Finding Interesting Subsets from Data: Pattern Mining and Skyline Query Approaches

Dominique H. Li

HABILITATION A DIRIGER DES RECHERCHES

en Informatique

Année Universitaire : 2022-2023

présentée et soutenue publiquement par

Dominique H. LI

le 12 juillet 2023

Finding Interesting Subsets from Data: Pattern Mining and Skyline Query Approaches

JURY

M. Hubert CARDOT	Professeur des universités	Université de Tours
M. Nicolas LABROCHE	Maître de conférences HDR	Université de Tours
Mme. Anne LAURENT	Professeur des universités	Université de Montpellier
M. Sofian MAABOUT	Maître de conférences HDR	Université de Bordeaux
M. Florent MASSEGLIA	Directeur de recherche	INRIA Montpellier
M. Alexandre TERMIER	Professeur des universités	Université de Rennes 1
Mme. Karine ZEITOUNI	Professeur des universités	Université Paris-Saclay

Table of Contents

Introduction

The efficient and effective discovery of interesting information from data is an important problem in many application and research domains. Presently, with the rapid development in data science and machine learning research work, many methods have been proposed. My main research interests focus on the applications and algorithms of pattern mining to machine learning and the skyline query processing problem.

I started actively working on data mining from the beginning of my Master Thesis on sequential pattern mining and its applications. After obtaining my PhD degree in 2009 from the University of Montpellier on the topic of mining unexpected sequential patterns under the direction of Professor Pascal Poncelet and Professor Anne Laurent, I did a one year post-doc at the University of Pisa in Italy under the direction of Professor Giuseppe Attardi on the application of pattern mining to the active learning in dependency parsing in natural language processing. Since I became an Associate Professor at the University of Tours from 2010, I joined the BdTln research team in the LIFAT Laboratory (former LI Laboratory) of the University of Tours. My research interests particularly focus on the applications and algorithms of pattern mining, pattern-based machine learning (since 2018), and skyline query processing (since 2019). All the results presented in this Habilitation Thesis are post-2010.

In 2010, as part of the international collaborative project PQUERY with the Universida de Federal de Uberlândia (Brazil), the Universidad de la República (Uruguay), and the University of Tours in the STIC-AmSud regional program, I started my first research participation on the application of frequent patterns. The objective of this research project is to automatically generate user profiles from relational data as interpretable rules, where the frequent pattern mining is used to find association rules to determine conditionally preference rules. With this project, Mouhamadou Saliou Diallo, (defended in 2014) is the first PhD student whom I supervised with my colleagues Arnaud Giacometti and Arnaud Soulet in the same research team BdTln on the topic of user profile construction by mining and ranking frequent patterns as preference rules [29, 44, 28].

Indeed, frequent patterns mining has been a well-studied problem for almost 30 years during almost 30 years since the introduction of the `Apriori` algorithms for mining frequent *itemset patterns* in *transactional data* [1] such as `GSP` [4], `Eclat` [148], `LCM` [126], `FP-Tree` [54], `CP-Tree` [119], or `LP-Tree` [108], and *sequential patterns* in *sequence data* [3], such as `PSP` [84], or `SPADE` [147], `PrefixSpan` [53]. However, the notion of frequent patterns is not limited to itemset patterns and sequential patterns. For instance, the definitions and problems of mining *tree patterns* and *graph patterns* are also deeply studied [128, 5, 141, 152] (which are out of my research scopes). On the other hand, to improve the efficiency of the above in-memory algorithms for frequent pattern mining, there are several directions:

- Design more efficient algorithms such as `Eclat`, `LCM`, `FP-Tree`, `PSP`, `SPADE`, and `PrefixSpan` did.
- Return a representative subset of frequent patterns, for instance, the *closed patterns* concerned by `CloSpan` [142] and `BIDE` [131], or the *pattern generators* concerned by `Gr-growth` [70] and `FEAT` [43].
- Mine frequent patterns in parallel computing with multicore processors, such as `PLCM` [89] and `ParaMiner` [90].

From 2016 to 2018, I joined the supervision of the post-doc researcher Khanh-Chuong Duong in the regional research project GIRAFON (Grands graphes: interrogation, fouille et analyse) collaborated with the LIFO Laboratory of the University of Orléans. Under our supervision, Khanh-Chuong Duong has been able to develop two new MapReduce-based algorithms [37, 38] to mine frequent itemset patterns in very large datasets. Furthermore, since 2018, I participated to the supervision of the PhD student Lamine Diop (defended in 2021) with Arnaud Giacometti, Arnaud Soulet, and Cheikh Talibouya Diop (Université Gaston Berger de Saint-Louis, Senegal), of whom the PhD thesis topic is to sample sequential patterns in order to reduce the computational time and to avoid returning a huge number of patterns. The sequential pattern sampling method [33, 35] proposed by Lamine Diop has been adapted to mine frequent patterns in distributed datasets [34, 36].

Since 2017, I formally started to focus my research on the applications of pattern mining in spatio-temporal sequence data to different machine learning problems, particularly in video leaning. First, I launched an international collaboration with the team leaded by Professor Yuqi Huang at Zhejiang University (China) and my colleague Donatello Conte in RFAI research team of the LIFAT Laboratory to identify two-phase flow videos by sequence patterns and machine

learning methods [58], which is expected to be helpful to the application of an international research funding between France and China in the future. Secondly, I have participated to the supervision of Luong Phat Nguyen (defended in 2021) with my colleagues Nicolas Ragot, Donatello Conte, and Julien Mille in the RFAI research team on trajectory extraction and on using probabilistic sequential patterns to dynamic texture video classification [92, 93]. The results obtained in the above research work show that the temporal-spatial sequence-based pattern mining can be effectively applied to the video learning problems. Besides, in my current supervision (since 2022) of the PhD student Thomas Kastner with my colleague Hubert Cardot in the RFAI research team, we have proposed a new form of sequence to simplify the representation of timestamp intervals in sequence data for anomaly learning in large datasets.

In 2019, I supervised the Master Thesis of Erasmus Mundus international student Rui Liu on skyline query processing [12], whose results have obtained the Best Master Thesis Award Erasmus Mundus 2019 in Computer Science. In the Master Thesis of Rui Liu, we have proposed a new framework `SDI` [77] to efficient skyline computation in high-dimensional datasets and we have successfully developed the algorithm `RSS` [76], the first algorithm on-top of the `SDI` framework, to maintain the skyline in streaming data. With such a success, besides pattern mining and machine learning, my research interests additionally focus on several skyline query processing problems including *full skyline queries*, *dynamic skyline queries*, *subspace skyline queries*, and *top-k skyline queries*, of which my most recent research result is to use subset queries to boost skyline computation algorithms [69].

In fact, during recent years, I deeply feel that there are several similarities between pattern mining problems and skyline query problems:

- Return the results faster. For this problem, many efficient skyline algorithms (detailed in Chapter 2) have been proposed during more than two decades, including MapReduce based ones for processing large scale datasets.

- Return less but useful results. As closed patterns and pattern generators in pattern mining, subspace skyline queries [103, 121, 104, 66], top-k skyline queries [20, 19, 55, 87], and their combination [122, 143, 127] are always actively studied.

- The applications of skyline to machine learning, which is one of the most important perspectives in my future work.

Therefore, I consider that the purpose of the pattern mining and the skyline query processing can be regarded as finding *interesting subsets* in the data, such subsets can be a set of patterns

(for pattern mining) or a set of points (for skyline query processing). Furthermore, the term *interesting* can be objective, for instance verified by machine learning methods, or subjective, for instance simply examined by domain experts.

My recent key contributions include:

- The application of patterns to visualized two-phase flow identification [58] (*Measurement*, corresponding author).
- The application of probabilistic sequence patterns to dynamic texture video classification [93] (*ICPR'22*).
- A scalable framework to efficient skyline computation in high-dimensionality domains [77] (*EDBT'20*, corresponding author).
- An efficient method of skyline maintenance in streaming data [76] (*DASFAA'20*, corresponding author).
- A subset approach to efficient skyline computation [69] (*EDBT'23*, unique author).

The rest of this HDR Thesis is organized as follows. In Chapter 1, I introduce my supervisions and research work with my colleagues on pattern mining algorithms and the applications of patterns in machine learning. My supervision and my research work on skyline computation are presented in Chapter 2. Finally, I conclude and introduces my perspectives in future research work.

Chapter 1

Pattern Mining Applications

1.1 Introduction

In this section, we present the applications of pattern mining in several machine learning problems. To make the notion of *Pattern Mining* clear in this chapter, the following formal definitions are indispensable.

Definition 1 (Itemset)**.** *Let $\mathcal{I} = \{i_1, i_2, \ldots, i_n\}$ be a finite set of n binary-valued attributes, an* item *is an attribute $i_j \in \mathcal{I}$. An* itemset *is an unordered collection $I = (i_1 i_2 \ldots i_m)$ of sorted distinct items, we have $I \subseteq \mathcal{I}$.* ■

According to the terms described in [1], a *transactional database* is a large set $\mathcal{D}$ of *transactions*, where each transaction is an itemset. If a itemset X is a subset of a transaction $T \in \mathcal{D}$, that is, $X \subseteq T$, then we say that T *supports* X. If there are an *enough* number (called *minimum support*, denoted by σ in the rest of this chapter) of transactions in a given database $\mathcal{D}$ that support an itemset X, then we say that X is a *frequent itemset*, which is firstly defined as a *pattern* in [1].

Definition 2 (Sequence)**.** *Based on Definition 1, a* sequence *is an ordered list $s = \langle I_1 I_2 \ldots I_k \rangle$ of itemsets, where I_j is an itemset.* ■

Given two sequences $s = \langle I_1 I_2 \ldots I_m \rangle$ and $s' = \langle I'_1 I'_2 \ldots I'_n \rangle$, if there exist integers $1 \leq i_1 < i_2 < \ldots < i_m \leq n$ such that $I_1 \subseteq I'_{i_1}, I_2 \subseteq I'_{i_2}, \ldots, I_m \subseteq I'_{i_m}$, then s is a *subsequence* of s', denoted as $s \sqsubseteq s'$; we also say that s *is included in* s', or s' *supports* s. Hence, similar to the definition in frequent itemsets mining, the *support* of a sequence s in a database $\mathcal{D}$ is the total number of sequences, and therefore *frequent subsequences* with respect to a given *minimum support* σ in $\mathcal{D}$ are called *sequential patterns* in [3].

In considering both *pattern* and *sequential pattern* are frequent[1] sub-structures in data, such as *tree patterns* or *graph patterns*, we use the terms *itemset pattern* and *sequence pattern* in this thesis to distinguish the original terms proposed in [1] and [3], and which allows to unify the notion of *pattern mining*.

The rest of this chapter is organized as follows. IIn Section 1.2, we introduce the construction of user profile by pattern mining and ranking rules generated by patterns. Section 1.3 presents the application of sequence patterns to two-phase flow pattern identification in video data. Section 1.4 presents the application of sequence patterns to the dynamic texture classification problem in video learning. The perspectives are presented in Section 1.5.

1.2 Profile Construction

The enhancing database systems with context-aware preference queries has been attracting a lot of interests on constructing *user profiles*, which has a variety of applications ranging from e-commerce to personalized search engines where the prediction of user preferences is essential in the system model design. The user preference prediction can be considered as a learning problem. Several important research work has been dedicated to this topic, including research on the development of:

- Powerful frameworks for preference modeling and reasoning [14, 136].
- Preference query languages with high declarative and expressive power for personalized database applications [60, 23, 2, 106].

However, little work has been dedicated so far to the topic of *preference elicitation*, that is, in the way user preferences are obtained for building the profile. In this section, we present our rule-based approach `ProfMiner` that consists in extracting a user profile from a set of user preference samples. In our setting, a *profile* is specified by a set of *contextual preference rules* verifying properties of soundness and conciseness [29, 44, 28].

1.2.1 Preference Elicitation

Elicitation of preferences consists basically in providing the user a way to inform the preferences on tuples belonging to a dataset, with a minimal effort for a user. It can be achieved by following different strategies: (1) by using a query interface where users are asked to express their

[1]There are many other criteria that can replace the measure *frequent* to find interesting sub-structures in data.

preferences [14], or (2) by capturing implicit user's choices and applying preference mining algorithms [57]. The first alternative is not efficient since the users in general are not able to express their preferences in an exact and consistent way, hence, we focus on the second alternative for contextual preference elicitation.

We assume that our data is constituted by pairwise comparisons and do not discuss the way the user informed the choices knowing that different strategies can be applied [116]. Our approach also assumes that the comparison of the *pairs of tuples* can express user preferences with respect to particular contexts, thus, *contextual preference rules* [2] can be generated by pattern mining methods from a *preference database*.

Definition 3 (Preference Database)**.** *A* preference database $\mathcal{P} \subseteq \mathcal{D} \times \mathcal{D}$ *is a set of pairs of tuples representing a sample of user preferences over the dataset* $\mathcal{D}$*. Each tuple consists of a set of distinct literals (items) in all items* $\mathcal{I}$*.* ■

Intuitively, a *user preference* $\langle t, u\rangle \in \mathcal{P}$ means that the user prefers the tuple t to the tuple u. Given a user preference $\langle t, u\rangle \in \mathcal{P}$, t is said to be the *preferred* tuple and u is said to be the *non-preferred* tuple according to the user. Our main objective is to extract a user profile from a preference database provided by the user. Basically, a *contextual preference rule* $i^+ \succ i^- \mid X$ means that the item i^+ is preferred to the item i^- when the context X occurs.

Definition 4 (Contextual Preference Rule)**.** *A* contextual preference rule *is an expression of the form* $X \rightarrow I^+ \succ I^-$ *where* X *is an itemset of context,* I^+ *and* I^- *are itemsets of* $\mathcal{I} \setminus X$*.* ■

A given contextual preference rule π can *agree* with a user preference $\langle t, u\rangle \in \mathcal{P}$ when the preference rule π leads to prefer t rather than u, or *contradict* $\langle t, u\rangle \in \mathcal{P}$ when the preference rule π leads to prefer u rather than t. The *agreement* of a contextual preference rule π in $\mathcal{P}$ gathers the set of pairs that are in agreement with π: $agree(\pi, \mathcal{P}) = \{\langle t, u\rangle \in \mathcal{P} \mid t \succ_\pi u\}$. Similarly, the *contradiction* of a contextual preference rule π in $\mathcal{P}$ denotes the set of pairs that are in contradiction with π: $contradict(\pi, \mathcal{P}) = \{\langle t, u\rangle \in \mathcal{P} \mid u \succ_\pi t\}$. The *cover* of a contextual preference rule π in $\mathcal{P}$ denotes the set of pairs that are covered by π: $cov(\pi, \mathcal{P}) = agree(\pi, \mathcal{P}) \cup contradict(\pi, \mathcal{P})$. Hence, we have the following definitions of the *support* and the *confidence* of contextual preference rules.

Definition 5 (Support)**.** *The support of a contextual preference rule* π *in* $\mathcal{P}$ *is defined as:*

$$supp(\pi, \mathcal{P}) = \frac{|agree(\pi, \mathcal{P})|}{|\mathcal{P}|}.$$

■

Definition 6 (Confidence)**.** *The confidence of π with respect to $\mathcal{P}$ is defined as:*

$$conf(\pi, \mathcal{P}) = \frac{|agree(\pi, \mathcal{P})|}{|cov(\pi, \mathcal{P})|}.$$

■

Obviously, a contextual preference rule is similar to an association rule [1], where the only difference is that the left-side is an itemset (the context) and the right-side consists of two itemsets (the preference). Therefore, the extraction of contextual preference rule can resolved by frequent pattern mining with pre-processing and post-processing:

- *Frequent itemset pattern mining*: in this case, the user preference on two tuples can be combined to a transaction by adding prefix to each item to distinguish common items (the context X), preferred items (I^+), and non-preferred items (I^-), then, the preference rules can be generated [29, 28].
- *Frequent sequence pattern mining*: in this case, the user preference on two tuples can be represented by a sequence $\langle$(common items)(preferred items)(non-preferred items)$\rangle$, where a sequence pattern is directly a preference rule [44].

The main strategy of the algorithm `ProfMiner` responsible for building user profiles is the ability of selecting the best contextual rule to decide which transaction is the preferred one. The following definition introduces a total order on the set of contextual preference rules.

