criteria that prioritizes distance over price. In this case, $\mathcal{F} = \{w_1 P + w_2 D | w_1 \geq w_2\}$. However if the agency wants to consider temporal context they could provide a time distribution: $TD = \{\phi(0, \frac{1}{2}) + \frac{1}{2}, 1\}$ which increases the relevance of *Price* attribute during the summer months. Thus defining the following scoring function set:

$$\mathcal{F}_{TD} = \{\hat{W}_1 P + \hat{W}_2 D | \vec{W}_1 = w_1 \cdot \phi(0, \frac{1}{2}) + \frac{1}{2}, \vec{W}_2 = w_2 \cdot 1\}. \tag{3.5}$$



(a) Dominance regions obtained using $\mathcal{F}$.

(b) Dominance regions obtained using $\mathcal{F}_{TD}$.

Figure 3.2: Example 1: Dominance regions in time ($t = 0$).

As it can be seen in Figure 3.2, the point H2 (0.25,0.20), is not dominated when $\mathcal{F}$ is considered

since the tradeoff between price and distance is sufficient. However, taking context into account, it can be noticed that all tuples have the same normalized date around mid-year. Because of that, the increased relevance of price in $\mathcal{F}_{TD}$ alters $\mathcal{ND}$ and the point H2 falls into H1 dominance region.

3.3. Modified SVE1F algorithm

The proposed modification to the process of obtaining the $\mathcal{ND}$ subset cannot be immediately implemented using the already designed algorithms in [6] and [9]. This is because, while the standard weights are constant numbers that act equally over all the tuples, the new weights are composed by functions that depend on the timestamp value of each individual tuple.

In this section, a modified version of SVE1F, Algorithm 2.1, will be proposed in order to be able to process the new set of scoring functions. To do so, it is necessary to identify the algorithmic processes that involve scoring functions:

- Computation of the set of vertices V.
- Initial ordering of the tuple set using V centroid.
- Computation of the dominance region of a tuple, using the inequalities defined in Equation 2.8.

These processes present difficulties because all of them depend on the set of scoring functions. The first one is especially problematic since it implies calculating all the vertices $W^1, ..., W^q$ in the set of scoring functions $\mathcal{W}(C) = \{W \in \mathcal{W} : C(\mathcal{W}) = true\}$. This, given the fact that $\mathcal{W}(C)$ would be different for each tuple, would imply an extremely high computational cost to maintain correctness.

3.3.1. Database virtualization

In order to avoid performing the computations mentioned above, a virtualization of the database is proposed. The core idea behind this concept is that in relevant calculations, the weights of $\mathcal{F}_{ND}$ appear multiplying the attributes of the tuples, see Equations 3.4 and 2.8. This implies that for W_i weights, the constant factor w_i could be maintained, and the non-constant factor, which is time-dependent, $f_i(t)$ could be multiplied by the attribute's value a_i of the tuple. This would result in a new virtualized tuple that could be used to perform calculations using the usual SVE1F algorithm. Let $s \in \mathcal{DB}$ be a tuple, and let s_i be the value of the i-th attribute of s. Then:

$$\hat{W}_i \cdot s_i = \frac{w_i \cdot f_i(s_t)}{|W|} \cdot s_i = w_i \cdot s'_i, \tag{3.6}$$

where $s' \in \mathcal{DB}$ is the virtualized tuple associated with s.

Using this argument we proceed to tackle the three conflictive processes mentioned above. To calculate V, which is the set of $\mathcal{W}(C)$ vertices, only the usual w_i weights are used, thus this process is unaltered from its version in SVE1F. The initial tuple sorting required to perform the algorithm uses the following equation to calculate the scores of each tuple and proceed to sort them in descending order:

$$s_{score} = \sum_{i=1}^{n} ce_i \cdot s_i, \tag{3.7}$$

where ce_i is the i-th coordinate of the centroid ce of the set of vertices V. This equation can be modified to use the virtualized tuples as follows:

$$s_{score} = \sum_{i=1}^{n} \frac{ce_i f_i(s_t)}{|W_{ce}|} s_i = \sum_{i=1}^{n} ce_i s'_i. \tag{3.8}$$

Finally, the computation of tuple dominance regions is performed using the inequalities defined in Equation 2.8 and is precomputed in line 6 of Algorithm 2.1. Again, the application of the virtualized tuples should suffice to maintain correctness. Let $s, v \in \mathcal{DB}$, s is in v dominance region if and only if:

$$\sum_{i=1}^{d} \frac{w_i^l f_i(s_t)}{|W|^l} s_i^p \geq \sum_{i=1}^{d} \frac{w_i^l f_i(v_t)}{|W|^l} v_i^p \qquad l \in 1, ..., q, \tag{3.9}$$

which is equivalent to:

$$\sum_{i=1}^{d} w_i^l s_i'^p \geq \sum_{i=1}^{d} w_i^l v_i'^p \qquad l \in 1, ..., q, \tag{3.10}$$

where s' and v' are the virtualized tuples of s and v respectively.

These modifications maintain the algorithm's correctness while producing time-aware skyline queries due to tuple virtualization, which allows to perform the standard computations proposed in [6]. Moreover, the algorithm is expected to have the following properties:

- **Expression power:** The algorithm now supports the conditions over weights that characterize $\mathcal{F}$ subset as well as time distributions. This allows one to express a priori preferences over the attributes and tweak their relevance over time.
- $\mathcal{ND}$ **cardinality control:** Functions included in time distributions can be used to control the cardinal of $\mathcal{ND}$, when $f_i \in TD$ reaches higher values, tuples with low s_i values will become more dominant, thus decreasing the cardinality of $\mathcal{ND}$.
- **Time complexity:** The modifications proposed to the algorithm do not theoretically add complexity. As mentioned in [6], it is expected that the algorithm has a time complexity of $O(|r| \cdot |\mathcal{ND}|)$, where $|r|$ is the number of tuples in $\mathcal{DB}$.

4

Design and implementation of the system

This chapter focuses on the design of the test bed system that will be used to support the generation of time-aware skyline queries. The design of the system will be divided in the following sections:

- **Data preprocessing:** Focused on describing the process of raw data cleaning as well as the text preprocessing required to apply the topic detection algorithm.
- **Topic detection system:** Implementation of the topic detection system and explanation of different design choices taken to maximize topic coherence.
- **Skyline query system:** Description of implementation details of the modified SVE1F algorithm presented in Chapter 3.