Definition 7 (Best Rule Order)**.** *Given $\mathcal{P}$, the* best rule order *on a set of contextual preference rules, denoted by $>_{best}$, is a total order defined for any contextual preferences π and π' as:*

$$\pi >_{best} \pi' \Leftrightarrow \begin{cases} conf(\pi, \mathcal{P}) > conf(\pi', \mathcal{P}) \text{ or} \\ conf(\pi, \mathcal{P}) = conf(\pi', \mathcal{P}) \text{ and } supp(\pi, \mathcal{P}) > supp(\pi', \mathcal{P}) \text{ or} \\ conf(\pi, \mathcal{P}) = conf(\pi', \mathcal{P}) \text{ and } supp(\pi, \mathcal{P}) = supp(\pi', \mathcal{P}) \\ \quad \text{and } |context(\pi, \mathcal{P})| < |context(\pi', \mathcal{P})| \text{ or} \\ conf(\pi, \mathcal{P}) = conf(\pi', \mathcal{P}) \text{ and } supp(\pi, \mathcal{P}) = supp(\pi', \mathcal{P}) \\ \quad \text{and } |context(\pi, \mathcal{P})| = |context(\pi', \mathcal{P})| \text{ and } \pi <_{\mathcal{CP}} \pi' \end{cases}$$

■

As the profile should contradict at most a small number of user preferences in order to have a high precision, the confidence is the most important criterion. The support criterion naturally comes in second place, followed by the size of the context. The fourth criterion, where $<_{\mathcal{CP}}$ is an arbitrary total order, is only used to definitely decide between two indistinguishable rules.

1.2.2 The Ranking Problem

In order to predict between two tuples (t, u) which is the preferred one, it is necessary to benefit from the preference rules that cover these transactions created from such tuples.

Let $\Pi_{\langle t,u\rangle}$ denote the set of preference rules that covers (included in the transaction consisting of t and u) the pair of tuples (t, u), i.e. $\Pi_{\langle t,u\rangle} = \{\pi \in \Pi \,|\, t \succ_\pi u \vee u \succ_\pi t\}$. When a preference rule $\pi \in \Pi_{\langle t,u\rangle}$ induces that t is preferred to u, the *confidence* of π represents the degree of preference of t with respect to u. The weight of a preference rule π for evaluating a pair $\langle t, u\rangle$ is then defined by: $w_\pi(t, u) = conf(\pi, \mathcal{P})$ if $t \succ_\pi u$ or $w_\pi(t, u) = 1 - conf(\pi, \mathcal{P})$ if $u \succ_\pi t$. In this context, the challenge of ranking transactions is to select the right rules in $\Pi_{\langle t,u\rangle}$ and to aggregate their weights.

Best Rule A very natural way for determining whether a transaction t is preferred to another u is to use the *best* contextual preference rule $\pi \in \Pi_{\langle t,u\rangle}$. For instance, a best contextual preference rule according to $>_{best}$ covering $\langle CDE, AC\rangle$ is $C \rightarrow A \succ D$ that induces AC is preferred to CDE if the confidence is 1. The best rule preference function [26] is defined as the following:

$$br_\Pi(t, u) = \begin{cases} w_\pi(t, u) & \text{if there exists a best rule } \pi \text{ in } \Pi_{\langle t,u\rangle} \\ 0.5 & \text{otherwise} \end{cases} \tag{1.1}$$

A value of br_Π which is close to 1 is interpreted as a strong recommendation that t should be ranked before u with respect to the user profile Π.

Weighted Voting It is counterproductive to consider only the best rule when the profile can contain 10 other rules whose conclusion is opposite. Based on this observation, we introduce a second alternative which consists in using a weighted voting strategy as proposed in [71]. When comparing two transactions, the idea is to used not only the best rule that covers this pair, but the set of all rules that covers it (i.e., $\Pi_{\langle t,u\rangle}$). In this case, each rule is weighted by the confidence in the final recommendation using w_π. The preference function wv_Π is defined as:

$$wv_\Pi(t, u) = \begin{cases} \frac{1}{|\Pi_{\langle t,u\rangle}|} \sum_{\pi \in \Pi_{\langle t,u\rangle}} w_\pi(t, u) & \text{if } \Pi_{\langle t,u\rangle} \neq \emptyset \\ 0.5 & \text{otherwise} \end{cases} \tag{1.2}$$

Using the preference function wv_Π, we can defined the preference relation $\succ_\Pi^{wv}$ by: $t \succ_\Pi^{wv} u$ iff $wv_\Pi(t, u) > 0.5$. Because of the simplicity of our toy example, the preference relation $\succ_\Pi^{wv}$ is exactly the same as $\succ_\Pi^{br}$.

Range Voting Regardless of our profile construction and ordering methods, repairing and completing a preference relation to respectively tackle its inconsistency and its sparseness is a challenging task [83]. Instead of directly comparing two transactions thanks to the relation $\succ_{\Pi}^{br}$ or $\succ_{\Pi}^{wv}$, we propose to organize a *range voting* for making decisions. A range voting is a voting system for one-seat elections under which voters in T rate each candidate with a number within 0 and 1 using a preference function like br_{Π}or wv_{Π}. The scores for each candidate are summed, and the candidate with the highest value is the best.

Definition 8 (Range Voting Order)**.** *Given a preference function $pref_{\Pi}$, the range voting order is defined as:*

$$t \succ_{\Pi}^{T,pref} u \Leftrightarrow \sum_{v \in T} pref_{\Pi}(t, v) > \sum_{v \in T} pref_{\Pi}(u, v)$$

Indeed, as range voting satisfies resolvability criterion [137] that ensures a low possibility of tie votes, $\succ_{\Pi}^{T,br}$ enables to rank more transactions than $\succ_{\Pi}^{br}$. Furthermore, even if the range voting remains consistent with most of user preferences.

1.2.3 Experimental Evaluation

This section presents the experimental study conducted in order to evaluate the performances of our proposal. All the experiments have been conducted on real world datasets based on APMD-Workbench [105] automatically built from MovieLens[2] and IMDB[3]. This dataset consists of 800, 156 ratings (ranging from 1 to 5) from 6, 040 users on 3, 881 movies. Each user has rated at least 20 movies, and each movie is described by a set of features such as Genre, Director, Year, Actor, etc.

In order to evaluate our approach, we extract from the initial dataset 20 datasets of rated movies evaluated by 20 different users[4]. These users are randomly selected among the 140 users that rated between 500 and 600 movies, shown in Table 1.2. To compare our approach with the baseline method `SVMRank` [59], we use the same file format, i.e. each line of the 20 datasets contains first a movie's rating, and then a list of feature/value pairs representing characteristics of the rated movie (Genre, Director, Year, Actor, ...). It is important to note that our algorithm is trained and tested over a preference database, and not a set of rated tuples. However, one can easily build a preference database from a set of rated tuples, considering the set of all comparable pairs of tuples.

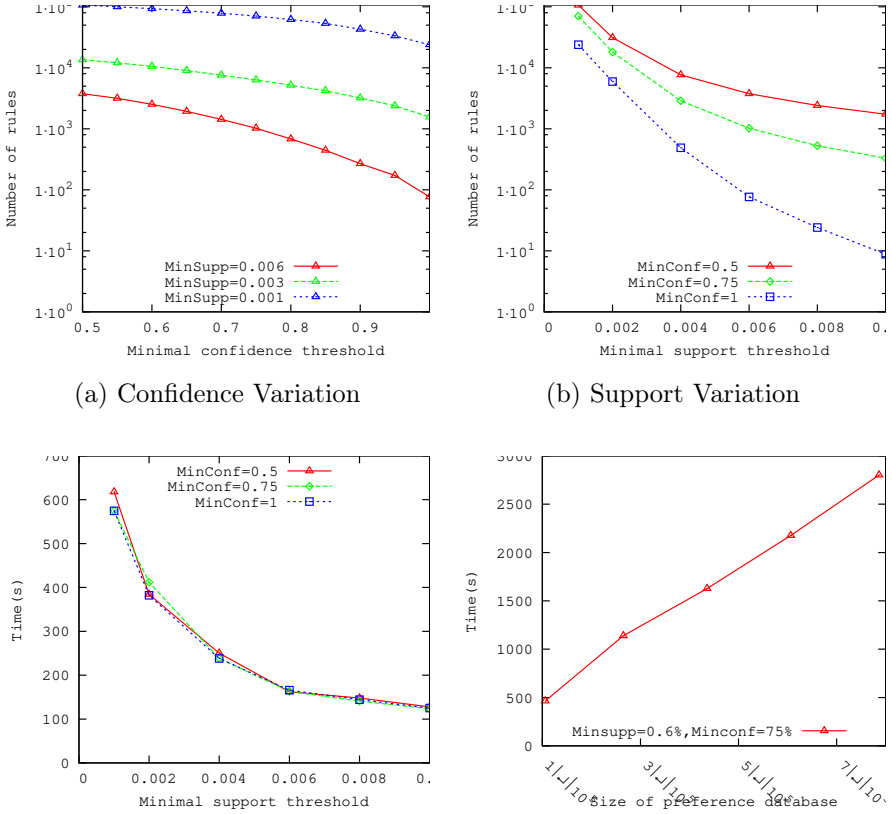
[2] `https://www.movielens.org`

[3] `https://www.imdb.com`

[4] `https://www.info.univ-tours.fr/~soulet/profminer`

We summarize our experimental evaluation as follows.

- *Preference rule extraction*: (1) as the minimum confidence and support increase, the number of extracted preference rules exponentially decreases; (2) the execution times are almost independent from the values of the minimum confidence, which means that the set of preference rules explored is not pruned by the confidence constraint; (3) the run time varies when the size of the preference databases increases while fixing minimum support and confidence. (Figure 1.1.)



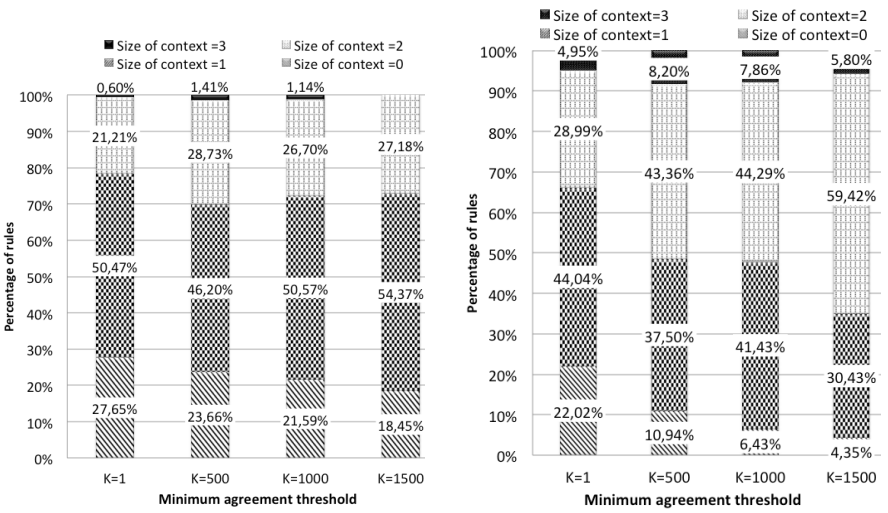
(a) Confidence Variation (b) Support Variation

(c) Support Variation (d) Size of preference database

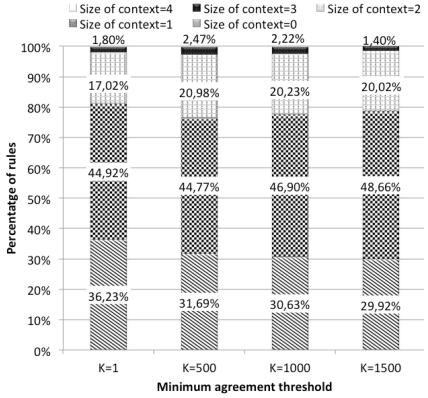
Figure 1.1: Performance of our method.

- *Conciseness and main characteristics of the profiles*: we analyzed the conciseness and readability of the user profiles built by `ProfMiner` according to a given minimal agreement threshold (denoted k), which shows that the user profiles can be as concise as desired by the user and that is very important to present understandable user profiles. (Figure 1.2 and Figure 1.3.)

- *Best Rule vs. Weighted Voting*: (1) the performance of the two ranking strategies are very



(a) User48 (b) User1125



(c) All Users

Figure 1.2: Percentage of rules according to the size of the context.

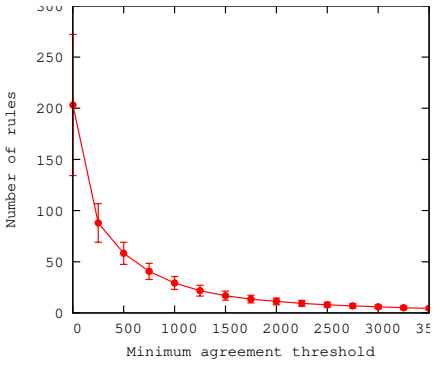


Figure 1.3: Number of preference rules w.r.t k.

similar; (2) by construction, each user preference is covered at the end by a very small

number of preference rules, which implies that weighted votes are mainly conducted within a very small number of preference rules; (3) the predictive quality of the mined profiles can be fairly high. (Figure 1.4.)

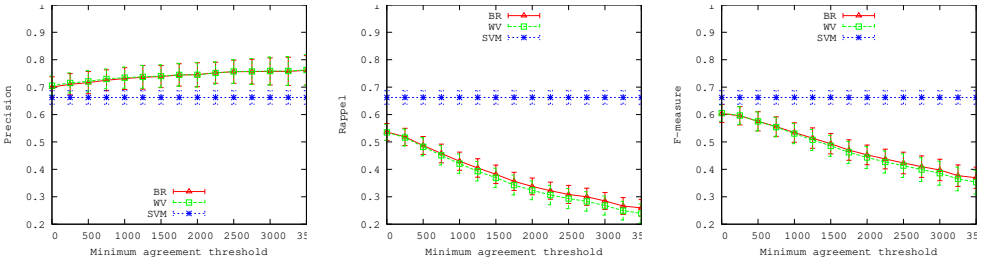


Figure 1.4: Efficiency of Best Rule strategy and Weighted Voting strategy

- *Best Rule vs. Range Voting*: (1) the precision of the both ranking strategies always remains fairly high, i.e. greater than the precision obtained using `SVMRank`; (2) with very small user profiles, the predictive quality (both precision and recall) of the user profiles is very high. (Figure 1.5.)

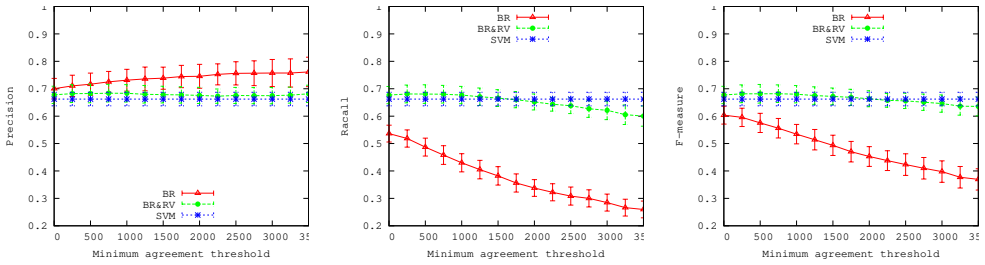


Figure 1.5: Efficiency of the ranking strategies.

1.2.4 Conclusion

We proposed the method `ProfMiner` for mining user profiles from preference databases. A set of experiments on a real-world database of user preferences about movies showed the efficiency of the method. More interestingly, our approach is the first one to build readable user profile based on the notion of contextual preference rules. In the last decade, preference mining techniques have been revealed very useful in database research concerning query personalizing. Preference models extracted from a given sample of user preferences can be stored in repositories which will be queried in order to personalize the answer to an SQL query proposed by the user. Preference mining techniques can also be very helpful in the development of recommendation systems for e-commerce applications.

1.3 Two-Phase Flow Pattern Visualized Identification

In this section, we present a sequence-based video learning application to identify and classify complex oscillatory two-phase flow patterns (note that *flow pattern* is a term in fluid mechanics, where the word *pattern* is different than the term *pattern* in frequent pattern mining) in the context of the cooling gallery of internal combustion engines. Note that this is the first video learning approach in this domain.

1.3.1 Engineering Background

In automobile industries, as one of the most important in-cylinder components of internal combustion engines, the piston operates under the most severe conditions with the highest thermal load, which requires proper thermal controlling methods to maintain its lifetime and prevent fatigue damages. The *two-phase flow* heat exchange process in the cooling gallery definitively affects the piston thermal state, however, due to the special working environment of pistons, including high temperature, high strength, and reciprocating motion, most of the early experimental studies could only measure the temperature of the piston as a calibration basis, but could not directly observe the movement of the two-phase flow in the piston. Hence, many studies have been conducted to investigate the oscillating flows via numerical simulation and experiments, as well as the studies presented in [50, 117].

The oscillatory two-phase flow patterns and heat transfer enhancements are closely related to various factors, including but not limited to the size, the shape, the heating state of the container, the surface tension of the medium, and the shear stress between phase interfaces. Therefore, it is difficult to construct a common mathematical model to describe the oscillatory two-phase flow that takes all of the above variables into account. Actually, Computational Fluid Dynamics (CFD) approach, which can study fluid flow and heat transfer of the gallery, has been extensive employed in providing effective assistance for the evaluation of pistons' thermal state and structural design, such as the research work presented by [31, 32]. Due to the complicated features of the structure of piston, the previous study presented by [144] shows that most of the existing methods can only be observed from one direction, that is, along the line of sight, the depth of field is 10–20cm; moreover, the liquid also fluctuates during the movement, which makes it difficult to achieve a high definition. Thus, it is necessary to make the visualization equipment reach a large viewing range, high resolution, and high frame rate at the same time. However, the performance of existing equipment basically cannot meet all these requirements.

Our experiment used the structure, size and stroke of the real piston so that the *Reynolds*

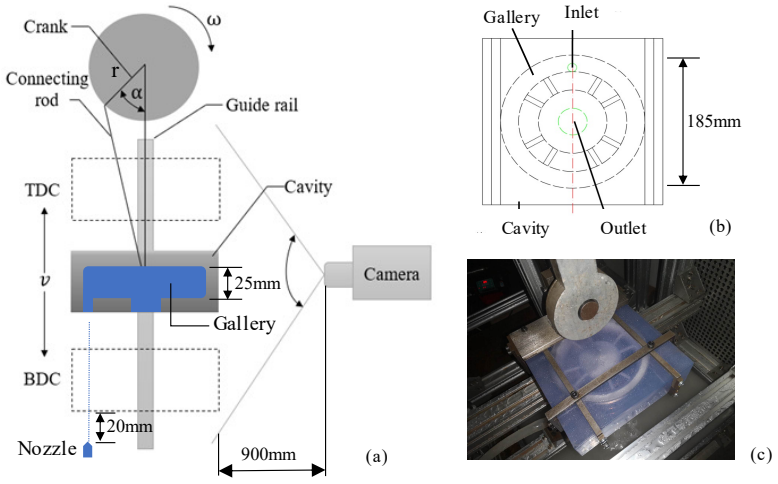


Figure 1.6: (a) The movement of the cavity. (b) The design of the cavity. (c) A photo of the standby cavity.

numbers[5] concerned in this section are in the same range as real working conditions. Because the experimental conditions of our study are close to the real working conditions, the clarity of visualization results is actually affected. Therefore, the machine learning methods are introduced to analyze and classify the captured videos, seeking for the relationship between Reynolds numbers and internal two-phase flow status, and to provide more stable and reliable verification data for analysis with the CFD results.

We simulate the movement of a piston within a simplified and transparent model of cooling gallery under motor-driven forced high-speed oscillations, so that the two-phase flow patterns can be visualized by a high-speed camera. The cavity made by 3D printing technology contains a $1:1$ model of the cooling gallery in the diesel engine piston studied in our research, where the cooling medium is continuously sprayed. The high-speed camera employed in this measurement is Photron Fastcam Mini AX100, which can provide a resolution of 1024×1024 pixels at a maximum speed of 4000 frames per second (4000fps). The playback speed of recorded videos is 30fps (about 133:1), hence, we are able to trace the flow patterns. Figure 1.6 shows a schematic of the crank rod mechanism. The cavity is driven by the motor to make up and down reciprocating oscillations due to the connecting with the crank and connecting rod. In Figure 1.6 (a), α and r represent the crankshaft angle and radius respectively. The designed piston stroke (the distance

[5]We use the Reynolds number (**Re**) to generalize and to measure the two-phase flow patterns within a cooling gallery.

between the TDC and BDC positions of the cavity) is 210mm. With different motor speed, the oscillation speed of the cavity will change so the liquid oscillates back and forth with the cavity, which will lead to different results of the flow state, and will produce different oscillatory two-phase flow patterns.

In order to capture the both status in TDC and BDC, the range in photography need to cover the whole stroke, which limits the shooting speed (4000fps). In further faster motions, the clarity and focus will be affected directly. In our experiments, the oscillation motion starts to be observed in the cavity while the motor speed reaches 200rpm, so we select the minimum speed as 200rpm; due to the security concerns, we limit the maximum speed as 600rpm as real environments on marine low-speed diesel engines. Totally 9 video clips of 30fps are recorded in our experiments, corresponding to the motor speeds at 200rpm, 250rpm, 300rpm, till to 600rpm, where each clip lasts about 10 minutes. The proposed method has also been successfully applied to verify CFD simulation results on the oscillating cooling gallery.

Note that the Reynolds number is a dimensionless constant that can distinguish the patterns and boundary of oscillatory two-phase flow. Based on the studies by [15], the corresponding expression is

$$R_e = \frac{\overline{v} D_e \rho}{\mu}, \tag{1.3}$$

where R_e is a dimensionless Reynolds number of the cavity under forced oscillation, $\overline{v}$ is the mean oscillating speed of cavity with a diameter D_e, and ρ and μ represent the density and viscosity of the glycerin into the cavity, respectively. The instantaneous oscillation speed v of cavity is defined as

$$v = r\omega(\sin\alpha + \frac{\lambda}{2}\sin 2\alpha), \tag{1.4}$$

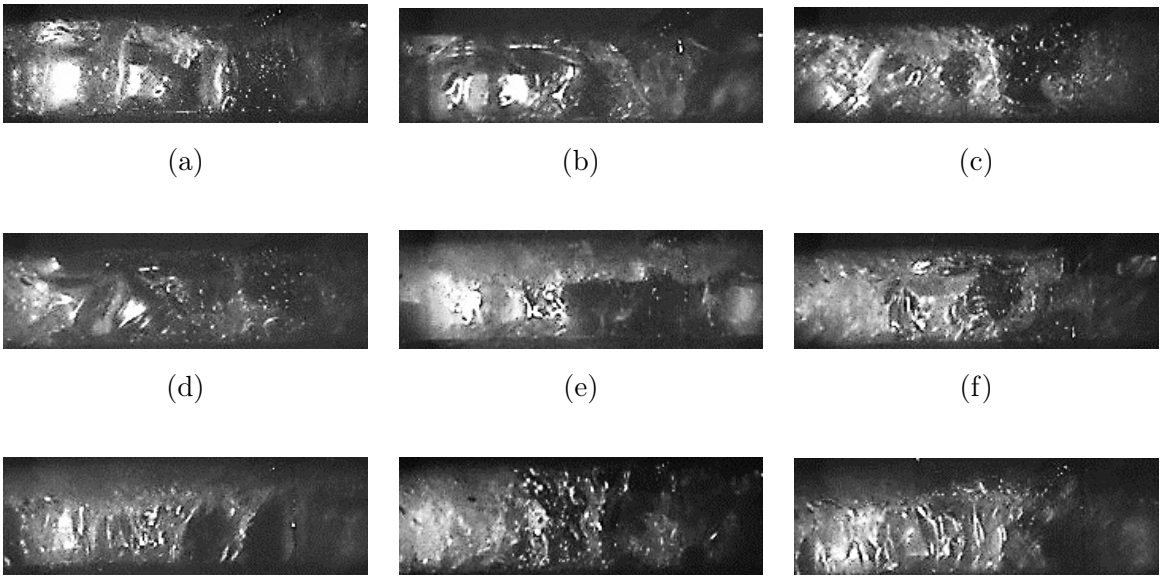
where ω is the motor speed, λ is the connecting rod ratio, and α and r denote the angle and radius of the crank.

Figure 1.7 shows examples of nine different oscillatory two-phase flow patterns with respect to different Reynolds number during the experiment analyzed in this section from the videos at various Reynolds number when the cavity reached the Top Dead Center (TDC). Obviously, the characteristics of a flow field at various Reynolds number is difficult to identify and classify directly because the images shown in Figure 1.7 are similar and there are no significant visible differences in the two-phase flow patterns contained in them. However, it is a manual observation result in the static context (single images) instead of the dynamic procedure of the two-phase flow, that is why we use video data captured by high-speed camera to study two-phase flow patterns.

The results of our studies show that in the dynamic context (videos), the machine learning

$\omega(rpm)$	$\overline{v}(m \cdot s^{-1})$	R_e
200	2.31	10568
250	2.89	13221
300	3.46	15830
350	4.04	18484
400	4.62	21137
450	5.20	23791
500	5.77	26398
550	6.35	29052
600	6.93	31704

Table 1.1: Velocity curves of pistons with respect to motor speed.



(a) (b) (c)

(d) (e) (f)

(g) (h) (i)

Figure 1.7: Analyzed flow patterns under the Reynolds number: (a) 10568, (b) 13221, (c) 15830, (d) 18484, (e) 21137, (f) 23791, (g) 26398, (h) 29052, and (i) 31704. Each image represents the 185mm $\times$ 25mm cooling gallery inside the cavity.

methods can identify complex two-phase flow patterns, and the mean classification accuracy obtained in our experiments show that the presented method allows to find the differences between different flow patterns corresponding to different Reynolds numbers.

1.3.2 Visualized Identification of Two-Phase Flow Patterns

Based on image processing and machine learning techniques that makes the visualization results of oscillatory two-phase flow with complex flow patterns convinced, our method can be summarized as the following 3 steps:

1. Extract two-phase flow patterns from traced optical flow points of video segments.
2. Represent two-phase flow patterns as a set of sequences of discretized offsets on the 2D X/Y axis.
3. Characterize two-phase flow patterns by feature vectors consisting of n-grams generated from offset sequences.

We consider that two-phase flow pattern descriptors can be able to characterized by the features used by video classification approaches since such features allow to classify video data and their performance can be effectively evaluated by the accuracy of classification results. Note that although deep neural networks, such as, typically, convolutional neural networks (CNN) can accurately accomplish video classification tasks, the iterations inside the multilayer network make it impossible to exploit learned features in other applications, such as CFD in the context of our research.

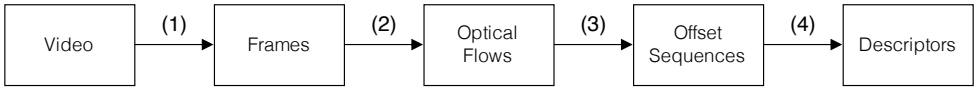
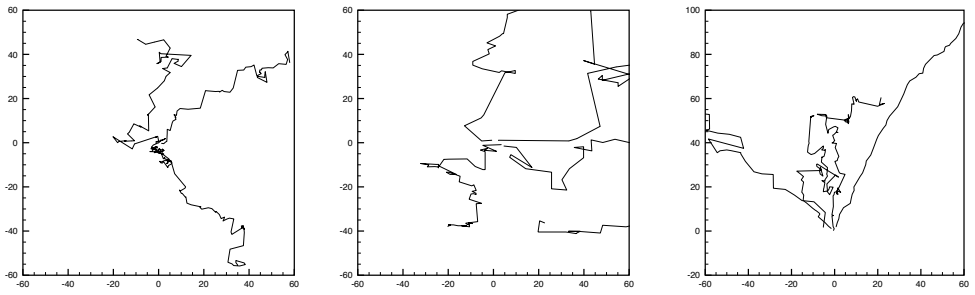


Figure 1.8: Extraction of flow pattern descriptors.



(a) (b) (c)

Figure 1.9: Sample optical flow points at motor speed (a) $R_e = 10568$, (b) $R_e = 21137$, and (c) $R_e = 31704$. The units of axes are pixel-distance in captured videos.

The data processing chain of our method is shown in Figure 1.8. First, at the step (1), a captured video clip is converted to a sequence of frames; then, at the following step (2), according to the method developed by [113], a set of moving points are extracted by optical flow detection that represent the movements of phase interfaces; at the step (3), we represent the traced moving points by consecutive offsets on the 2D X/Y axis, hence, such offset sequences depict two-phase flow patterns; finally, we extract classification assessed features as two-phase flow pattern descriptors at the step (4). Obviously, an offset sequence is a time series-like sequence of bidimensional numeric values, of which, however, the form is very different than that of time series, as shown in Figure 1.9, where 4 randomly selected sample offset sequences are plotted with respect to Reynolds numbers 10568, 21137, and 31704. Therefore, our problem is different than the time series classification problem although there exist many efficient methods with respect to a recent review of [8]. In fact, the two-phase flow pattern characterization problem studied in this section is rather close to the feature selection problem in general sequence classification surveyed by [138], where the numeric value discretization-based sequential patterns are used as features, such as in the work of [39] and of [35]. However, the data model of traditional sequential patterns does not fit our requirements because of the lack of temporal continuance while defining patterns: undetermined gaps between the elements in a sequence are allowed, which make the description of two-phase flow imprecise. Hence, we finally consider frequent discretized n-grams, which can been regarded as sequential patterns with consecutive items, as two-phase flow pattern descriptors.

Figure 1.10: Evaluation of flow pattern descriptors.

The quality of extracted two-phase flow pattern descriptors is measured by N-fold cross validation of video classification as shown in Figure 1.10. Traditionally, each video clip is split to N independent segments without overlap, of which $N-1$ segments are used as training set and 1 segment is used as testing set for each fold. Flow pattern descriptors for training and testing the classifier are discretized only based on training data (that will be detailed in the next section) at the step (1), then at step (2) we construct per-segment feature vectors so that any classification methods can be used to evaluate flow pattern descriptors as at steps (3) and (4).

To detect two-phase flow patterns, we use *corner points* contained in consecutive sampled frames of each video clip and calculate the optical flow for each point by using the method of [113], which is a technique widely used in computer vision. The feature points used in our approach

are therefore corner points detected and tracked in frames that are the best for estimating the motions in the video.

The corner points are tracking in order to calculate *optical flow* for these points. Optical flow is the pattern of apparent motion of image objects between two consecutive frames caused by the movement of object or camera, which intuitively reflects the two-phase flow pattern in our study. An optical flow consists of X/Y offset vectors showing the movement of points from the first sampled frame to the second (intuitively reflecting the two-phase flow patterns) to which the Lucas-Kanade method [81, 13] is the most widely used one. The Lucas-Kanade method can provide an estimate of the movement of interesting features in successive images of a scene, with some implicit assumptions: (1) two frames are separated by a small time increment Δt (see Section 3.2 for the estimation), in such a way that objects have not displaced significantly; (2) textured objects exhibiting shades of gray which change smoothly. Assume that the increase of brightness per pixel at position (x, y) is respectively $I_x(x, y)$ and $I_y(x, y)$ in X/Y directions, the total increase of brightness after a movement by u and v pixels respectively in the X/Y directions can be calculated by

$$I_x(x, y) \cdot u + I_y(x, y) \cdot v$$

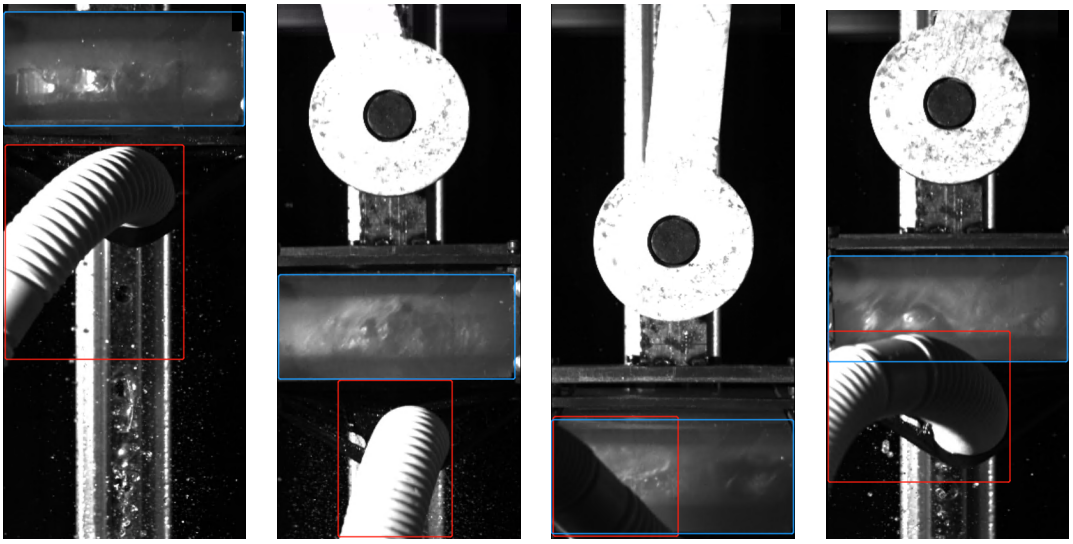
that matches the local difference in intensity which we call $I_t(x, y)$, so that

$$I_x(x, y) \cdot u + I_y(x, y) \cdot v = -I_t(x, y).$$

In global, the goal is to find all those pixels in each frame that represent a corner, that is, a corner lies on a neighborhood in which edges in several directions appear in the image. Notice that the corners detected at the this step are suppressed if they are not a local maximum neighborhood in terms of intensity value, and those with the minimal eigenvalue λ_m less than a threshold are also rejected. Given a corner pixel $p = (x, y)$, its minimal eigenvalue λ_m is computed on the covariance matrix of derivatives centered on p, that is,

$$\lambda_m = \begin{bmatrix} \sum_{S(p)} (dF/dx)^2 & \sum_{S(p)} dF/dx \cdot dF/dy \\ \sum_{S(p)} dF/dx \cdot dF/dy & \sum_{S(p)} (dF/dy)^2 \end{bmatrix},$$

where $S(p)$ is the neighborhood of the pixel p and function dF/dx is the derivative of the frame F at the point with respect to x and to y. Let λ_{max} the maximum value of λ_m over the whole frame F, it is suggested to retain 10% or 5% of the image pixels that have a λ_m value larger than a percentage of λ_{max} [13]. The remaining corners are sorted by the quality measure in the descending order, and finally, the algorithm removes each corner for which there is a stronger corner at a distance less than a threshold. In our approach, the percentage is set to 10%.