Before going into individual details of each section, a general system workflow diagram is presented:

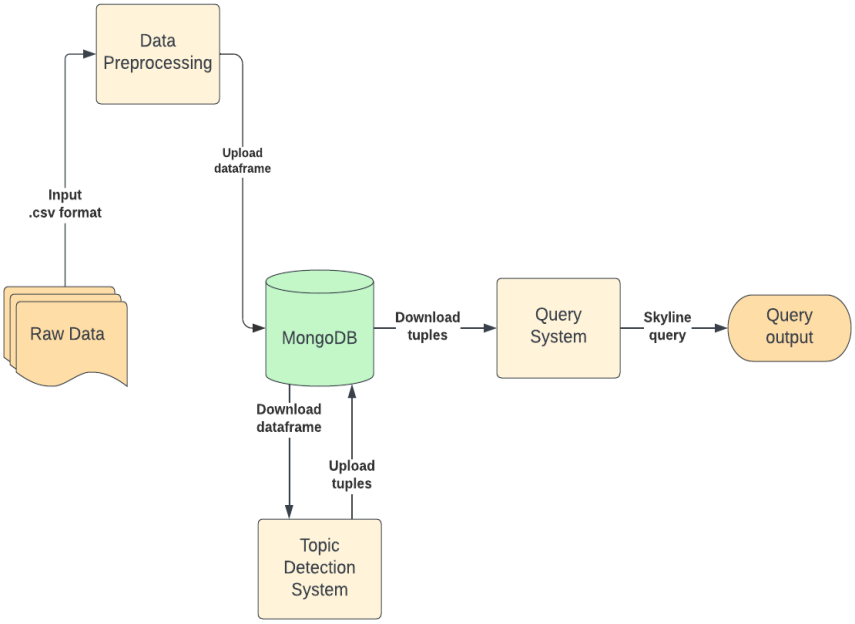


Figure 4.1: General system flow diagram.

4.1. Data preprocessing

The data acquisition process will be carried out by generating a dataframe from a .csv file containing the following fields:

Field	Type
id	string
page_id	string
name	string
message	string
description	string
caption	string
post_type	int
status_type	int
likes_count	int
comments_count	int
posted_at	string

Table 4.1: News dataframe fields.

The dataframe will undergo a data cleaning process to filter out the unnecessary elements and to prepare the data for the topic detection system. The following steps will be taken:

- Remove the following rows from the dataframe: name, caption, post_type, status_type, likes_count, comments_count. These fields are not relevant for the topic detection system.
- Transform the posted_at field into a datetime object using *Datetime* library.
- Remove NaN values in message and description fields. Note that page_id and posted_at fields are present in all rows, despite that, NaN are also preventively removed in these fields.
- Concatenate message and description fields into a new field called text.

4.1.1. Text preprocessing

Text preprocessing will happen before loading the dataframe objects into the database to avoid unnecessary data storage. GenSim library [30] will be used to apply a simple tokenization process removing accents and capital letters. Afterwards, the Natural Language Toolkit (NLTK) library will be used [31] to download a stopwords set, the tokenization process will be able to remove non-significant words that otherwise would be an obstacle to the topic detection system. The dataframe will be uploaded to a MongoDB cluster once the preprocessing has been completed.

Field	Type
id	string
page_id	string
text	string
posted_at	Datetime

Table 4.2: Topic detection system input fields.

4.2. Topic detection system

The topic detection system plays a fundamental role in producing suitable topic distributions that can be queried afterwards. To successfully carry out this process, it is important to take adequate design choices capable of increasing key metrics such as perplexity or topic coherence. Given that the training set is composed of a relatively small number of texts, around 340000, computational speed can be traded for more accurate results. In [29] it is mentioned that Latent Dirichlet Allocation can be more efficient than other clustering methodologies and it has become a staple solution when it comes to topic detection. In addition, the output of LDA is a topic distribution where all parameters are between zero and one and the sum of all parameters is equal to one. This turns out to be extremely convenient to perform flexible skyline queries.

To properly design the module, it is necessary to take into account the characteristics of the data set. The texts have been gathered from 2016 onwards, and the average text length is 43.2 words. This is a small number of words compared to other data sets. Which is likely produced due to the use of complementary multimedia content such as images or videos, as explained in [32].

In [27], it is mentioned that reduced length increases error in methods such as LDA due to how sparse words are, and the use of a Biterm Topic Model (BTM) is recommended. This model operates very similarly to LDA but instead of distributing topics over words, it distributes topics over biterms. Which are pairs of words appearing together in the same corpus. To increase topic detection accuracy, BTM will be used instead of LDA.

Moreover, [33] mentions the advantages of introducing author-topic distributions, which also increase topic coherence. Since the used dataset comes with a page_id field that denotes the media that published the news, topic-author distributions will be used.

The design of the topic detection module is meant to leverage the data input to its fullest, however, it will not consider the temporal data included in the input. The reason behind this decision is to isolate the time evaluation in the skyline query model. Not doing so would enormously increase the difficulty of interpreting the final results.

The LDA model used incorporates author-topic a distributions. As mentioned in [33], each author is associated with a distribution of topics that is then used to perform LDA. The originally proposed

model also presents the possibility of selecting an author from a distribution of possible ones when a document has been co-authored. However, since this is not the case, the model has been simplified removing that functionality.

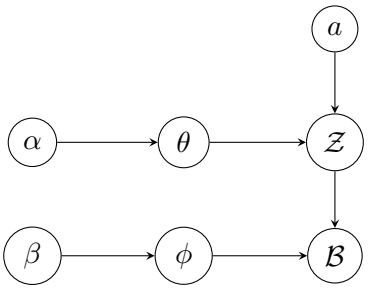


Figure 4.2: LDA based model structure implementing BTM and author-topic distributions.

It is easy to see that the only two modifications with respect to the original LDA model diagram, Figure 2.10, are changing words W by biterms $\mathcal{B}$ and the addition of the a parameter, representing the author-topic distributions, which affects the topic selection $\mathcal{Z}$. In the following diagram, a flow of the implementation is presented:

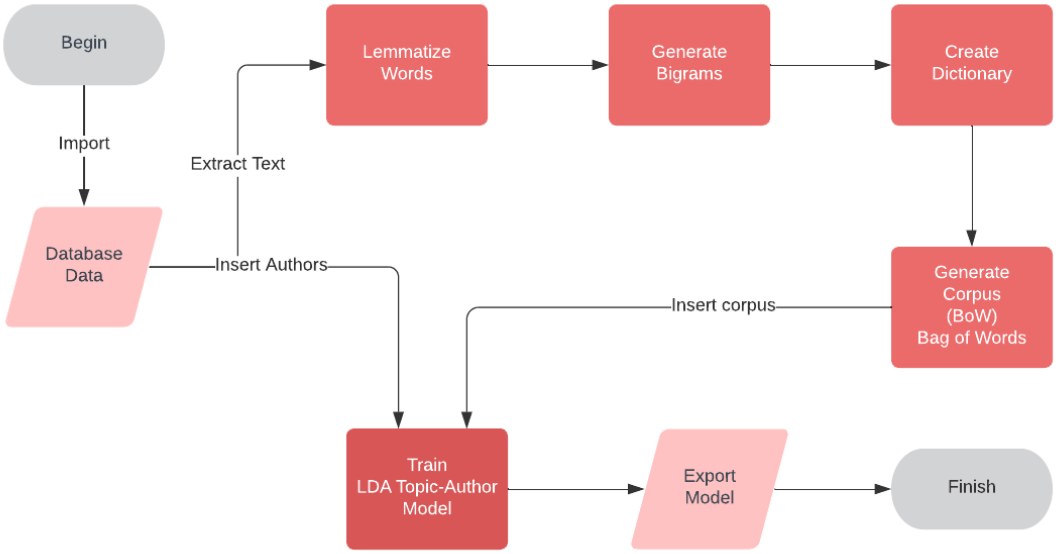


Figure 4.3: LDA Model training procedure.