(a) (b) (c) (d)

Figure 1.11: Motion segments: (a) top, (b) downward, (c) bottom, and (d) upward.

By default, Lucas-Kanade method takes a 3×3 patch around an identified point (x, y), so all the 9 points shall have the same motion. Therefore, we can find (f_x, f_y, f_t) for these 9 points (the derivatives along vertical, horizontal axis and along the intensity dimension) so the problem becomes solving 9 equations with two unknown variables which is over-determined, to which a solution is obtained with least square fit method, that is,

$$\begin{bmatrix} u \\ w \end{bmatrix} = \begin{bmatrix} \sum_i {f_{x_i}}^2 & \sum_i f_{x_i} f_{y_i} \\ \sum_i f_{x_i} f_{y_i} & \sum_i {f_{y_i}}^2 \end{bmatrix}^{-1} \begin{bmatrix} -\sum_i f_{x_i} f_{t_i} \\ -\sum_i f_{y_i} f_{t_i} \end{bmatrix},$$

where (x_i, y_i) are points belonging to the neighborhood of (x, y). The solution u and w of the above equation is therefore the optical flow vector of the point (x, y), we also call it a *offset vector*. For each traced optical flow point, the time-ordered list of all offset vectors is an *offset sequence*, so we can represent a video as a set of offset sequences.

With respect to the total number of frames F and the physical motion duration T of the video, the interval of each frame is clearly $\Delta f = T/F$. According to our experiment setup, the ratio of frame number and motion duration is about 30 (frames per second), so the value of frame interval is taken as $N = (F/T)(1/i) = 30(1/i)$, where $i = 2$ ($i = 6$ for *segmented motion*, is finally selected with considering the mean length of extracted offset sequences, and therefore, the Δt mentioned in Lucas-Kanade is finally reduced to be every 15 frames (every 5 frames for segmented motion).

During our visualized experiments, we noticed the existence of incomplete flow patterns

caused by a plastic pipe that occasionally adheres to the cavity at the bottom position and at the upward stage, shown as Figure 1.11, where the red and blue boxes represent the tube and the cavity respectively. The plastic pipe shown in the video is scalable, one side is sticked at the outlet of piston model, and another side is fixed on a collection can to collect the oil through the cavity. Indeed, our experiments confirms the influence of such block of visibility in flow pattern classification, hence, it is important to reduce such influences. In our approach, we segment video clips into four physical motions including *top*, *downward*, *bottom*, and *upward* with respect to the position of the cavity, and each video clips contains several cycles of motions, in order to further prove the usefulness and the efficiency of our proposed two-phase flow pattern descriptors: the classification results obtained from different motion segments must be different, which is confirmed by our experimental results.

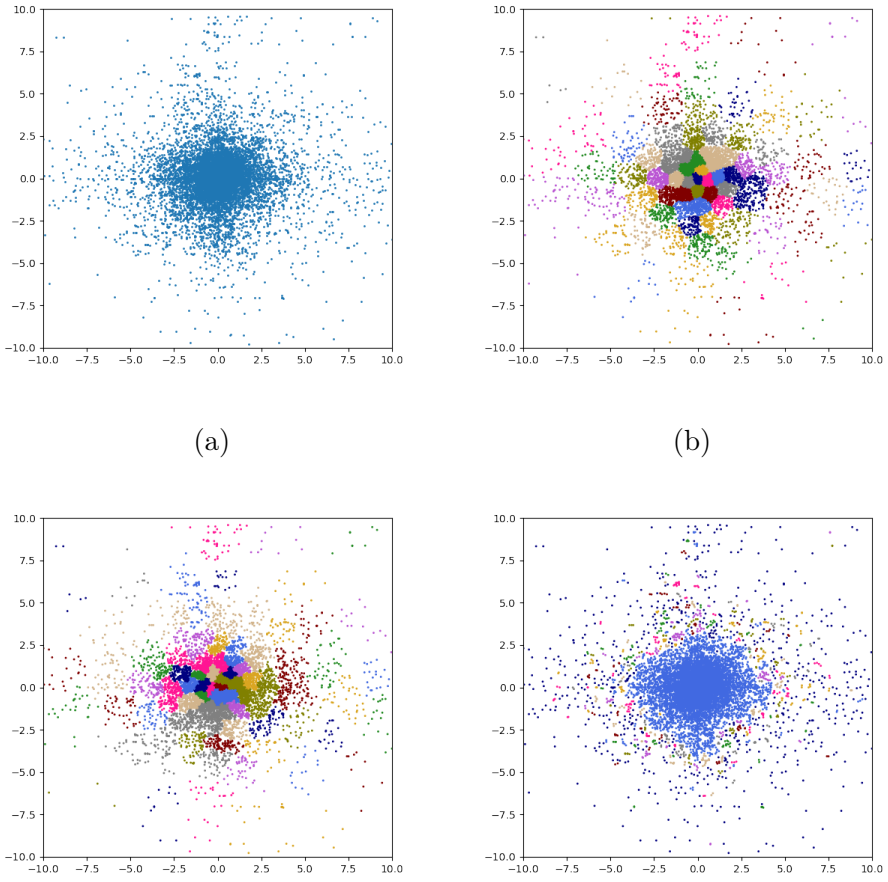
Let λ_1^t and λ_2^t be the time points of the physical top position of the cavity between two consecutive cycles, the motion duration per cycle $\sigma = \lambda_2^t - \lambda_1^t$ shall be a constant since the motor speed is fixed. The total number k of cycles contained in a video clip is therefore $k = F/(30\sigma)$. Let the physical top and bottom positions of the cavity between two contiguous cycles be denoted by λ_m^t, λ_{m+1}^t, λ_m^b, and λ_{m+1}^b, where $m \in \mathbb{Z}$ and $1 \leq m < k$, and in considering that the cavity stays at the top and bottom for a short time δ, the physical motion of a video can be therefore divided to 4 segments:

$$
\begin{aligned}
T_{top} &= [\lambda_m^t - \delta, \lambda_m^t + \delta], \\
T_{downward} &= [\lambda_m^t + \delta, \lambda_m^b - \delta], \\
T_{bottom} &= [\lambda_m^b - \delta, \lambda_m^b + \delta], \\
T_{upward} &= [\lambda_m^b + \delta, \lambda_{m+1}^b - \delta].
\end{aligned}
$$

For instance, before estimating time increment, the video clip of 200rpm contains 16939 frames, let $\delta = 1s$, we get respectively 1901, 5391, 1291, and 7819 frames for the above 4 segments without counting incomplete cycles. Our experiments show that the *top* and *downward* segments can individually characterize the flow patterns in our cooling gallery model and outperform full length video clips.

1.3.3 Classification of Two-Phase Flow Patterns

Assume that a video clip P contains n traced points of which the movement reflects the gas-liquid border, each point p_i $(1 \leq i \leq n)$ characterizes P, denoted as $p_i \models P$. Then, the movement of a point p is identified as an ordered list of offsets (x, y) of contiguous values, that is, an *offset sequence*. An offset sequence depicts a two-phase flow pattern. Since a video clip can contain



(a) (b)

(c) (d)

Figure 1.12: (a) All offset vectors. (b) The clusters generated by the k-`means` method ($k = 100$). (c) The clusters generated by the Agglomerative method ($c = 100$). (d) Clusters generated by the `DBSCAN` method ($\epsilon = 0.157$ and $MIN = 2$). The units of axes are pixel-distance in captured videos.

multiple two-phase flow patterns, in this context, a video clip can be considered as a set of offset sequences. Therefore, to characterize the two-phase flow patterns appearing in a video clip is indeed to feature a set of offset sequences.

The contiguous values contained in offset sequences prevents the collection of common values, which is a necessity of featuring offset sequences. A general way to discretize numerical vector attributes (i.e. the offset vectors) is to use a clustering method to generate symbolic attributes with respect to predefined constraints (for instance, as k-`means`, the number of attributes) so

that a limited number of feature attributes can cover all contiguous values. We shown in Figure 1.12 a comparison between different clustering methods with respect to the clusters generated from all offset vectors gathered from a training video at the motor speed 200rpm, of which the Reynolds number $R_e = 10568$. Figure 1.12 (a) shows the spatial distribution of all offset vectors; Figure 1.12 (b), (c), and (d) respectively show the clusters generated by the k-`means` method with $k = 100$, by the agglomerative method that stops at 100 clusters, and by the `DBSCAN` method developed by [40] with $\epsilon = 0.157$ and $MIN = 2$. In fact, the tuning of parameters does not make the application of `DBSCAN` in our problem, by which no meaningful clusters can be generated with different parameters.

There is no definitive difference between the results obtained by k-`means` and Agglomerative methods, hence, we have finally adopted k-`means` to convert offset vectors to k discretized offset attributes because of its simplicity and wide usage. By k-`means` based conversion, making an offset sequence discretized to symbolic sequence generate common features can be extracted. As described in previously, the contiguous movements of two-phase flows are much interesting than a frequent set or a frequent subsequence of all discretized offset vectors, which are very probably discontinued movements. We therefore propose a n-gram based encoding of *feature vectors* that ensures the effectiveness of our method by keeping the efficiency (which can be considered as mining *frequent consecutive subsequences*): given a set S of discretized offset sequences that represents a set P of two-phase flow patterns (i.e. a video), each feature is the frequency of a n-gram g present in a sequence $s \in S$, where g is a n-gram that exists in any (at least one) sequence in the set S.

According to the above statement, we are two parameters to setup in order to find the best feature vectors to two-phase flow patterns: the k value for k-`means` and the n value for n-gram. The above task can be resolved by video classification processes. In our approach, each video clip corresponds to a Reynolds number, that is, a particular class of two-phase flow patterns. Since a set of two-phase flow patterns $P \to C$ can be represented as a set S_k^n of discretized offset sequences (where k and n are the parameters of k-`means` and of n-gram), denoted as $S_k^n \models P \to C$, and since $P \to C$ is a subjective mapping, we have $S_k^n \models C$, that is, the ensemble of all moving points identifies the class C of two-phase flow patterns P. Let $\mathcal{P}$ be a set of two-phase flow patterns represented as video clips and $\mathcal{C}$ be a set of classes, each video clip $P_i \in \mathcal{P}$ is assigned to a class $C_i \in \mathcal{C}$, our goal is to learn a factor $f(k, n) : S \to C$ where $S_k^n \models P$, where $S_k^n \models P$. The problem mentioned in this section can be therefore described as: learn a best factor $f(k, n) : \mathcal{S}_k^n \to \mathcal{C}$, where $\mathcal{S}_k^n = \{S_k^n \models P \mid P \in \mathcal{P}\}$.

We propose the following strategy to classify offset sequence set represented video data. Let

$\mathcal{C}$ be the set of all classes, S_k^n be a sequence set with respect to the two parameters k and n, h be a classifier such that for each sequence $s \in S_k^n$, $h(s) = C$ where $C \in \mathcal{C}$ is a class. Given a classifier h, we define the *sequence-level score* of a class C with respect to a sequence set S_k^n, denoted by $\sigma_h(C, S_k^n)$, as the number of all sequences $s \in S_k^n$ classified to C, that is,

$$\sigma_h(C, S_k^n) = \frac{|\{s \in S_k^n \mid h(s) = C\}|}{|S_k^n|} \times 100.$$

A class C is called the *dominant class* of a sequence set S_k^n with respect to a classifier h if for all other classes $C' \in \mathcal{C}$ we have $\sigma_h(C, S_k^n) > \sigma_h(C', S_k^n)$; otherwise we say that there is no dominant class so the sequence set S_k^n cannot be classified by the classifier h. Within the typical context of classification, a score is a sequence-level accuracy of a class, which can be calculated from the confusion matrix, and the computational task is to find the best pairwise (k, n) values that maximizes the number of correctly classified videos.

The principal idea of the classification is to validate the features that we extracted from video data that characterize two-phase flow patterns. Although there exist many classifiers, we mainly consider the Bayesian Network (`BN`) classifier. The Bayesian Network is a probabilistic graphical model that encodes the probabilistic relationships among the variables of interest. It is a natural way to express causal information and to discover hidden patterns from data. The Bayesian Network avoids using joint probability to reason directly, but uses independent relationship between variables to decompose joint distribution. Bayesian Network can analyze the dependence between features qualitatively and quantitatively, and then establish the network structure for probability reasoning. A Bayesian Network consists of two parts: network structure and parameter structure. The network structure is a directed acyclic graph, in which the nodes represent random variables, and the edges between nodes represent the dependence between variables; the parameter structure refers to the local probability dependence, and each node has a probability distribution. In brief, a Bayesian Network is learned to determine a network model that could best represent the dependent relationships of the variables in the data. In our approach, each variable treated by the Bayesian Network is a n-gram, so the dependency among n-grams can be handled.

On the other hand, the large number of features compose a high-dimensional search space, in which the performance of other classifiers such as the Support Vector Machine `SVM` classifier or the Decision Trees (`DT`) classifier might be very limited. Our experimental results have confirmed this selection of classifier. Although recent work [92] shows that deep learning on video data can also classify two-phase flow patterns, however the iterations inside the multilayer network make it impossible to exploit learned features in other applications, such as CFD in the context of our research.

1.3.4 Experimental Evaluation

We report our experimental results on the visualization experiments, on the Reynolds number classification with the mean accuracy of 94% and on the comparison with CFD results.

To make this section less complex, we only report the results of BN and SVM. All classification tasks have been done by the `BayesNet` (the BN classifier) and `SMO` (the SVM classifier with Sequential Minimal Optimization, using polynomial kernel) classifiers under Weka 3.9.3 with default parameters. The results of the DT classifier is very similar to the results of the SVM classifier. The results shown in our experimental evaluation are average values obtained by 5-fold of cross validation where the discretization of offset sequences is strictly independent between training sets and test sets.

Each video clip is first convert to a sequence of frames so that the optical flow detection method can be applied to extract offset sequences, and a Round Robin mechanism is used to distribute all offset sequences extracted from all video clips to 5 independent datasets without overlaps. Then, for each fold, we apply the k-means method only to each training set to generate symbolic offset attributes and then all offset vectors in each pair of training and test sets are converted to the nearest symbolic offsets, so all test sets are absolutely excluded from the training process. Table 1.2 lists basic statistical information about all sequence sets, where we denote $|S|$ the number of sequences, $||S||$ the number of distinct discretized offsets, $|s|$ the mean number of discretized offsets per sequence. We use $\{f, t, d, b, u\}$ to identify different segmented motions, f represents the full-cycle segments; t, d, b, and u represent respectively top, downward, bottom, and upward segments.

A fixed k value is necessary to evaluate the performance of our approach to two-phase flow identification by classification method, since this k value must be applied to both training data and test data. Figure 1.13 shows the performances of the BN classifier with unigram represented segmented motions to determine the best k values of the k-means method, where the results of the SVM classifier is shown as a reference. Each unigram is the centroid of one of the k clusters of offset vectors extracted from the offset sequences that represent concerned two-phase flow patterns. All these data show a comparison between mean sequence-level score of Reynolds number classification with different k values, where $k = 80$ and $k = 160$ are two interesting ones. In the curves, we see that there are much more dots in the range $0 < k < 200$ than the range $k > 200$, indeed the dichotomy is used to avoid test every possible k value. The curves also show that the *range* of k values does make the impact, but the sequence-level scores do not depend on individual k values. The results from the bottom (b) and upward segments (c) confirm the influence of the pipe, and so that the results from full cycle segments are affected (Figure 1.14).

R_e	$\|S\|_f$	$\|\|S\|\|_f$	$\|s\|_f$	$\|S\|_t$	$\|\|S\|\|_t$	$\|s\|_t$	$\|S\|_d$	$\|\|S\|\|_d$	$\|s\|_d$	$\|S\|_b$	$\|\|S\|\|_b$	$\|s\|_b$	$\|S\|_u$	$\|\|S\|\|_u$	$\|s\|_u$
10568	1028	64154	62	382	22920	60	1158	69480	60	78	4692	60	896	48436	54
13221	355	20864	58	133	6558	49	492	25950	52	148	3724	25	589	30476	51
15830	556	40084	72	311	18191	58	560	30800	55	182	11843	65	630	41774	66
18484	494	32437	65	198	12276	62	744	46128	62	187	11619	62	654	40548	62
21137	469	35685	76	134	7877	58	340	16866	49	87	6333	72	612	39154	63
23791	285	19907	69	127	8340	65	366	21152	57	78	4172	53	328	21561	65
26398	426	36783	86	264	25872	98	537	52626	98	233	22373	96	506	49278	97
29052	386	31563	81	177	10975	62	303	14055	46	107	8377	78	398	33693	84
31704	377	31337	83	178	11228	63	376	22694	60	128	9365	73	424	35612	83
TRAIN	2857	221933	77	1232	84014	68	3051	196336	64	796	54961	69	3498	255081	72
TEST	720	55954	77	312	21347	68	769	49526	64	206	14312	69	878	64154	73

Table 1.2: Statistical information of full cycle segments (f), top segments (t), downward segments (d), bottom segments (b), and upward segments (u) with respect to the Reynolds numbers. The mean values of Test sets and Train sets are respectively listed.