4.3. Implementation of modified SVE1F algorithm

This section will discuss relevant implementation details regarding the modified version of the SVE1F algorithm [6], presented in Chapter 3. The objective of modified SVE1F is to compute the $\mathcal{ND}$ subset

given a time distribution and a set of restrictions over the scoring functions. However, applying the theoretical concepts used as a foundation for flexible skylines is not trivial, especially when it comes to computations related to $\mathcal{F}$ which in most cases is an infinite set of scoring functions over $\mathbb{R}^n$.

SVE1F was described in Chapter 2, see Algorithm 2.1. The main subprocesses that conform the algorithm will be described in detail. Starting by providing a description of the algorithm's input in this implementation.

Input description

The algorithm requires the following input variables to perform the computation of time-aware flexible skyline queries.

Variable	Type	Description
r	List[tuple]	List of tuples to be evaluated.
N	Integer	Number of attributes per tuple, time is not considered.
TD	List[function]	Time distribution.
|C|	Integer	Number of constraints imposed on $\mathcal{M}$.
A	Numpy matrix (|C| x N+1)	Matrix of weight coefficients.
operators	Numpy vector (|C|)	Matrix of operators.

Table 4.3: SVE1F algorithm input variables.

Table 4.3 presents the input variables required to perform the calculation of $W(C)$ vertices, which is one of the most complex parts of the algorithm. To better understand this process, it is important to describe the related input variables. |C| is the number of constraints imposed on $\mathcal{M}$, which is the set of scoring functions, and it determines the number of rows in matrix A as well as the length of the "operators" vector. These two variables are used to codify the constraints imposed on $\mathcal{M}$. For example, consider the following set of constraints in $\mathbb{R}^3$, $C = \{w_1 \leq w_2, w_1 \geq w_3 + 0.5\}$, the equivalent algorithm input is:

$$\begin{matrix} & w_1, w_2, w_3, cte \\ A = \end{matrix} \begin{bmatrix} 1 & -1 & 0 & 0 \\ 1 & 0 & -1 & 0.5 \end{bmatrix} \; operators = \begin{bmatrix} \leq \\ \geq \end{bmatrix}$$

The matrices presented above are a representation of the following equation system:

$$\begin{bmatrix} 1 & -1 & 0 \\ 1 & 0 & -1 \end{bmatrix} \begin{bmatrix} \leq \\ \geq \end{bmatrix} \begin{bmatrix} 0 \\ 0.5 \end{bmatrix}$$

$\mathcal{W}(C)$ vertices calculation

SVE1F bases important processes, such as tuple ordering and dominance region calculation, on a set of points $V = W^1, ..., W^q \in \mathbb{R}^N$ which are the vertices of $\mathcal{W}(C) = \{W \in \mathcal{W} : C(\mathcal{W}) = true\}$. It is important to remark that C, apart from considering the input constraints, always contains the following condition by default:

$$1 = \sum_{i=1}^{n} w_i. \tag{4.1}$$

In order to calculate the vertices, all constraints vector (C) inequalities must be treated as equalities. Then, if the number of variables is N, the algorithm will produce all possible combinations of N equalities from C and solve the linear system they produce, unless the matrix is singular. This will generate a set of candidate points in $\mathbb{R}^N$. Then each point has to fulfill all of the conditions in C, if that is the case it will be included in the set of vertices. The algorithm to compute them is presented in Algorithm 4.1.

input : A, b, $Operators$
output: $W^1, ..., W^q$

```
1  combinations ← ProduceCombinations( A, N );
2  for C in combinations do
3      if IsSingular( C ) then
4          continue;
5      end
6      t ← SolveLinearSystem( C );
7      if CheckConditions( t, A, b, Operators ) then
8          V ← Add( t );
9      end
10 end
11 return V;
```

Algorithm 4.1: Algorithm to compute V set.

Obtaining the centroid is a simple task once the set of vertices has been computed since it is their average coordinate by coordinate. The centroid vector $CE = (ce_1, ..., ce_N)$ is computed using the following equation:

$$ce_i = \frac{1}{|V|} \sum_{l=1}^{|V|} W_i^l. \tag{4.2}$$

The centroid is then used to order the tuples in r as described in Equation 3.8.

$\mathcal{ND}$ set generation

This subprocess of the algorithm ranges from line 6 to 12 in Algorithm 2.1 and is responsible for generating the $\mathcal{ND}$ set. The ordering performed over the set of tuples guarantees that the first tuple in the loop always belongs to the $\mathcal{ND}$ set. Afterwards, a tuple will only be included in the $\mathcal{ND}$ set if it is not dominated and not $\mathcal{F}-$dominated by any of the tuples already in the $\mathcal{ND}$ set.

Regarding $\mathcal{F}-$dominance, the proposed inequalities over the virtualized database will be used, see Equation 3.10. The following piece of code performs the checks required to see if a tuple s is dominated in the classic sense of skyline queries by another tuple t, see Equation 2.1:

Code 4.1: Skyline operator dominance function. It returns true if t has equal or lower values than s in all attributes but at least one is strictly lower.

```
1  def dominates(self, t, s):
2      dominates = False
3      for i in range(self.dim):
4          t_i, s_i = t[i], s[i]
5          if t_i > s_i:
6              return False
7          if t_i < s_i:
8              dominates = True
9      return dominates
```

Observe that time variables are not considered in Code 4.1. This is due to the monotone property of scoring functions, if all the tuple attributes are equal or better than the attributes of another tuple, the score of the first tuple will be equal or better than the score of the second tuple, independently of the time distribution.

5

Results

After designing and implementing the modified version of the SVE1F algorithm, the test bed system will be used to obtain metrics. Going forward, several aspects of the algorithm will be analyzed, such as the behavior of the $\mathcal{ND}$ set when different parameters are used, the alterations caused by the use of time distributions adding context to the queries, and the time complexity of the algorithm.

All the tables presented in this section will contain data directly obtained from the implemented test bed system, given the nature of the algorithm, there are several parameters that need to be set to perform a query. To avoid confusion, the parameters that will remain constant throughout a table will be indicated in its caption.

Some of the tables will make use of calculated parameters such as arithmetic means, the raw data to compute those metrics will be provided in the Appendix.

5.1. $\mathcal{ND}$ set behavior

The main objective of this project is to provide a data querying system capable of filtering data using time as a form of context. To achieve this goal, it is paramount to study how different time distributions affect the $\mathcal{ND}$ set. To efficiently measure the similarity between two obtained $\mathcal{ND}$ sets, the **intersection coefficient** will be used.