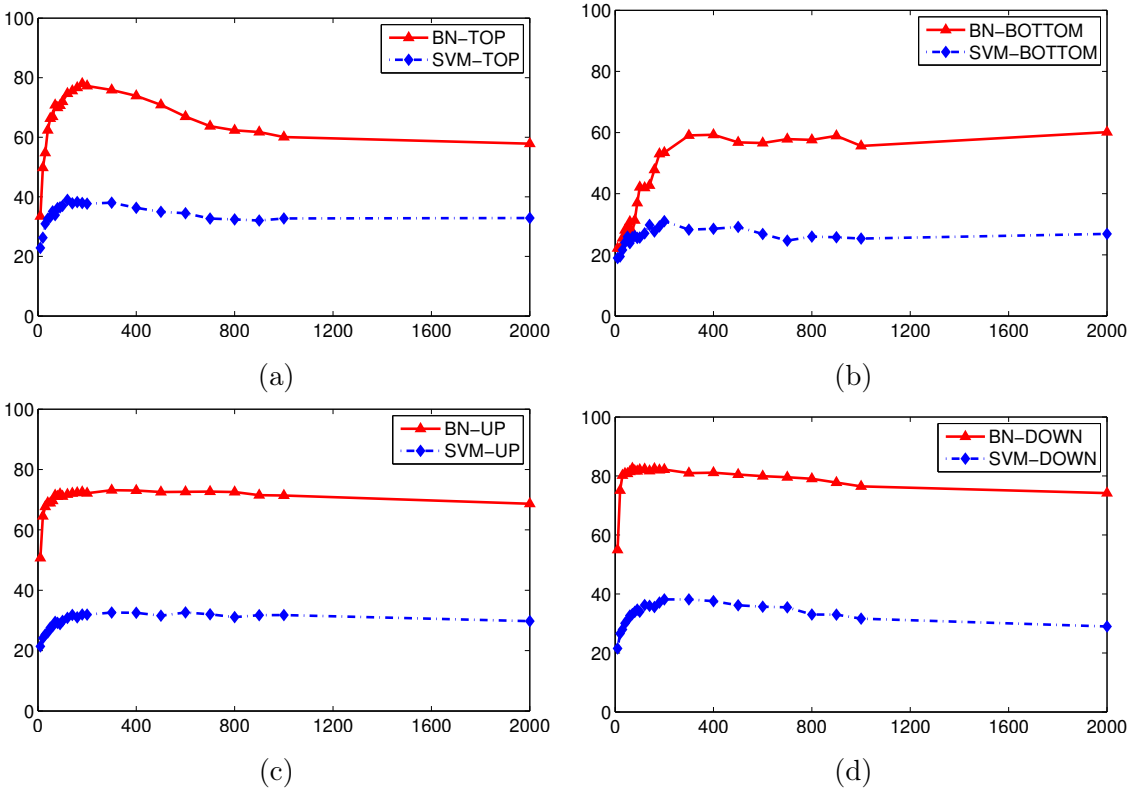


Figure 1.13: Reynolds number classification mean sequence-level score (Y axe) and k values (number of features, X axe). (a) Top segments. (b) Bottom segments. (c) Upward segments. (d) Downward segments.

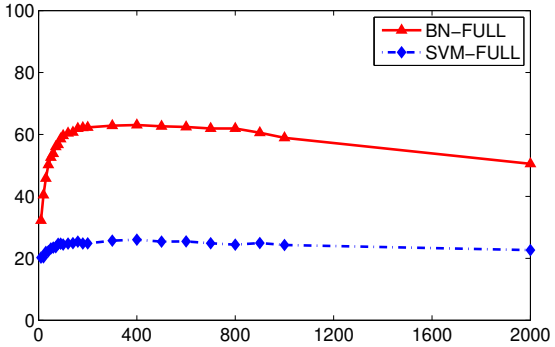


Figure 1.14: Reynolds number classification mean sequence-level score (Y axe) and k values (number of features, X axe) in full-cycle segments.

Segment	Features	Accuracy	10568	13221	15830	18484	21137	23791	26398	29052	31704	Mean
Full	400	8/9	97.70	94.37	51.72	73.56	37.93	**63.16**	64.71	**62.34**	21.33	63.07
Top	180	**9/9**	**100.00**	80.77	92.31	**100.00**	61.54	44.00	97.44	51.43	40.00	77.97
Downward	70	**9/9**	96.91	70.10	**98.97**	**100.00**	73.53	36.99	**100.00**	48.33	**90.67**	**82.54**
Bottom	2000	7/9	66.67	**100.00**	50.00	**100.00**	**94.12**	20.00	73.08	23.81	32.00	60.12
Upward	400	8/9	73.15	91.67	71.30	**100.00**	55.56	15.38	97.03	51.90	64.29	73.03

Table 1.3: Best k values (**Features**) of video-level classification accuracy of Reynolds numbers from **10568** to **31704** with unigram model and the BN classifier. The values in bold are the best ones in the column.

Table 1.3 shows the best k values (number of features) for video-level Reynolds number (from 10568 to 31704) classification accuracy with respect to each segment. In top and downward segments, all the 9 Reynolds numbers are correctly identified and the best score is 100% however the worst one is 15.38% (upward segment of $Re = 23791$). Although the plastic pipe does not appear in all clips of bottom and upward segments, the exceptionally high number of features corroborated the instability of flow identification caused by this pipe.

Based on the above results, we tested bigram and frequent multi-gram based models by two fixed k values, 80 and 160, where $k = 80$ is the best trade-off value on balancing features/accuracy in all segments and $k = 160$ is a reference value. Not surprising, the results obtained by $k = 160$ are much worse than the results obtained by $k = 80$, so it makes no sense to further detail the case while $k = 160$. Nevertheless, we report the results of the SVM classifier in order to enrich the performance of probabilistic models (i.e, Bayesian Network in our research) in two-phase flow identification. The comparative results are shown in Table 1.4 and Table 1.5.

The number of n-grams generated from N variables is N^n. Let $k = 80$, then 80^1 unigrams, $80^2 = 6400$ bigrams, $80^3 = 512000$ trigrams, etc. can be generated. It is clear that the number of multi-grams cannot be handled while $n \geq 3$, so we apply a *frequency* filter f to reduce the number of features, that is, only the multi-grams whose frequency in the data is superior or equal f, called *f-frequent multi-grams*, are kept as features. The f-frequent n-gram based models are generated by frequency $f \in \{2, 3, 4, 5\}$ and $n \in \{2, 3, 4, 5, 6\}$ where n stands for n-gram. In Table 1.4 and Table 1.5, f and n are presented as explosion values f/n, for instance, $12259^{2/6}$ denotes that 12259 features obtained by $f = 2$ and $n = 6$. We did not apply the frequency filter to bigram, the number of features based on bigrams are varied because not all bigrams appear in the data. The comparative results show that bi-gram and frequent multi-gram based features improves the final results. In terms of video-level, the mean accuracy of Reynolds numbers classification of all models and segments reaches 89.9% and 94.4% if we exclude the evidently disturbed bottom segments; on sequence-level score of Reynolds numbers classification, our best mean accuracy of all models reaches 81.17% in downward segments (there are common patterns in different Reynolds numbers).

Figure 1.15 shows that a large number of droplets and bubbles interfere with observation, as shown by the red arrow. Indeed, the entire appearance is vague and difficult to identify, which was also noted in some previous studies, presented by [82] and by [132], that abandoned the circular structure and adopted a straight structure cavity for visualization research to obtain clear images. However, as the Reynolds number increases, the image quality decreases significantly. One inevitable reason is that the cavity in the piston is circular, which would affect light refraction; in

Segment	Features	Accuracy	10568	13221	15830	18484	21137	23791	26398	29052	31704	Mean
UG_f	80	8/9	85.06	88.73	42.53	58.62	39.08	50.88	52.94	45.45	14.67	56.72
BG_f	5130	8/9	95.40	97.18	50.57	73.56	34.48	50.88	62.35	64.94	22.67	61.37
MG_f	$433^{5/3}$	8/9	95.40	94.37	52.87	78.16	34.48	63.16	62.35	61.04	21.33	62.87
UG_t	80	**9/9**	92.31	80.77	84.62	92.31	61.54	36.00	94.87	60.00	40.00	69.91
BG_t	4097	7/9	97.44	73.08	84.62	89.74	30.77	48.00	89.74	54.29	37.14	69.94
MG_t	$632^{3/2}$	**9/9**	97.44	80.77	89.74	92.31	53.85	48.00	100.00	57.14	62.86	77.53
UG_d	80	**9/9**	96.91	65.98	97.94	100.00	63.24	35.62	100.00	53.33	92.00	81.75
BG_d	4133	**9/9**	97.94	69.07	93.81	98.97	72.06	30.14	94.85	56.67	81.33	79.82
MG_d	$827^{3/2}$	**9/9**	100.00	72.16	93.81	100.00	67.65	32.88	96.91	56.67	88.00	81.94
UG_b	80	4/9	40.00	57.69	3.85	19.23	5.88	0.00	38.46	42.86	28.00	31.29
BG_b	3980	6/9	80.00	73.08	50.00	88.46	23.53	6.67	76.92	71.43	16.00	53.44
MG_b	$891^{3/3}$	8/9	66.67	69.23	61.54	84.62	29.41	53.33	88.46	61.90	28.00	60.40
UG_u	80	8/9	73.15	94.44	67.59	98.15	53.70	20.00	96.04	46.84	66.67	71.05
BG_u	4956	**9/9**	72.22	95.37	68.52	99.07	54.63	29.23	94.06	54.43	66.67	73.90
MG_u	$1764^{3/3}$	**9/9**	75.00	92.59	67.59	100.00	55.56	30.77	96.04	50.63	69.05	74.06

Table 1.4: Comparison of video-level Reynolds number (**10568** to **31704**) classification accuracy and sequence-level score w.r.t. unigram/bigram/multi-gram (UG/BG/MG) models and BN classifier, where $k = 80$. The values in bold are the best ones in the column.

Segment	Features	Accuracy	10568	13221	15830	18484	21137	23791	26398	29052	31704	Mean
UG_f	80	2/9	59.77	28.17	14.94	14.94	20.69	8.77	38.82	14.29	16.00	24.69
BG_f	5130	5/9	41.38	32.39	18.39	14.94	21.84	15.79	22.35	20.78	9.33	22.84
MG_f	$433^{5/3}$	**8/9**	25.53	31.03	39.44	17.24	24.14	22.99	22.81	32.94	28.57	22.67
UG_t	80	5/9	61.54	15.38	46.15	41.03	15.38	0.00	56.41	40.00	20.00	36.21
BG_t	4097	4/9	58.97	30.77	46.15	10.26	15.38	24.00	64.10	20.00	17.14	35.56
MG_t	$4871^{2/4}$	6/9	40.36	53.85	38.46	53.85	38.46	19.23	24.00	69.23	37.14	20.00
UG_d	80	6/9	40.21	63.92	43.30	27.84	26.47	0.00	50.52	5.00	6.67	33.90
BG_d	4133	**8/9**	36.08	59.79	44.33	48.45	42.65	21.92	59.79	20.00	24.00	39.19
MG_d	$12259^{2/6}$	7/9	49.48	60.82	50.52	53.61	39.71	16.44	71.13	11.67	22.67	45.99
UG_b	80	3/9	6.67	50.00	46.15	15.38	5.88	26.67	23.08	23.81	20.00	26.16
BG_b	3980	5/9	13.33	46.15	65.38	23.08	11.76	6.67	38.46	33.33	44.00	30.47
MG_b	$1416^{3/6}$	7/9	37.32	40.00	26.92	73.08	38.46	29.41	20.00	50.00	33.33	32.00
UG_u	80	3/9	26.85	62.04	12.04	21.30	14.81	9.23	46.53	18.99	35.71	29.18
BG_u	4956	**8/9**	28.70	62.04	18.52	32.41	24.07	24.62	43.56	32.91	33.33	32.09
MG_u	$9459^{2/6}$	7/9	37.96	50.93	36.11	32.41	29.63	21.54	57.43	31.65	40.48	37.94

Table 1.5: Comparison of video-level Reynolds number (**10568** to **31704**) classification accuracy and sequence-level score w.r.t. unigram/bigram/multi-gram (UG/BG/MG) models and SVM classifier, where $k = 80$. The values in bold are the best ones in the column.

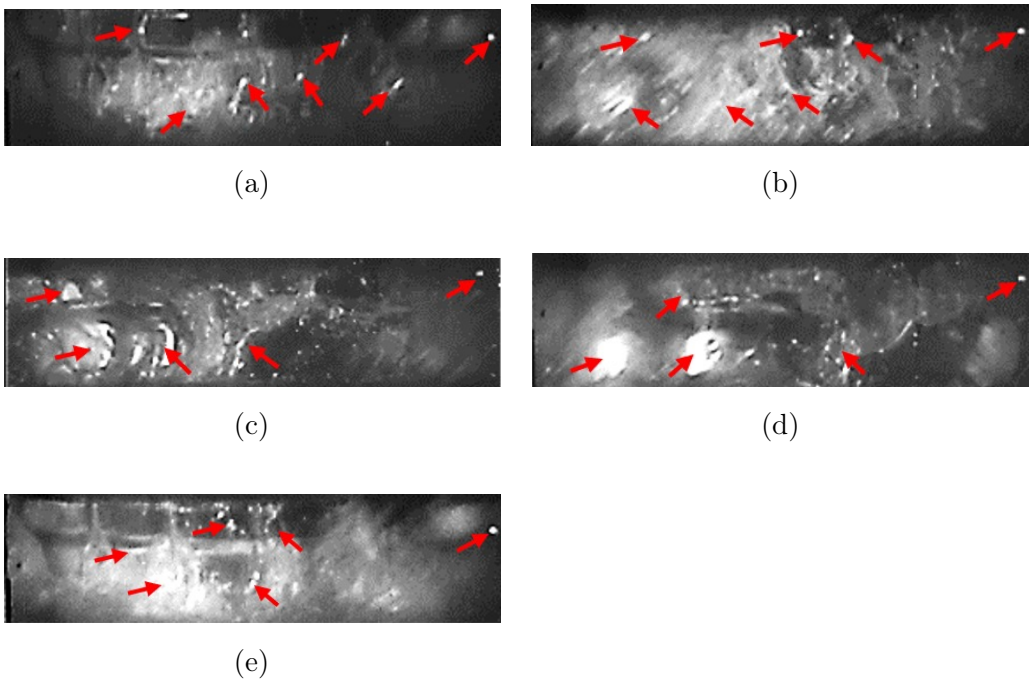


Figure 1.15: Oscillatory two-phase flow patterns when $R_e = 15830$ with respect to the angle of crank at (a) 0° , (b) 90° , (c) 180° , (d) 270° , and (e) 360°. Each image represents the 185mm × 25mm cooling gallery inside the cavity.

addition, the cavity's maximum relative velocity reaches 4.37 m/s when the calculated Reynolds number is about 20,000, and the splashed droplets and transient bubbles further damage the picture quality. Therefore, it is not easy to identify flow patterns directly via such pictures, and, nevertheless, the motion of the gas-liquid interface can be distinguished by observation at various angles. Considering that the optical flow algorithm has good flow feature extraction effect in the field of flow visualization, such as the work presented by [16] and by [78], it can be considered that the algorithm can be applied to the oscillatory two-phase flow pattern of continuous frames to generate offset vector.

In this study, the numerical simulations were conducted using the commercial software ANSYS FLUENT 19.0 combined with user defined functions (UDFs) as the main solver. The CFD simulation method used in this study draws on the results of the method used by [82]. The initial boundary conditions set in this section are described as the following: the injection hole is the flow inlet boundary condition; the flow rate is 0.025 kg/s; the bottom is the pressure outlet and the surface pressure is 0 Pa. Besides, the turbulence model used SST k-ω, and the two-phase flow model used VOF. Mixed grids were generated in the model, the total number of the grid

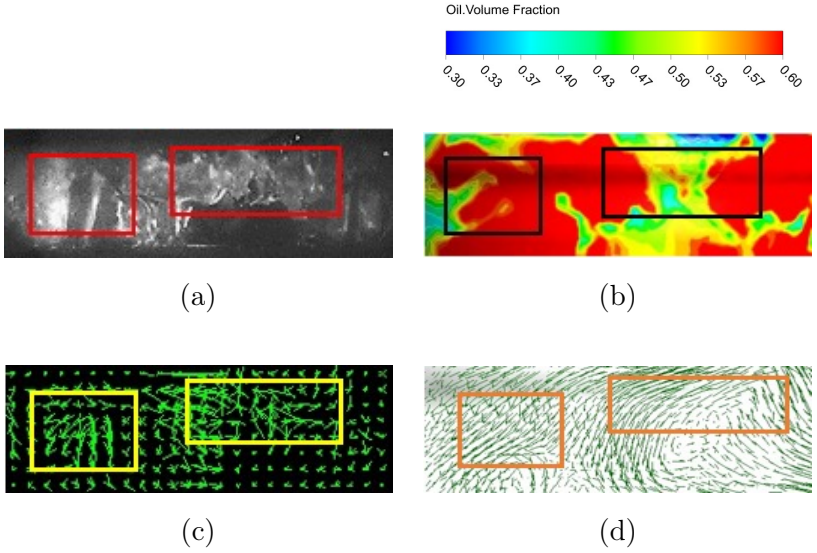


Figure 1.16: Comparison of contrast area at TDC position for optical flow images (c, yellow box areas) with volume fraction (b, black box areas) and liquid-phase velocity vectors (d, orange box areas) by CFD results at $R_e = 10568$ (a, red box areas).

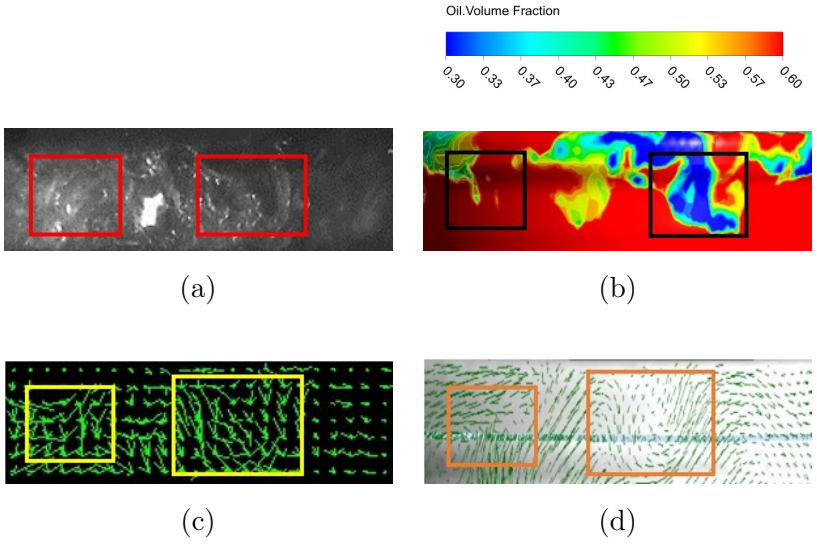


Figure 1.17: Comparison of contrast area at BDC position for optical flow images (c, yellow box areas) with volume fraction (b, black box areas) and liquid-phase velocity vectors (d, orange box areas) by CFD results at $R_e = 21137$ (a, red box areas).

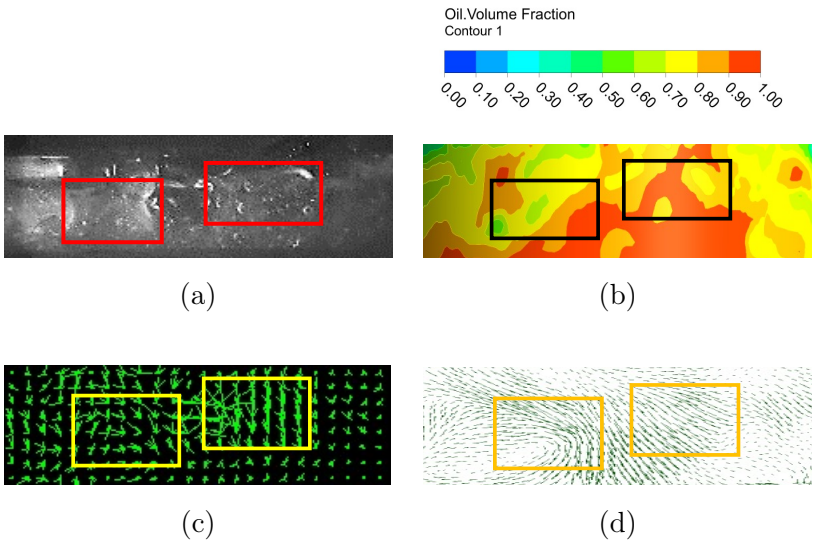


Figure 1.18: Comparison of contrast area at BDC position for optical flow images (c, yellow box areas) with volume fraction (b, black box areas) and liquid-phase velocity vectors (d, orange box areas) by CFD results at $R_e = 31704$ (a, red box areas).

elements was 1.86 million after the grid independence analysis. The transient calculation was adopted and the time step was set as the same as the moving time of 0.5° crankshaft angle.

Figure 1.16, Figure 1.17, and Figure 1.18 show representative comparison results between the optical flow and CFD at different Reynolds numbers when the cavity at the top dead center (TDC) or the bottom dead center (BDC) positions. Although some differences can be seen between the gas-liquid interface of the images and the volume fraction interface of the CFD results, the vectors obtained by combining images with the optical flow method and compared with velocity vectors by CFD can further support the reliability of the simulation results. The results show that the vortices in the flow patterns (a, red box areas) can be displayed accurately according to the optical flow image (c, yellow box areas) processed by the optical flow algorithm. At the same time, the results of CFD simulation are compared with respect to the flow rules calculated in the corresponding regions (b, black box areas; d, orange box areas), which indicate that the results have a good fitting effect in reflecting the trend in motion of oscillatory two-phase flow patterns.

1.3.5 Conclusion

Computer vision and machine learning methods were innovatively applied to oscillating two-phase flow pattern identification in providing more valuable analysis for visualization experiment results. Our classification assessed experimental results showed the performance of the presented work, which can help to obtain a clearer motion trend of oscillatory two-phase flow and provide verification for CFD simulation. The vectors obtained by combining images with the optical flow method could be taken as a reference to compare with the velocity vectors from the numerical simulation. The method provides a new strategy for non-contact two-phase flow field detection and can deeply analyze and excavate the visualization results of the oscillatory two-phase flow with complex flow patterns with high reliability and application value. The introduced research method can be widely applied in the visualization of multiphase flow which is a key area to be developed on the basic research of heat transfer systems. In our future work, we are interested in understanding and predicting complex two-phase flow patterns by probabilistic models.

1.4 Dynamic Texture Classification

In this section, we propose to tackle dynamic texture video classification as a sequence pattern mining problem. Indeed, dynamic textures are repetitive spatio-temporal patterns characterized by non-rigid and complex motions [115]. Extraction of spatio-temporal features and patterns may allow the characterization and classification of dynamic texture videos by deep learning approaches.

1.4.1 Problem Statement

Early hand-crafted spatio-temporal feature-based methods depend on optical flow [91, 107, 80, 27]. For example, in the work of [27], a motion/no-motion map is built and combined with mixed-state statistical models. However, optical flow-based methods, which lend themselves to the extraction of smooth motion fields, and thus might not represent dynamic textures well, which are usually made by chaotic motions in several directions. The time-evolving appearance of textures was explicitly modeled in [115], relying on a Linear Dynamical System (LDS). The method uses the model parameters as the input features. Another well-known family of hand-crafted features are spatio-temporal extensions of the Local Binary Pattern (LBP) method, such as CVLBP [123] or 2D LBP computed with 3 orthogonal planes (LBP-TOP) [150]. [94] come up with a combination of Gaussian filters and LBP patterns, where LBP descriptors are calculated from both blurred volumes and 3D difference of Gaussians volumes. Recently, a feature called

momental directional pattern (MDP) [95] has been developed.

Since the breakthrough of AlexNet [63], many deep learning approaches have been developed for dynamic texture classification. Some works are based on information that is purely spatial [109], where 2D convolution filters are applied to each frame of a video. In the work of [109], two-level strategy is proposed: utilizing the transfer learning to extract mid-level features and forming a video feature representation by concatenating the mean vector and the diagonal entries of the co-variance matrix of the mid-level features. In [6], feature extraction on 3 orthogonal planes based on convolutional neural networks is used, which achieves good results on many dynamic texture benchmark datasets. [52] introduce a learning-free ConvNet, operating on oriented 3^{rd} order Gaussian-based filtering, to extract features, fed to a classifier.

Beside machine learning approaches, pattern mining, especially sequence pattern mining methods, have also been used to tackle various computer vision problems, such as image classification [98, 145, 146] or action recognition [45, 97, 130]. For example in [98], frequent itemset patterns are used to create a bag-of-visual-words representation [42], for both unsupervised image ranking and supervised image classification. In [42], frequent itemset patterns are used as mid-level features for image classification. For video data, [97] propose a pattern-growth mining method to extract interesting sequence patterns. These patterns are then used as a bag-of-words to construct feature histograms, to classify human actions. The presented methods use a hard assignment from input observations to items and patterns, which can lead to unstable patterns and sensitive decisions that are not suited to the fuzzy nature of dynamic textures. In order to avoid such problems, [130] propose a framework for action recognition where patterns are sequences of soft-assignments, rather than itemsets themselves.

1.4.2 A Probabilistic Sequence Pattern Approach

In this section, we propose a novel representation of dynamic texture data, with video classification as final purpose, based on sequence pattern mining. The proposed framework uses a sequence pattern mining algorithm to discover frequent subsequences of image patches, that are referred to as key motifs (in the rest of this section, we use the term *motif* as *pattern* with respect to the video learning domain), inspired from [130].

In the training phase, first, a set of symbols, corresponding to representative image patches, is extracted by unsupervised clustering as the main modes of patch distribution. Then, for each video class, a set of key motifs is constructed: this extraction handles both deterministic sequences (hard-assigned) and probabilistic sequences (soft-assigned). Finally, the sets of key motifs of the different classes are merged into a global one, in which each class is fairly represented. In the

inference phase, probabilistic soft-assigned sequences are extracted from the video to be classified. The match between these probabilistic sequences and the key motifs of the unified set, makes a feature vector fed to the final classifier. The experimental analysis studies the use of this motif-based representation for video classification with an extensive parameter study, considering the impact on the number of key-motifs and the impact on classification performance. We emphasize the fact that the proposed method does not follow the trend of convolutional neural networks, and more generally deep learning approaches.

Consider the task of video classification in C classes, we introduce an efficient framework that is based on a novel mining method of *p-sequences*, which are sequences of probability vectors. The proposed method is divided into three main stages:

- Unsupervised clustering of patches using a Gaussian mixture model, and construction of p-sequences.
- Mining key motifs from p-sequences for each class by sequence pattern mining.
- Constructing feature vectors which are based on probabilistic supports of the union of key motifs and classification using a SVM with χ^2 kernel.

From a video $\mathbf{V} \in \mathbb{R}^{T \times H \times W}$, multiple non-overlapping patches of size $\sigma \times \sigma$ are extracted. For the training dataset, we build a set $\mathcal{X} = \{\boldsymbol{x}_i\}_{i=1,\dots,|\mathcal{X}|}$, where $\boldsymbol{x}_i \in \mathbb{R}^{\sigma^2}$ is a flat vector representation of the i^{th} patch, and all theses patches are used for calculating clusters. Beside K-Means clustering, Gaussian mixture models (GMM) are often used as clustering methods in computer vision to create dictionaries of visual symbols, where the parameters of the GMM are estimated using the Expectation-Maximization (EM) method. The Gaussian mixture model is made up by mixture weights $\pi_k \in \mathbb{R}$, means $\boldsymbol{\mu}_k \in \mathbb{R}^{\sigma^2}$ and covariances $\boldsymbol{\Sigma}_k \in \mathbb{R}^{\sigma^2 \times \sigma^2}$. The Probability Density Function at point $\boldsymbol{x}$ is:

$$p(\boldsymbol{x}|\boldsymbol{\theta}) = \Sigma_{k=1}^{K} \pi_k \mathcal{N}(\boldsymbol{x}|\boldsymbol{\mu}_k, \boldsymbol{\Sigma}_k),\ 0 \leq \pi_k \leq 1,\ \Sigma_{k=1}^{K} \pi_k = 1 \tag{1.5}$$

where $\boldsymbol{\theta}$ is the collection of all parameters of the GM (mixture weights, means and covariances). The expectation maximization algorithm (EM) is used as a learning method to update the parameters in the GM model [96]. The means computed by the EM algorithm form the dictionary of K symbols, hence, the clustering is done on the whole training dataset $\mathcal{X}$, regardless of video classes.

After having learned the parameters of the model, a soft assignment is used rather than a hard assignment. Where, a data point is represented by a vector of posterior probabilities rather

Algorithm 1: Expansion Algorithm

Input: $\mathcal{T}^l = \{\boldsymbol{T}_1^l, \boldsymbol{T}_2^l, \ldots, \boldsymbol{T}_{|\mathcal{T}^l|}^l\}$: set of $l-$motifs

Output: $\mathcal{T}^{l+1}$: set of $l+1-$motifs

```
T^{l+1} ⟵ ∅
for i = 1, 2, ..., |T^l| do
    tail ⟵ T_i^l(2 : |T_i^l|)
    for j = 1, 2, ..., |T^l| do
        head ⟵ T_j^l(1 : |T_j^l| − 1)
        if tail = head then
            T^{l+1} ⟵ T^{l+1} + concat(T_i^l, T_j^l(|T_j^l|))
        end
    end
end
```

than an assignment to a cluster. For a given data point $\boldsymbol{x}$, the posterior probability $p(k|\boldsymbol{x})$, i.e., the probability that $\boldsymbol{x}$ belongs to the k^{th} cluster, is

$$p_k(\boldsymbol{x}) = p(k|\boldsymbol{x}) = \frac{\pi_k \mathcal{N}(\boldsymbol{x}|\boldsymbol{\mu}_k, \boldsymbol{\Sigma}_k)}{\Sigma_{j=1}^{K} \pi_j \mathcal{N}(\boldsymbol{x}|\boldsymbol{\mu}_j, \boldsymbol{\Sigma}_j)} \tag{1.6}$$

According to Equation 1.6, the soft assignment of a patch of size $\sigma \times \sigma$ is represented by a probability vector $\boldsymbol{p} = [p_1, p_2, \ldots, p_K] \in [0,1]^K$. At a given spatial position, a sequence of consecutive patches forms a spatio-temporal patch of size $T \times \sigma \times \sigma$. By applying Equation 1.6 to each 2D patch of that sequence and each cluster k, the p-sequence $\boldsymbol{P}$ of length T is built, such as: $\boldsymbol{P} = \langle \boldsymbol{p}^1, \boldsymbol{p}^2, \ldots, \boldsymbol{p}^T \rangle$ where each $\boldsymbol{p}^j (j = 1, 2, \ldots, T)$ is a probability vector in $[0,1]^K$. Then, $\mathcal{P}$ is the set of p-sequences to be mined, $\mathcal{P} = \{\boldsymbol{P}_1, \ldots, \boldsymbol{P}_{|\mathcal{P}|}\}$. Algorithm 1 demonstrates how super-sequence patterns can be generated from shorter mined patterns. From patterns containing l symbols, referred to as l-motifs, candidate patterns of $l+1$ symbols are generated. For a particular case, an item can be considered as a 1-motif composed of one symbol in the dictionary.

At a given position i in $\boldsymbol{T}$ and j in $\boldsymbol{P}$, the probabilistic support function $f(\boldsymbol{T}(1:i), \boldsymbol{P}(1:j))$ is calculated using Equation 1.7:

$$f(\boldsymbol{T}(1:i), \boldsymbol{P}(1:j)) = p_{t(i)}^j \times \max_{k \in \{j-g, \ldots, j-1\}} f(\boldsymbol{T}(1:i-1), \boldsymbol{P}(1:k)) \tag{1.7}$$

which can be implemented using dynamic programming. When finished constructing the dynamic

Algorithm 2: Key Motif Mining Algorithm

Input: $\mathcal{P}^c$: *p-sequences* set of class c, ϵ: support threshold

Output: Mined key motifs of the class c – $\mathcal{T}^c$

$\mathcal{T}^1 \longleftarrow \{1-\text{motifs}\}$
$l \longleftarrow 2$
while $\mathcal{T}^{l-1} \neq \emptyset$ **do**
  $\mathcal{T}^l \longleftarrow \text{expand}(\mathcal{T}^{l-1})$
  for $i = 1, 2, \ldots, \left|\mathcal{T}^l\right|$ **do**
    support $\longleftarrow 0$
    for $j = 1, 2, \ldots, |\mathcal{P}^c|$ **do**
      support $\longleftarrow$ support $+ \eta\left(\boldsymbol{T}_i^l, \boldsymbol{P}_j^c\right)$
    end
    if $\frac{support}{|\mathcal{P}^c|} \leq \epsilon$ **then**
      Remove $\boldsymbol{T}_i^l$ from $\mathcal{T}^l$
    end
  end
  $l \longleftarrow l + 1$;
end

table, the probabilistic support is the maximum value of the last row of the table: $\eta(\boldsymbol{T}, \boldsymbol{P}) \longleftarrow \max_{j \in \{1,\ldots,|\boldsymbol{P}|\}} f\left(\boldsymbol{T}\left(1 : |\boldsymbol{T}|\right), \boldsymbol{P}\left(1 : j\right)\right)$.