Given two queries performed over the same database, consider their respective $\mathcal{ND}$ sets, $\mathcal{ND}_1$ and $\mathcal{ND}_2$. The **intersection coefficient** between both of them is defined as:

$$IC(\mathcal{ND}_1, \mathcal{ND}_2) = \frac{|\mathcal{ND}_1 \cap \mathcal{ND}_2|}{|\mathcal{ND}_1 \cup \mathcal{ND}_2|}. \tag{5.1}$$

This metric delivers values close to one when a large portion of elements is shared between both sets. On the other hand, a low coefficient indicates that few elements are shared between both sets.

5.1.1. Evaluation of r size, conditions vector and number of parameters

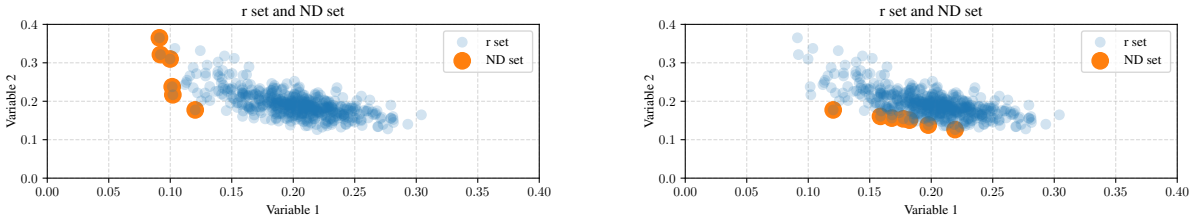
To set the groundwork for the following sections, the behavior of the $\mathcal{ND}$ set will be studied when the basic parameters associated with normal flexible skyline queries vary. Table 5.1 presents the results obtained when computing $\mathcal{ND}$ cardinal for tuple datasets of size $|r|$ and different numbers of considered attributes (N).

N	$\|r\|$ = 500	$\|r\|$ = 1000	$\|r\|$ = 2000	$\|r\|$ = 5000	$\|r\|$ = 10000	$\|r\|$ = 20000	$\|r\|$ = 50000
$N = 2$	6	5	7	7	2	4	7
$N = 3$	14	14	22	30	25	14	28
$N = 4$	45	60	95	129	125	131	213
$N = 5$	60	74	125	153	194	187	364
$N = 6$	250	450	724	1114	1648	2082	3135
$N = 7$	363	685	1029	1966	3136	4138	6843

Table 5.1: Cardinal of $\mathcal{ND}$ subset depending on r cardinal and N. Problem with condition $\{w_1 \geq w_2\}$.

Both factors seem to have a positive correlation with $\mathcal{ND}$ cardinal. This is an expected result, especially regarding N, since increasing the number of parameters implies having to fulfill a larger number of inequalities, such as those presented in Equation 3.9, to dominate a tuple. Thus, the cardinal of $\mathcal{ND}$ is naturally expected to increase.

Moreover, it is relevant to consider the possible impact of the conditions vector C in the outcome of a query. This effect is much more reliant on the data distribution; however, to illustrate the expected changes, two plots representing the $\mathcal{ND}$ set and the whole r tuple space will be presented.



(a) $\mathcal{ND}_1$ where $C = \{w_1 \geq w_2\}$

(b) $\mathcal{ND}_2$ where $C = \{w_1 \leq w_2\}$

Figure 5.1: Plots of $\mathcal{ND}$ subset and r using conditions $\{w_1 \geq w_2\}$ and $\{w_1 \leq w_2\}$, $N = 2$, $|r| = 500$.

Visualizing both scatter plots, it becomes clear that when w_1 is prioritized over w_2, the $\mathcal{ND}$ the set becomes "vertical" due to the fact that any lower values in Variable 1 become increasingly important to avoid being dominated, even if there are abrupt differences in Variable 2 and vice versa. An important aspect to notice is that not many elements seem to simultaneously belong to both $\mathcal{ND}$ subsets. As a

matter of fact the IC between both sets is: $IC(\mathcal{ND}_1, \mathcal{ND}_2) = \frac{1}{12} = 0.0833$.

5.1.2. Evaluation of time distributions effect on $\mathcal{ND}$

The following section presents data related to the impact of time distributions on the $\mathcal{ND}$ subset. The following table shows the values of the cardinal of the $\mathcal{ND}$ subset for $N = 2$ and different values of $|r|$, but using four different time distributions, $\phi \equiv \phi(0, \frac{1}{2}) + \frac{1}{2}$.

Time distribution	$\|r\|$ = 500	$\|r\|$ = 1000	$\|r\|$ = 5000	$\|r\|$ = 10000	$\|r\|$ = 20000	$\|r\|$ = 50000
[1,1]	6	5	7	2	4	7
[ϕ,1]	2	3	2	1	2	5
[1,ϕ]	12	6	8	3	7	9
[ϕ,ϕ]	3	4	3	1	4	6

Table 5.2: Cardinal of $\mathcal{ND}$ subset depending on r cardinal and time distribution. Problem with condition $\{w_1 \geq w_2\}$.

Table 5.1 shows that applying time distributions to the problem does not increase the cardinal of non dominated tuples. This is theoretically expected since the number of inequalities that must be satisfied to dominate a tuple is not affected by time distributions. Nonetheless, time distributions are expected to change the composition of the $\mathcal{ND}$ subset since time-aware skyline queries modify the coefficients in the inequalities previously mentioned, Equation 3.9.

To verify this hypothesis, the test bed will be used to perform $\mathcal{ND}$ set calculations using different time distributions. Since specific data sets can introduce bias in the results, the test bed will use a set of eight variables and compare them in groups of two to reduce error. For each pair of attributes, the system will calculate the $\mathcal{ND}$ set using three different time distributions and obtain the IC between the resulting sets. The results are shown in Table 5.3 where the average IC between distributions is shown. Raw data used to compute the coefficients are provided in Table A.1.

Time distribution	[1,1]	[$1 + x/2$,1]	[1,$1 + x/2$]
[1,1]	1	0.6339	0.7465
[$1 + x/2$,1]	-	1	0.5376
[1,$1 + x/2$]	-	-	1

Table 5.3: Intersection coefficient matrix depending on time distributions. Queried parameters $|r| = 50000$, $N = 2$, condition $\{w_1 \geq w_2\}$.

The intersection coefficients indicate that significant changes in the composition of $\mathcal{ND}$ take place when modifying the time distribution. As Table 5.1 indicates, the cardinal of $\mathcal{ND}$ subset is higher for $N = 3$, it is relevant to compute the intersection coefficient table for a higher dimension in order to see whether a higher $\mathcal{ND}$ cardinal can affect the intersection coefficients. Raw data used to produce this

table are shown in Table A.2 and Table A.3.