A video sequence contains multiple non-overlapping video subsequences of size $T \times H \times W$. Each video subsequence consists of several non-overlapping spatio-temporal patches (represented by the $\boldsymbol{P}$ using Equation 1.6). In this part, a bag-of-motifs method is used from the mined motifs of all classes. The process is divided into 3 steps: (a) uniting mined motifs of each class; (b) constructing probability histograms for each video sample; and (c) classification learning. After having mined sequential motifs using Algorithm 2, a union of the motifs of all considered classes is made. The union set of mined motifs of all classes is $\mathcal{T}_\cup = \bigcup_{c=1}^{C} \mathcal{T}_c$. The previous mining of key motifs on a per-class basis ensures that, in this union, each class is sufficiently represented. If key motifs have been extracted regardless of classes, we could end up with a set where some classes would be much more represented than other ones when dealing with unbalanced datasets. A bag-of-feature vector is generally a histogram that counts the number of occurrences of each symbol in a sample [68, 114, 79]. However, in our case, a feature vector is built based on the probabilistic supports. For a given test p-sequence $\boldsymbol{P}$, a feature vector $\boldsymbol{f}$, is made up of the

probabilistic supports between each key motifs of the union set $\mathcal{T}_\cup$ and $\boldsymbol{P}$. Hence, we have $\boldsymbol{f} \in \mathbb{R}^{|\mathcal{T}_\cup|}$. A feature vector of a video subsequence is calculated using the average of the $\boldsymbol{f}$'s of the p-sequences in the video subsequences.

There exists several differences between our method and [130]. Since the application is different, we use image instead of 3D body joint coordinates. We extract symbols by fitting a GMM, while [130] use the activated simplices method [129]. Contrary to what is done in [130], the mining is done with a fixed threshold ϵ. Indeed, thanks to the modification mentioned above in Algorithm 2, there is no need to change ϵ during the mining process in order to obtain a given number of key-motifs. Our experiments show that this threshold does not play a crucial role for accuracy but only for compactness of the model (number of extracted motifs). Lastly, [130] classify a test example by assigning it to the class which has the key motifs leading to the highest probabilistic supports.

The final Fourier transform magnitude patch $\boldsymbol{x}^{FT}$ is computed using the following procedure: (1) each gray-scale patch is transformed into the frequency domain using the FFT algorithm which is widely used for computing the Fourier transform of a patch $\boldsymbol{x}^{FT} \in \mathbb{C}^{\sigma \times \sigma}$; (2) a top left quadrant of the FFT patch is taken as it is symmetric along the horizontal and vertical axis, so the FFT patch has a size of $\mathbb{C}^{\frac{\sigma}{2} \times \frac{\sigma}{2}}$; (3) the magnitude of the FFT patch is computed as $\boldsymbol{x}^{FT} \in \mathbb{R}^{\frac{\sigma}{2} \times \frac{\sigma}{2}}$.

1.4.3 Experimental Evaluation

We report the results obtained by our proposed method on the benchmark datasets **Traffic** and **UCLA** while comparing to the state-of-the-art methods.

Traffic dataset [17] consists of 254 video samples. The sequences are recorded with a resolution of 320×240 with a temporal length between 42 and 52 frames, at 10 fps. Each video sample is then spatially resized to 80×60 and cropped to a size of 48×48 over the area where the motion is the most prominent. There are 3 classes in this database: heavy, medium and light traffic. For this dataset, a 4-fold evaluation protocol is used [17]. The average score of the 4-fold is recorded as the final result.

UCLA dataset [115] consists of 50 classes, where each class contains 4 sequences. As the result, the dataset has 200 dynamic textures sequences in total. Each original sample is captured with 75 frames, each of size 160×110. A slightly modified version of UCLA dataset is often utilized for dynamic texture classification where the original samples into sub-sequences of size 48×48. Three popular challenges of this dataset [125, 110, 95] are often used for classification. Note that the GM model is initialized randomly. To assess the stability of our method against

Datasets	UCLA			Traffic
Method	50-4fold	9-class	8-class	
3D-OFT [140]	290	290	290	–
HLBP [124]	1536	1536	1536	–
MEWLSP [125]	1536	1536	1536	–
RI-VLBP [151]	16384	16384	16384	16384
LBP-TOP [150]	768	768	768	768
CDT-TOP [48]	75	75	75	75
CNDT [49]	144	420	336	336
DM [110]	169	169	169	297
MDP [95]	3880	3880	3880	–
Gray-scale	651	3610	1196	118

Table 1.6: Comparison of feature vector dimension for dynamic texture classification.

the choice of symbols (that depends on these random selections), the impact on the results is evaluated over 8 runs for each split of the N-fold validation.

The minimum support threshold ϵ (algorithm 2) is an important parameter. Therefore, an analysis could be useful to study the impact of such parameter ϵ on the number of mined motifs. Consequently, during the 8 runs, an analysis of parameters is also conducted to evaluate the impact on both accuracy and the number of motifs. This process is done for each set of parameters on the Traffic dataset and UCLA dataset. This analysis is only done on the gray-scale patches.

In the training and testing stage, the spatio-temporal patches are extracted at every possible position with a non overlapping condition. For each dataset and their configurations, both gray-scale patches and FFT patches are evaluated.

To Traffic database, looking at figure 1.19, the best results are obtained with a patch size of 8 (compared to 12). This is the most important parameter. Figure 1.19 shows that the number of motifs generated from codebook of sizes 20 and 50 are almost the same. Furthermore, the performances based on these codebook sizes are very close. For the minimum support threshold, good results can be achieved with a minimum support between 0.01 and 0.07. However, the number of mined key-motifs increases as the minimum support threshold decreases. Therefore, rather than using 500 or 600 key motifs with ϵ of 0.01, high results (eg. 96,56% of accuracy) can be achieved by using approximately 100 key motifs with an $\epsilon = 0.05$. Column 5 of table 1.7 shows the performance of the proposed method for the Traffic database. The proposed method

Datasets	UCLA			Traffic
Method	50-4fold	9-class	8-class	
KDT-MD [18]	97.5	–	–	–
DFS [139]	89.5	–	–	–
CFV [134]	–	85.10	85.00	–
3D-OFT [140]	87.10	97.23	99.50	–
HLBP [124]	95.00	98.30	97.50	–
MEWLSP [125]	96.50	98.50	98.04	–
GoogleNet [6]	99.50	98.35	99.02	–
AlexNet [6]	99.50	98.05	98.48	–
CVLBP [123]	93.00	96.90	95.65	–
RI-VLBP [151]	77.50	96.30	91.96	93.31
LBP-TOP [150]	95.00	96.00	93.67	93.70
CDT-TOP [48]	95.00	96.33	93.41	93.70
MDP [95]	100.00	98.90	98.70	–
V-BIG [94]	99.50	97.95	97.50	–
CNDT [49]	95.00	95.61	94.32	96.46
DM [110]	98.50	97.80	96.22	96.60
Gray-scale	98.50	96.69	95.45	96.56
FFT	99.50	96.55	95.54	98.44

Table 1.7: Comparison of recognition rates (%) on the UCLA dataset and Traffic database.

outperforms most of existing methods on this dataset and it is even on par with the diffusion-based method [110] with only 0.04% less (96.60% and 96.56%) with the average accuracy over 8 runs. Comparing to the diffusion-based model methods [48, 49], our motif-based method shows better results by at least 1%. Looking at the results obtained by the LBP-based methods [150, 151], it classifies the dataset much better by up to almost 3% being more compact in terms of feature vector size, which can be seen with Table 1.6 (16384[151] and 768[150] to 118 with our approach). In addition, the result obtained by using a quadrant of FFT image as input is very high with 98.44% of accuracy (table 1.7). It is currently the highest score for this dataset. This result shows the efficacy of the proposed method for the Traffic dataset.

To UCLA 50-class, best results are obtained with the patch size of 8 (figure 1.20). But this time, the accuracy is better with 50 clusters than with 20 clusters. This can be explained by the complexity of textures occurring in the videos. For the minimum support, good results are

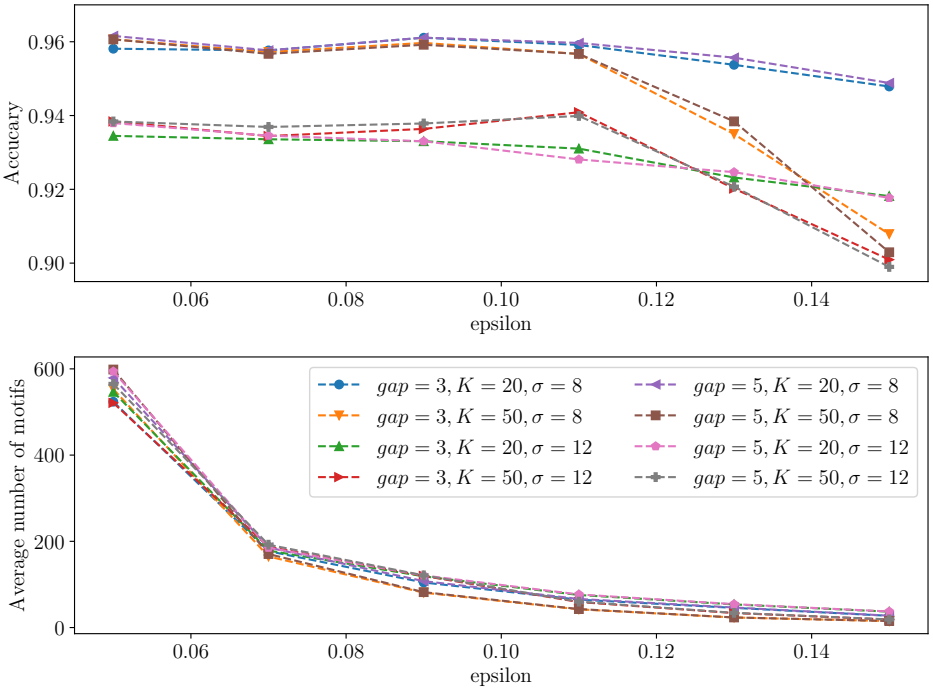


Figure 1.19: An analysis of the stability of the method in terms of minimum support threshold and the number of mined motifs for the traffic dataset.

achieved with ϵ between 0.19 and 0.3. In figure 1.20, the averaged highest result is obtained with only 651 motifs. From table 1.7, the proposed method achieves 98.50% of the classification rate on this subset as the average score over 8 runs. It performs on par with the diffusion-based model (DM) [110] and outperforms other recent existing methods for example HLBP [123] or MEWLSP [125] as well as many other hand-craft methods. Nonetheless, in comparison with the DL method, the mining of key motifs method is just 1% less than the existing DT-CNN models [6]. Furthermore, in comparison to a recent LBP-based method, MDP [95] which scores 100% of accuracy for this subset, our method is just 1.5% less, with a higher compacity from 3880 to 651 features (see Table 1.6). When FFT patches are used, the result (99.50%) increases by 1% in terms of accuracy and even on par with the DL approaches.

To UCLA 9-class and 8-class, Figures 1.21 and 1.22 illustrate the parameters analysis of the UCLA 9-class and UCLA 8-class schemes. The tendency of classification rates is approximately the same as the UCLA 50-class and Traffic dataset as good results are obtained with ϵ ranging in the middle of the analyzing interval. The number of motifs begins to converge as ϵ passes 0.15. From table 1.7, our model outperforms some of the other classical computer vision methods of

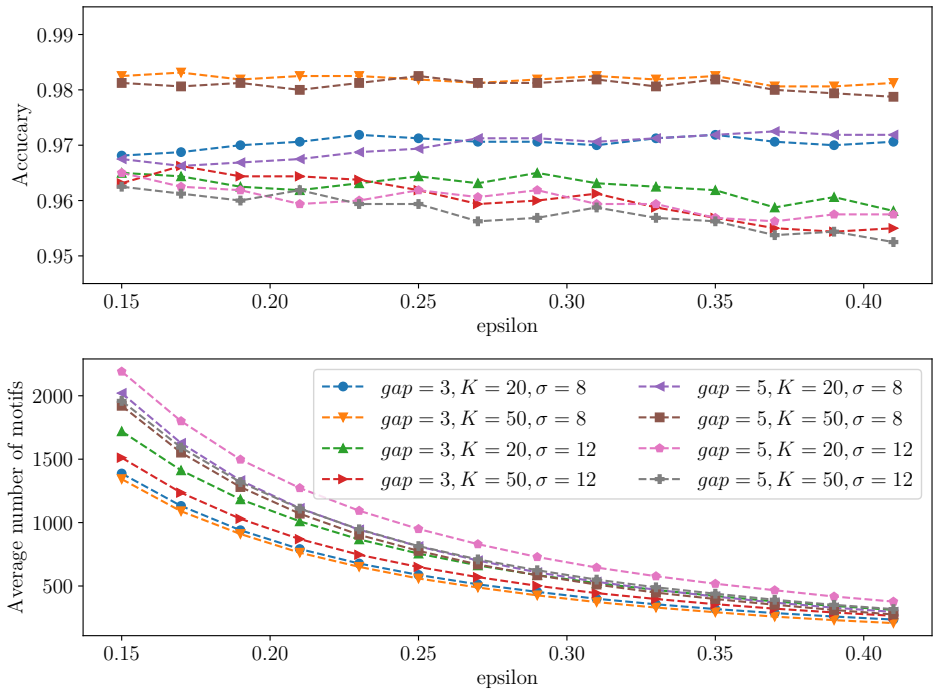


Figure 1.20: An analysis of the stability of the method in terms of minimum support threshold and the number of mined motifs for the UCLA 50-class dataset.

about 1% or even better than CFV [134] of about 10% for the 9-class configuration. Nevertheless, it cannot achieve as good results as DL models [6] or a recent vision-based model of MDP [95] which are at 98.35% and 98.90% respectively. In the 8-class configuration, our method reaches a result of 95.45% of accuracy. This result is higher than many methods such as CDT-TOP [48], CVLBP [123] or CFV [134]. Nonetheless, the best results for this configuration is obtained with GoogleNet [6] with an accuracy of almost 100%. Plus, using only one quadrant for this scheme also results in good result (96.55%) but not as high as with grayscale images as inputs. Moreover, UCLA 8-class with FFT patches helps improving the performance by only 0.1% (95.54% in accuracy) comparing with the grayscale patches. Overall, the results, even if not the best, are satisfactory considering both accuracy and compacity (Table 1.6).

In general, the performance of our proposed approach is comparable to the state-of-the-arts methods. As the overall accuracy of the experiments, different sets of parameters are tested in order to measure the impact of each parameter for each module. The two most important parameters in our approach is the patch size σ and the number of cluster K. A smaller patch size of 8 with a greater number of cluster of 50 shows the best performance. Moreover, a maximum

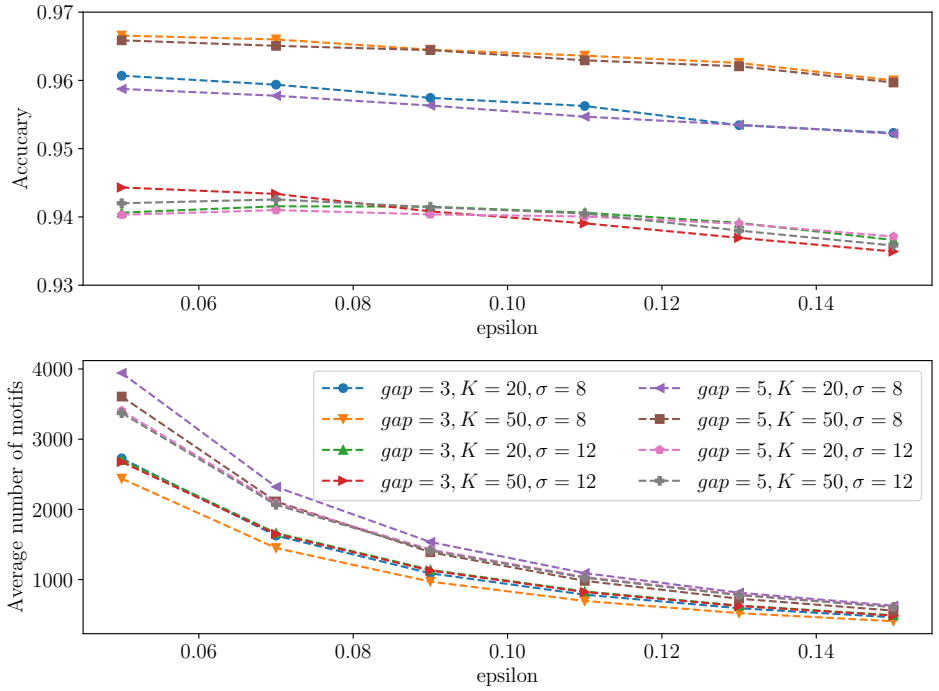


Figure 1.21: An analysis of the stability of the method in terms of minimum support threshold and the number of mined motifs for the UCLA 9-class dataset.

gap constraint also contributed to the performance of the approach. A smaller gap generates less motifs than a bigger gap (a maximum gap constraint of 3 and 5) but with slightly better accuracy, which means that the set of extracted key motifs using a gap of 3 are more meaningful and more discriminant than with a gap of 5. As for the minimum support threshold (directly related to the number of motifs), increasing gradually this value can help us reach an optimal value and achieve highest result.