Time distribution	[1,1,1]	$[1+x/2$,1,1]	[1,$1+x/2$,1]	[1,1,$1+x/2$]
[1,1,1]	1	0.6755	0.7209	0.9006
[$1+x/2$,1,1]	-	1	0.5307	0.6378
[1,$1+x/2$,1]	-	-	1	0.6734
[1,1,$1+x/2$]	-	-	-	1

Table 5.4: Intersection coefficient matrix depending on time distributions. Queried parameters $|r| = 50000$, $N = 3$, condition $\{w_1 \geq w_2\}$.

The average of all coefficient rates shown in Table 5.3 is 0.6403, while the average of all coefficient rates shown in Table 5.4 is 0.7389. This may indicate that as attribute number increases, the impact of time distributions on the composition of the $\mathcal{ND}$ subset might decrease, nevertheless, more tests would be needed to confirm this hypothesis.

5.2. SVE1F algorithm time performance

This section focuses on evaluating the time performance of the modified SVE1F algorithm. Based on the theoretical background of the flexible skyline operator and the algorithm itself, see [2], the following factors will be considered:

- Number of tuples in the analyzed set.
- Number of variables per tuple.
- Number of elements in $\mathcal{ND}$.
- Number of $W(C)$ vertices ($|V|$).

Before providing the experimental results, it is important to briefly explain the algorithm's complexity. As it can be seen in Algorithm 2.1, the algorithm is composed by two main subprocesses, the first one is comprises the computation of $W(C)$ vertices and the sorting of the tuples in the database using the centroid. This process takes $O(r)$ time, where r is the table of tuples. Due to the usually low number of conditions imposed on the query, calculating V does take a negligible amount of time.

The second subprocess is the generation of the $\mathcal{ND}$ subset, which has complexity $O(r \cdot |\mathcal{ND}|)$, since it checks dominance relationships between each tuple and all the elements in $\mathcal{ND}$. However, due to the fact that once dominated by an element in $\mathcal{ND}$, a tuple will not be considered again, plus the usually low number of elements in $\mathcal{ND}$ compared to $|r|$, this process real complexity may be closer to $O(r)$. Each measure will be repeated five times to reduce error, see Appendix Table B.1.

As we can see in Figure 5.2, there seems to be a linear relationship between the execution time and the number of tuples in the database, as it was theoretically predicted. Of course, there will be small

N	$\|r\|$	$\|\mathcal{ND}\|$	Time (s)
2	5000	5	0.0990
2	10000	1	0.1684
2	50000	6	1.1089
2	100000	8	1.7082
2	200000	10	3.4462

Table 5.5: Time-aware skyline queries time cost (s), and $|N|$ for different r cardinals. Queried parameters $N = 2$, condition $\{w_1 \geq w_2\}$.

variations due to the fact that the number of elements in $\mathcal{ND}$ is not constant, which affects the number of dominance comparisons performed. The cardinal of V is not relevant in this case since the query performed is always the same, thus the number of elements in V remains constant.

Figure 5.2 shows that, as the number of tuples increases, the time cost of the algorithm does the same linearly. This confirms the theoretical expectation of linear growth regarding $|r|$.

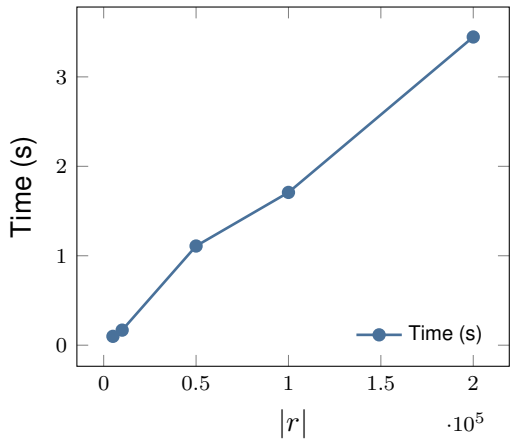


Figure 5.2: Time cost in seconds of modified SVE1F algorithm depending on $|r|$. Data obtained from Table 5.5.

The number of compared attributes (N) is expected to increase complexity since, as Table 5.1 shows, it is related to the growth of the $\mathcal{ND}$ subset. The other metric affected by the increase in N is the number of vertices obtained after computing $W(C)$ intersections. This may also be a factor driving time cost increase in the algorithm, since most calculations require to operate with the set of $W(C)$ vertices.

The theoretical complexity of the algorithm states that if $|\mathcal{ND}|$ tends to $|r|$, which represents the worst case in terms of computational cost, then the algorithm will have a complexity of $O(|r|^2)$, in terms of the r set growth.

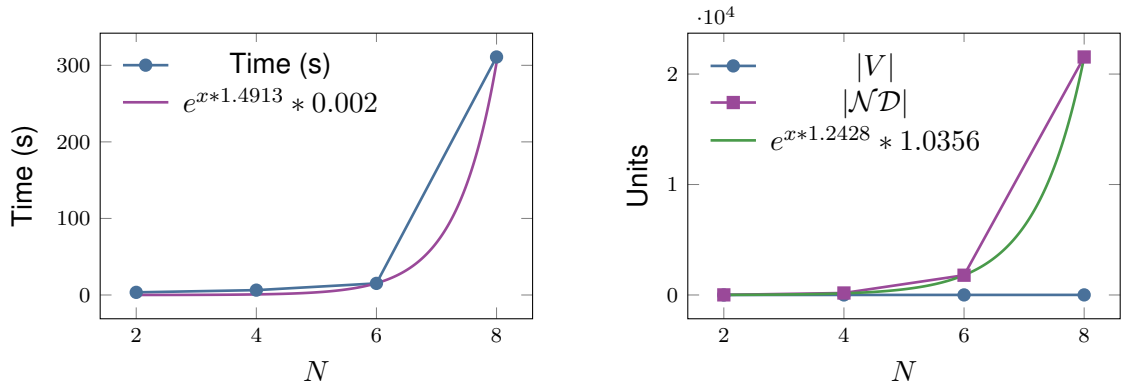
Results presented in Figure 5.3 seem to provide two clear insights. First, that the cardinal of $W(C)$ vertices ($|V|$) is not correlated to the algorithm's time cost, since $|V|$ is linearly related to the number

N	$\|r\|$	$\|\mathcal{ND}\|$	$\|V\|$	Time (s)
2	200000	10	2	3.4462
4	200000	180	4	6.3353
6	200000	1789	6	15.1596
8	200000	21547	8	310.8934

Table 5.6: Time-aware skyline time cost, $|\mathcal{ND}|$ and $|V|$ depending on the number of considered variables N. Queried parameters $|r| = 200000$, condition $\{w_1 \geq w_2\}$.

of variables, but the time cost is exponentially related to them. Second, the cardinal of the $\mathcal{ND}$ subset seems to scale exponentially with the number of variables, which is consistent with the theoretical perspective. Pondering the fact that the fit curves show reasonably similar exponential parameters, it is coherent to assume that time complexity and $|\mathcal{ND}|$ cardinal are exponentially correlated.

To conclude this section, the algorithm's time cost is exponentially correlated to the number of attributes analyzed and the cardinal of the $\mathcal{ND}$ subset. However, the number of tuples in the database has been found to be linearly correlated with the time cost. This means that the algorithm is effectively $O(|r|)$, which can be considered a positive result since the database size is expected to be much larger than the number of attributes analyzed.



(a) Execution time (s) depending on N.

(b) $|\mathcal{ND}|$and $|V|$ values depending on N.

Figure 5.3: Execution time (s), $|\mathcal{ND}|$ and $|V|$ values as the number of attributes (N) increases. Queried parameters $|r| = 200000$, condition $\{w_1 \geq w_2\}$.

6

Conclusions and future work

This bachelor's thesis has studied the use of flexible skylines as a data filtering method, using time as a form of context; aiming to leverage the rising importance of time-related data in today's databases, usually in the form of time series. Many of the data filtering techniques related to flexible skyline either do not consider temporal variables or propose a solution that is difficult to implement or computationally unaffordable.

This thesis proposes a solution to this problem, by presenting an adapted version of the flexible skyline operator capable of generating time-aware flexible skyline queries. To achieve this, the weights of the set of infinite functions that usually constitute the core of flexible skyline queries have been modified using time distributions. In addition, an algorithm dedicated to computing flexible skyline queries has been adapted to generate these new queries.

To provide a real implementation of the concept proposed, a test bed system has been developed retrieving and preprocessing data to apply topic detection based on LDA, but implementing features such as author-topic distributions and biterms to tailor the system to the specific characteristics of the used dataset. The distributions obtained from the system have been used to perform tests on the modified version of the flexible skyline algorithm SVE1F.

6.1. Conclusions

The implemented system has behaved as expected, being able to correctly process data and perform queries that align with the theoretical expectations.

The time-aware flexible skyline algorithm is able to perform queries with few theoretical limitations, the only unavoidable error case would be the introduction of a condition vector able to produce a singular matrix, and even in that case, the algorithm is able to detect it and execute regardless.

The time-cost analysis has been reasonably positive; it fulfills the predicted complexity that would be $O(|r|)$ for cardinal increments in the tuple set and exponential for the increment in the number of analyzed variables, as it can be seen in Figure 5.3. Overall, it does not add additional complexity to the

original algorithm presented in [6], which is one of the objectives of this thesis.

In terms of the output produced, metrics, like those presented in Table 5.3, show that the inclusion of time as a context variable is impactful on the output generated by the queries. For example, in Table 5.4, an average of 31.02% of the tuples in $\mathcal{ND}$ change due to the use of a different time distribution, which is a more than satisfactory result. In addition, the functions in the time distribution can be tweaked to lower or increase the impact of time in the $\mathcal{ND}$ set, allowing one to tailor the output and control the cardinality of the obtained results.

6.2. Future work

The developed system has accomplished the objectives set for this thesis. Nonetheless, there are several aspects that can be addressed in the future:

- The set $\mathcal{PO}$ is the other specific output produced by flexible skylines. Finding an algorithm involving time context to generate it and checking that it maintains the theoretical relationships proposed in [6] between $\mathcal{ND}$ and $\mathcal{PO}$ subsets would constitute a natural continuation to this thesis.
- The time complexity of the modified algorithm scales exponentially when the number of attributes increases. There is a solid foundation to believe that increasing the number of conditions imposed on the tuples will decrease the time cost of the algorithm due to a reduction in the number of tuples in $\mathcal{ND}$. Thus, a convenient improvement would be to implement a mixed algorithm where tuples affected by the conditions vector are processed using flexible skylines and those that are not would be processed using the original skyline algorithm.
- The algorithm still requires manual generation of the time distribution. In order to apply this methodology in larger systems, it would be convenient to design a module capable of translating time-related preferences into time distributions that can be processed by the algorithm.
- The topic detection module can be improved by integrating the topic correlation distributions presented in the CTM model. Moreover, the training set can be expanded, and the average corpus size increased to obtain better perplexity and coherence metrics.

Bibliography

[1] E. Tiakas, A. N. Papadopoulos, and Y. Manolopoulos, "Skyline queries: An introduction," in *2015 6th International Conference on Information, Intelligence, Systems and Applications (IISA)*, pp. 1–6, 2015.

[2] P. Ciaccia and D. Martinenghi, "Flexible skylines: Dominance for arbitrary sets of monotone functions," *ACM Trans. Database Syst.*, vol. 45, 12 2020.

[3] D. Che, M. Safran, and Z. Peng, "From big data to big data mining: Challenges, issues, and opportunities," in *Database Systems for Advanced Applications* (B. Hong, X. Meng, L. Chen, W. Winiwarter, and W. Song, eds.), (Berlin, Heidelberg), pp. 1–15, Springer Berlin Heidelberg, 2013.

[4] S. Borzsony, D. Kossmann, and K. Stocker, "The skyline operator," pp. 421 – 430, 1 2001.

[5] H. T. Kung, F. Luccio, and F. P. Preparata, "On finding the maxima of a set of vectors," *J. ACM*, vol. 22, p. 469–476, oct 1975.

[6] P. Ciaccia and D. Martinenghi, "Reconciling skyline and ranking queries," *Proc. VLDB Endow.*, vol. 10, pp. 1454–1465, 8 2017.

[7] I. Ilyas, W. Aref, and A. Elmagarmid, "Supporting top-k join queries in relational databases," pp. 754–765, 09 2003.

[8] D. Haws, "Quicklexsort: An efficient algorithm for lexicographically sorting nested restrictions of a database," 10 2013.

[9] P. Ciaccia and D. Martinenghi, "Beyond skyline and ranking queries: Restricted skylines (extended abstract)," 2018.

[10] C. Böhm, F. Fiedler, A. Oswald, C. Plant, and B. Wackersreuther, "Probabilistic skyline queries," in *Proceedings of the 18th ACM Conference on Information and Knowledge Management*, CIKM '09, (New York, NY, USA), p. 651–660, Association for Computing Machinery, 2009.

[11] K. Chakrabarti, E. Keogh, S. Mehrotra, and M. Pazzani, "Locally adaptive dimensionality reduction for indexing large time series databases," *ACM Trans. Database Syst.*, vol. 27, p. 188–228, jun 2002.

[12] Q. Li, B. Moon, and I. Lopez, "Skyline index for time series data," *IEEE Transactions on Knowledge and Data Engineering*, vol. 16, no. 6, pp. 669–684, 2004.

[13] T. Pavlidis, "Waveform segmentation through functional approximation," *IEEE Transactions on Computers*, vol. C-22, no. 7, pp. 689–697, 1973.

[14] H. Shatkay and S. Zdonik, "Approximate queries and representations for large data sequences," vol. 536-545, pp. 536–545, 01 1996.

[15] M. Allahyari and K. Kochut, "Automatic topic labeling using ontology-based topic models," pp. 259–264, 12 2015.

[16] R. Ibrahim, A. Elbagoury, M. S. Kamel, and F. Karray, "Tools and approaches for topic detection from twitter streams: survey," *Knowledge and Information Systems*, vol. 54, 03 2018.

[17] N. Panahi, M. G. Shayesteh, S. Mihandoost, and B. Zali Varghahan, "Recognition of different datasets using pca, lda, and various classifiers," in *2011 5th International Conference on Application of Information and Communication Technologies (AICT)*, pp. 1–5, 2011.

[18] Y.-W. Seo and K. Sycara, "Text clustering for topic detection," 02 2004.

[19] A. K. Jain and R. C. Dubes, *Algorithms for Clustering Data*. USA: Prentice-Hall, Inc., 1988.

[20] C. M. Bishop, *Neural Networks for Pattern Recognition*. USA: Oxford University Press, Inc., 1995.

[21] P.-M. Difrancesco, D. Bonneau, and D. Hutchinson, "The implications of m3c2 projection diameter on 3d semi-automated rockfall extraction from sequential terrestrial laser scanning point clouds," *Remote Sensing*, vol. 12, p. 1885, 06 2020.

[22] R. Duda, P. Hart, and D. G.Stork, *Pattern Classification*, vol. xx. 01 2001.

[23] D. M. Blei, "Probabilistic topic models," *Commun. ACM*, vol. 55, pp. 77–84, 4 2012.

[24] T. Moon, "The expectation-maximization algorithm," *IEEE Signal Processing Magazine*, vol. 13, no. 6, pp. 47–60, 1996.

[25] D. M. Blei, A. Y. Ng, and M. I. Jordan, "Latent dirichlet allocation," *J. Mach. Learn. Res.*, vol. 3, pp. 993–1022, 3 2003.

[26] D. M. Blei and J. D. Lafferty, "Dynamic topic models," pp. 113–120, Association for Computing Machinery, 2006.

[27] X. Yan, J. Guo, Y. Lan, and X. Cheng, "A biterm topic model for short texts," in *Proceedings of the 22nd International Conference on World Wide Web*, WWW '13, (New York, NY, USA), p. 1445–1456, Association for Computing Machinery, 2013.

[28] T. L. Griffiths and M. Steyvers, "Finding scientific topics," *Proceedings of the National Academy of Sciences*, vol. 101, no. suppl_1, pp. 5228–5235, 2004.

[29] D. Blei and J. Lafferty, "Correlated topic models. inweiss, y., schölkopf, b., and platt, j., editors," *Advances in neural information processing systems*, vol. 18, 2006.

[30] P. S. Foundation, "Gensim." https://radimrehurek.com/gensim/. Accesed: 2023-25-01.

[31] N. Project, "Natural language toolkit." https://www.nltk.org/. Accesed: 2023-13-02s.

[32] D. Palau-Sampio and P. Sánchez-García, "Digital resources in the current journalistic narrative: Uses and limitations of hypertext, multimedia and interactivity," *Comunicacion y Sociedad*, vol. 33, pp. 1–16, 04 2020.

[33] M. Rosen-Zvi, T. Griffiths, M. Steyvers, and P. Smyth, "The author-topic model for authors and documents," pp. 487–494, AUAI Press, 2004.

Appendix

A

$\mathcal{ND}$ TIME DISTRIBUTIONS COEFFICIENT RATES

The following matrixes contain the coefficient rates of $\mathcal{ND}$ when comparing identical queries with different time distributions. In Table A.1, intersection coefficients are obtained for the following time distribution comparisons:

- C1 $\equiv$ ([1,1] ,[$1 + x/2$,1]).
- C2 $\equiv$ ([1,1] ,[1,$1 + x/2$]).
- C3 $\equiv$ ([$1 + x/2$,1] ,[1,$1 + x/2$]).

In Table A.2 and Table A.3 the intersection coefficients are obtained for the following time distribution comparisons:

- C1 $\equiv$ ([1,1,1] ,[$1 + x/2$,1,1]).
- C2 $\equiv$ ([1,1,1] ,[1,$1 + x/2$,1]).
- C3 $\equiv$ ([1,1,1] ,[1,1,$1 + x/2$]).
- C4 $\equiv$ ([$1 + x/2$,1,1] ,[1,$1 + x/2$,1]).
- C5 $\equiv$ ([$1 + x/2$,1,1] ,[1,1,$1 + x/2$]).
- C6 $\equiv$ ([1,$1 + x/2$,1] ,[1,1,$1 + x/2$]).

Time distribution	C1	C2	C3
(0, 1)	0.7692	0.75	0.5384
(0, 2)	0.5	0.625	0.5
(0, 3)	1.0	1.0	1.0
(0, 4)	0.5	0.625	0.3333
(0, 5)	0.8	1.0	0.8
(0, 6)	0.4	0.5454	0.1538
(0, 7)	0.5	1.0	0.5
(1, 2)	0.3181	0.3181	0.0909
(1, 3)	0.2	1.0	0.2
(1, 4)	0.3333	0.45	0.1923
(1, 5)	1.0	1.0	1.0
(1, 6)	0.2666	0.3888	0.1578
(1, 7)	0.625	0.4545	0.3636
(2, 3)	1.0	1.0	1.0
(2, 4)	0.75	1.0	0.75
(2, 5)	1.0	1.0	1.0
(2, 6)	0.6666	1.0	0.6666
(2, 7)	0.6666	0.6666	0.5
(3, 4)	0.4444	0.5	0.3333
(3, 5)	0.6	0.6666	0.4285
(3, 6)	0.6	0.5	0.25
(3, 7)	0.6666	1.0	0.6666
(4, 5)	0.25	0.2	0.1111
(4, 6)	1.0	0.6666	0.6666
(4, 7)	1.0	1.0	1.0
(5, 6)	0.1428	0.5454	0.1
(5, 7)	0.75	1.0	0.75
(6, 7)	1.0	1.0	1.0

Table A.1: Raw data: Intersection coefficient matrix depending on time distributions. Queried parameters $|r| = 50000$, $N = 2$, condition $\{w_1 \geq w_2\}$.

Time distribution	C1	C2	C3	C4	C5	C6
(0, 1, 2)	0.8548	0.75	0.8615	0.6712	0.7465	0.6623
(0, 1, 3)	0.8	0.84	0.9583	0.7308	0.7692	0.8077
(0, 1, 4)	0.7794	0.6933	0.9677	0.6203	0.7571	0.6753
(0, 1, 5)	0.8	0.8667	1.0	0.6875	0.8	0.8667
(0, 1, 6)	0.7179	0.6889	0.7955	0.587	0.5833	0.6346
(0, 1, 7)	0.75	0.76	0.913	0.5823	0.7123	0.7037
(0, 2, 3)	0.6078	0.6296	0.8148	0.4386	0.5862	0.5312
(0, 2, 4)	0.625	0.75	1.0	0.6429	0.625	0.75
(0, 2, 5)	0.5806	0.4667	0.7778	0.3409	0.5833	0.3962
(0, 2, 6)	0.5385	0.7222	1.0	0.3704	0.5385	0.7222
(0, 2, 7)	0.5125	0.6267	0.7528	0.3625	0.4896	0.4845
(0, 3, 4)	0.7297	0.7353	0.8824	0.575	0.6585	0.7027
(0, 3, 5)	0.7826	0.7255	0.9773	0.6154	0.766	0.7115
(0, 3, 6)	0.6923	0.8696	0.8846	0.7083	0.6207	0.7692
(0, 3, 7)	0.8667	0.8125	0.9333	0.7059	0.9333	0.7647
(0, 4, 5)	0.6607	0.7321	0.7969	0.5172	0.6061	0.5942
(0, 4, 6)	0.4516	0.6	0.8929	0.3871	0.4118	0.7
(0, 4, 7)	0.7059	0.6471	0.8333	0.4727	0.7143	0.55
(0, 5, 6)	0.75	0.825	0.9474	0.625	0.7174	0.7857
(0, 5, 7)	0.75	0.8571	1.0	0.6429	0.75	0.8571
(0, 6, 7)	0.5645	0.6441	0.85	0.3529	0.5588	0.5588
(1, 2, 3)	0.4789	0.4691	0.9	0.3243	0.4737	0.4318
(1, 2, 4)	0.5094	0.561	0.975	0.2222	0.5	0.5854
(1, 2, 5)	0.4255	0.4603	0.9778	0.2456	0.4167	0.4531
(1, 2, 6)	0.3205	0.5246	0.9821	0.1781	0.3165	0.541
(1, 2, 7)	0.6047	0.5862	0.9767	0.375	0.5909	0.5763
(1, 3, 4)	0.6512	0.7941	0.8857	0.5556	0.5957	0.8571
(1, 3, 5)	0.6667	0.7949	1.0	0.5556	0.6667	0.7949
(1, 3, 6)	0.5	0.8947	0.95	0.4231	0.4815	0.85
(1, 3, 7)	0.52	0.7708	0.9787	0.4375	0.5098	0.7551
(1, 4, 5)	0.5506	0.6042	0.8696	0.3333	0.5789	0.537
(1, 4, 6)	0.3208	0.6818	0.9286	0.2222	0.3036	0.75
(1, 4, 7)	0.5152	0.5941	0.9149	0.3495	0.5644	0.5505
(1, 5, 6)	0.6098	0.9706	0.8293	0.5854	0.5208	0.8049
(1, 5, 7)	0.6	0.8421	1.0	0.5	0.6	0.8421
(1, 6, 7)	0.5	0.6591	0.95	0.322	0.5091	0.6304

Table A.2: Raw data: Intersection coefficient matrix depending on time distributions. Queried parameters $|r| = 50000$, $N = 3$, condition $\{w_1 \geq w_2\}$.

Time distribution	C1	C2	C3	C4	C5	C6
(2, 3, 4)	0.7333	0.7333	0.6667	0.5556	0.6842	0.5238
(2, 3, 5)	0.9394	0.7838	1.0	0.7368	0.9394	0.7838
(2, 3, 6)	1.0	0.6875	0.7857	0.6875	0.7857	0.5789
(2, 3, 7)	0.6667	0.5625	0.8182	0.4211	0.5714	0.6875
(2, 4, 5)	0.8846	0.8846	0.8519	0.9259	0.7667	0.7667
(2, 4, 6)	1.0	1.0	1.0	1.0	1.0	1.0
(2, 4, 7)	0.9048	0.9048	0.7917	0.9091	0.7308	0.7308
(2, 5, 6)	0.775	0.5882	0.9231	0.4815	0.8049	0.5556
(2, 5, 7)	0.5455	0.75	1.0	0.4167	0.5455	0.75
(2, 6, 7)	0.9032	0.7778	0.6818	0.7027	0.6222	0.5918
(3, 4, 5)	0.6129	0.6667	0.873	0.5517	0.6364	0.5882
(3, 4, 6)	0.4231	0.7037	1.0	0.2963	0.4231	0.7037
(3, 4, 7)	0.76	0.6719	0.8929	0.5574	0.6786	0.6143
(3, 5, 6)	0.9091	0.8158	0.9412	0.7436	0.8571	0.775
(3, 5, 7)	0.6316	0.8	1.0	0.5217	0.6316	0.8
(3, 6, 7)	0.875	0.775	0.907	0.7	0.7955	0.7045
(4, 5, 6)	0.7333	0.5526	0.8929	0.4524	0.7188	0.5122
(4, 5, 7)	0.6538	0.5625	0.9524	0.4324	0.6296	0.5938
(4, 6, 7)	0.931	0.9643	0.6667	0.8966	0.6279	0.6429
(5, 6, 7)	0.6522	0.7381	0.8333	0.4681	0.7143	0.62

Table A.3: Raw data: Coefficient rate matrix depending on time distributions. Queried parameters $|r| = 50000$, $N = 3$ condition $\{w_1 \geq w_2\}$.

B

Time measures for SVE1F

N	$\|r\|$	$\|\mathcal{ND}\|$	Time (s)
2	5000	5	0.0985
2	5000	5	0.0979
2	5000	5	0.1010
2	5000	5	0.0985
2	5000	5	0.0989
2	10000	1	0.2076
2	10000	1	0.1565
2	10000	1	0.1674
2	10000	1	0.1510
2	10000	1	0.1586
2	50000	6	1.0962
2	50000	6	1.1154
2	50000	6	1.1029
2	50000	6	1.1370
2	50000	6	1.0924
2	100000	8	1.7079
2	100000	8	1.7077
2	100000	8	1.7091
2	100000	8	1.7149
2	100000	8	1.6990
2	200000	10	3.4840
2	200000	10	3.4190
2	200000	10	3.4174
2	200000	10	3.4283
2	200000	10	3.4821

Table B.1: Raw data: Time-aware skyline queries time cost (s), and $|N|$ for different r cardinals. Queried parameters $N = 2$, condition $\{w_1 \geq w_2\}$.

N	$\|r\|$	$\|\mathcal{ND}\|$	Time (s)
2	200000	10	3.4840
2	200000	10	3.4190
2	200000	10	3.4174
2	200000	10	3.4283
2	200000	10	3.4821
4	200000	180	6.2093
4	200000	180	6.2312
4	200000	180	6.2131
4	200000	180	6.6192
4	200000	180	6.4036
6	200000	1789	15.0634
6	200000	1789	15.5916
6	200000	1789	15.0349
6	200000	1789	15.0734
6	200000	1789	15.0346
8	200000	21547	302.7095
8	200000	21547	310.4929
8	200000	21547	308.2973
8	200000	21547	309.9050
8	200000	21547	323.0623

Table B.2: Raw data: Time-aware skyline time cost and $|\mathcal{ND}|$ depending on the number of considered variables N. Queried parameters $|r| = 200000$, condition $\{w_1 \geq w_2\}$.


UÁM
Universidad Autónoma
de Madrid