1.4.4 Conclusion

Experimental classification results show that our key motifs-based representation is relevant for dynamic textures. Although mining motifs using p-sequences can be effective, but not all mined motifs are useful for visual representation of video data. Therefore, for the perspective, motif filtering could be applied in order to extract less but more useful motifs. This addition could also help speeding up the classification step and reduce the number of elements in the feature vector.

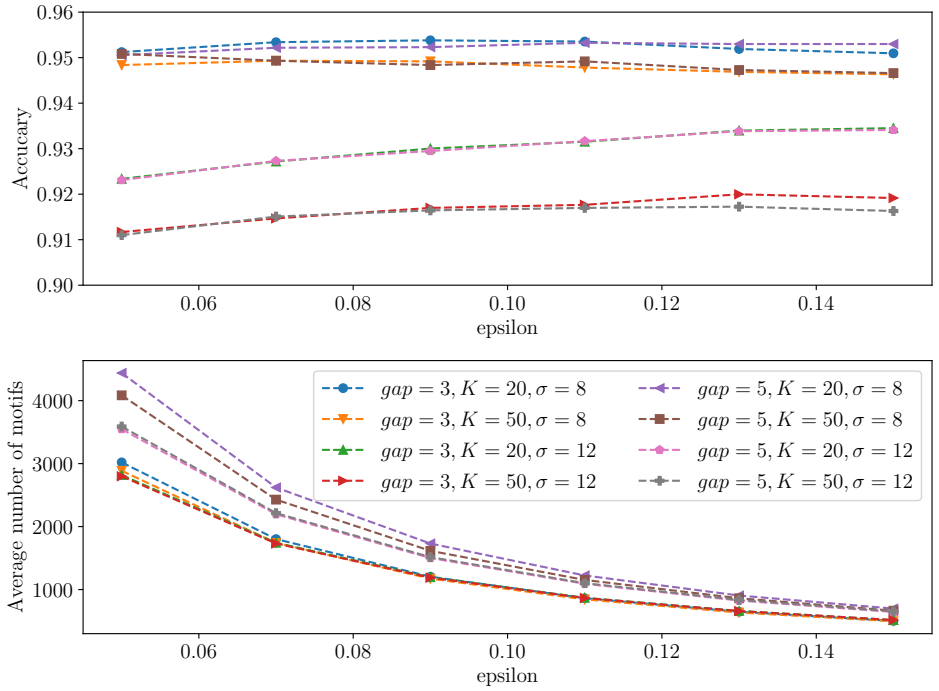


Figure 1.22: An analysis of the stability of the method in terms of minimum support threshold and the number of mined motifs for the UCLA 8-class dataset.

1.5 Perspectives

In the topic of this chapter, my main perspective in pattern mining is centered around the applications of patterns to machine learning problems as *effective features* with *explainable learning* [111, 51, 7]. Although frequent patterns are often used as features in supervised or non-supervised machine learning problems [75, 39, 35, 58, 93], however, with the varies of the minimum support threshold, the number of returned frequent patterns (including closed patterns or pattern generators) is uncontrollable: a too high threshold makes the mining process returning too few patterns and a too low threshold causes too much patterns, both cases have serious impacts to build an effective but small learning model. Hence, the mining and/or selection of *interesting* patterns are not less important than mining patterns faster or less.

To achieve the above perspective, I am interested in using explainable learning methods such as `LIME` [111] to build an objective interestingness measure system for prefix-based iterative pattern mining without human interactions, and in reverse, such interesting patterns are able to explain how the learning decision has been made by the model. This perspective is also cooperated with the team of Professor Yuqi Huang at Zhejiang University (China) in energy

engineering and with the team of Senior Researcher Dr. Jisheng Zhao at the University of New South Wales (Australia) in fluid mechanics to resolve related learning problems.

On the other hand, my perspectives in pattern mining also cover the mining of frequent sequence patterns in the context of Big Data, with the MapReduce [30] programming model or the Apache Spark[6] framework. There is an in working research in cooperation with Mostafa Bamha in the LIFO Laboratory at the University of Orléans. The basic idea is to study the use and the cost of prefix-projected databases of sequences in a distributed environment, typically in HDFS (Hadoop Distributed File System in the Apache Hadoop Framework[7]).

[6] `https://spark.apache.org`

[7] `https://hadoop.apache.org`

Chapter 2

Towards Efficient Skyline Query Processing

2.1 Introduction

The *skyline query processing problem* has received intensive attention from the database community since the first introduction of the *skyline operator* [12] in 2001.

Given a set of multidimensional points, the *skyline query* returns the *skyline* that is the set of all non-dominated points. A point p_i is said *non-dominated* if there is no any other point p_j such that p_j is better than p_i in all dimensions with respect to a user defined preference order. Figure 2.1 shows a skyline example that is very commonly used in the literature: assume a set of hotels where we want to select the ones with minimized price (Y axis) and distance from the beach (X axis), then the hotels $\{a, c, e, h, m\}$ form the skyline because no other hotels can be better than them on both of the price and the distance from the beach.

In the skyline computation problem, we consider a dataset $\mathcal{P}$ of N d-dimensional points, where we call N the *cardinality* and d the *dimensionality* of the dataset. Let p be a point, we denote $p[i]$ the *dimension value* of p in the dimension i, where $1 \leq i \leq d$. We consider the *preference order* as a total order in each dimension of points for the skyline. Without loss of generality, the preference order can be defined as the relation $<$ on the values in each dimension. Given two points p and q, $p[i]$ is *better than* $q[i]$ if $p[i] < q[i]$; $p[i]$ is *equal to* $q[i]$ if $p[i] = q[i]$; and $q[i]$ is *not worse* than $p[i]$ if $p[i] \leq q[i]$. To simplify the formal description, in the rest of this paper, any *dimension* refers to an integer value in the range $[1, d]$.

Definition 9 (Dominance)**.** *A point* p dominates *a point* q*, denoted by* $p \prec q$*, if and only if in each dimension* i *we have* $p[i] \leq q[i]$*, and in at least one dimension* k*,* $1 \leq k \leq d$*, we have*

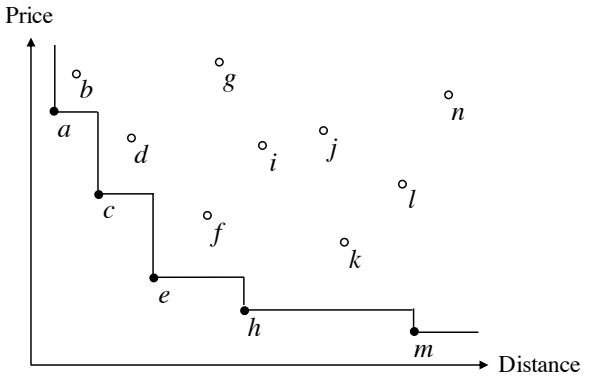


Figure 2.1: An example of skyline.

$p[k] < q[k]$. ■

Given two points p and q, we denote $p \not\prec q$ that p does not dominate q; we denote $p \nsim q \iff (p \not\prec q) \wedge (q \not\prec p)$ that p and q are *incomparable*. We extend $\{\prec, \not\prec, \preceq, \nsim\}$ to the set of points:

- $p \prec X$ (or $p \preceq X$) denotes $\forall q \in X, p \prec q$ (or $p \preceq q$);
- $X \prec p$ (or $X \preceq p$) denotes $\exists q \in X, q \prec p$ (or $q \preceq p$);
- $X \not\prec p$ (or $X \not\preceq p$) denotes $\nexists q \in X, q \prec p$ (or $q \preceq p$);
- $p \nsim X$ and $X \nsim p$ denote $\forall q \in X, p \nsim q$.

Definition 10 (Skyline)**.** *Given a dataset $\mathcal{P}$, a point $p \in \mathcal{P}$ is a* skyline point *if and only if $\nexists q \in \mathcal{P}$ such that $q \prec p$. The* skyline *of $\mathcal{P}$ is the complete set of skyline points $\{p \in \mathcal{P} \mid \nexists q \in \mathcal{P}, q \prec p\}$.* ■

The simplest way to compute the skyline is nested loop-based pairwise comparison: for each point p_i in the dataset, compare p_i with each other point p_j, if p_i dominates p_j, then drop p_j; if p_j dominates p_i, then drop the point p_i and break the nested loop; otherwise, continue the nested loop while keeping both p_i and p_j. Such a nested loop procedure finally outputs the set of all non-dominated points in $\mathcal{O}(dN^2)$ time where d is the number of dimensions in each point and N is the number of points in the dataset, if we consider $\mathcal{O}(d)$ time for testing the dominance relation (*dominance test*) between two points of d dimensions.

The dominance tests are the major cost of skyline computation. In order to efficiently resolve the skyline computation problem, many algorithms have been designed and developed based on the reduction of dominance testes, which can be categorized into two classes [65]: sorting-based

(such as BNL [12], Index [118], SFS [24, 25], LESS [46, 47], SaLSa [9, 10], BSkyTree-S [64, 65]) and partitioning-based (such as D&C [12], NN [62], BBS [99, 100], LS [86], OSPS [149], ZSearch [67], and BSkyTree-P [64, 65]). Besides, indexing techniques are also applied to skyline algorithms, such as Index, BBS, and ZINC [74] with different mechanisms.

The challenge that we tackled in the Erasmus Mundus BDMA International Master internship of Rui Liu, supervised by myself, is to develop a new scalable and extensible framework SDI that can be used in different skyline problems, in general the skyline computation in static datasets [77] or in streaming data [76], and can be extended to subspace queries, to top-k queries, and to resolve all such skyline problems in big data, as described in the perspective. With the presented results, the Master Theses of Rui Liu has been selected as the Best Master Thesis Award Erasmus Mundus 2019 in Computer Science. In this work, detailed in [77] and [76], we first proposed a sorting and indexing based framework SDI for efficient skyline computation in high-dimensional domains, and then, we successfully extended this framework to data streams.

By studying existing skyline algorithms, I independently proposed a new subset query approach to boost sorting-based skyline computation, detailed in [69], that makes SDI and SaLSa being the fastest skyline algorithms on uniform independent data.

I note that both two synthetic and real-world datasets have been considered and tested in the approaches presented in this chapter. The synthetic datasets consist of *anti-correlated* (AC), *correlated* (CO), and *uniform independent* (UI) with respect to the characteristics of real data [12][1] The real-world datasets include HOUSE (6-D, 127,931 points, 5,774 skyline points), NBA (8-D, 17,264 points, 1,796 skyline points), and WEATHER (15-D, 566,268 points, 26,713 skyline points) [22][2].

The rest of this chapter is organized as follows. In Section 4.2, I detail the SDI algorithm. In Section 4.3, an extension of SDI for skyline maintenance in streaming data is introduced. Section 4 presents my most recent subset approach to efficient skyline computation. Section 5 gives the perspective in order to further improve the processing of different skyline queries including in the context of big data.

[1] All concerned synthetic datasets are generated by Skyline Benchmark Data Generator from http://pgfoundry.org/projects/randdataset

[2] https://github.com/sean-chester/SkyBench

2.2 SDI: A Scalable Framework for Skyline Computation

2.2.1 Sorting and Indexing Dimensions

We present a dimension indexing based algorithm for skyline computation in this section. We first show that the dominance tests required to determine a skyline point can be sufficiently bounded to a subset of the current skyline, and then propose the algorithm `SDI` (`Scalable Dimension Indexing`), of which the time complexity is better than the best known algorithm in high-dimensionality domains with reasonably low cardinality. Our performance evaluation on synthetic and real datasets shows the efficiency of `SDI` in high-dimensionality domains as well as in low-dimensionality domains.

By indexing all dimensions, it is sufficient to test a point only with the existing skyline points on an arbitrary dimension instead of with the complete set of skyline points. We also show that any skyline point can be used as a *stop line* that traverses the indexed dimensions to stop the computation, which is much performant than the calculation of *stop point* mentioned in `SaLSa` [9, 10]. Furthermore, `SDI` adopts the *weak incomparability checking* to take the incomparability between points into account, which is the most important feature of latest skyline algorithms. Our analysis shows that the worst time complexity of `SDI` is better than the best known one [112] in high-dimensionality domains with reasonably low cardinality, and our performance evaluation shows that `SDI` outperforms the state-of-the-art `BSkyTree` algorithm[3] on high dimensional data, but less efficient than `BSkyTree` on low dimensional data.

Definition 11 (Dimension Index)**.** *Given a database, the* dimension index*, denoted $\mathcal{I}$, is the set of d ordered lists on a preference order $\prec$, in which each list $I_i \in \mathcal{I}$ is a* dimensional sub-index *that contains all dimension values sorted with respect to $<$.* ■

Example 1. *Let us consider the 5 points $\{a, e, g, h, i\}$ presented in Figure 2.2 that shows a dimension index example consisting of 10 points and 6 sub-indexes, where the dashed lines link the point among the sub-indexes. Obviously, a is a skyline point, which can be independently concluded from I_1, I_2, and I_6 because no point can dominate a; if we regard only I_4, h is immediately a skyline point and in order to determine whether i, the second point in this subindex, is a skyline point, it is enough to compare i with h because any point x located after the position of i in I_4 cannot dominate i since we have $i[4] \prec x[4]$ in I_4. Nevertheless, if we focus on I_3, we see that $h \prec i$ and $i \prec h$ must be first tested in order to decide whether h or/and i shall be skyline point(s)*

[3]The `BSkyTree` algorithm that we have compared is indeed the `BSkyTree-S` algorithm included in the source code of SkyBench [22]: `https://github.com/sean-chester/SkyBench`

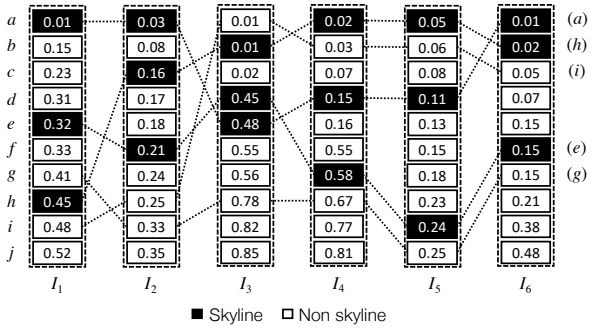


Figure 2.2: A dimension index example.

since $h[3] = i[3]$ *(indeed we have* $h \prec i$ *and* $i \not\prec h$*). That is also the case in* I_6*, where 3 points contain the same dimension value* 0.15*, so all these 3 points must be first locally compared in order to filter the potential skyline points (in our example, e). We call such points as* h *in* I_3 *and* e *in* I_6 *the* local skyline point. □

It is easy to see that any point is a local skyline point on a given dimension if there are no identical dimension values.

Lemma 1. *Let* $\mathcal{I}$ *be the dimension index of a database* $\mathcal{D}$ *and* $I_i \in \mathcal{I}$ *be an arbitrary subindex of the dimension* i*. Then, a local skyline point* t *is a skyline point if and only if (1) there is no point* u *such that* $u[i] \prec t[i]$*, or (2) for any skyline point* s *such that* $s[i] \prec t[i]$*, we have* $s \not\prec t$*.* ■

Proof. Let t be a local skyline point. If t is the first point in I_i, then t is a skyline point because no point is better than t. Otherwise, let s be a skyline point such that $s[i] \prec t[i]$, then if $s \prec t$, t cannot be a skyline point since s dominates t; if $s \not\prec t$, then there exists at least on dimension $k \neq i$ such that $t[k] \prec s[k]$ so no point dominated by s dominates t. Hence, for each skyline point s such that $s[i] \prec t[i]$, if we have $s \not\prec t$, then no point in the database dominates t, thus, t is a skyline point. If $t[i] \prec s[i]$, then $s \not\prec t$ so it is meaningless to compare t with s. □

Lemma 1 allows to determine whether a point is a skyline point only with a subset of the existing skyline. Furthermore, once the dimension index has been constructed, Lemma 1 allows to switch among sub-indexes so that the best one containing the least of known skyline points can always be selected.

In order to improve the efficiency of `SDI`, two heuristics are considered: the *stop line* to prune non-skyline points and the *incomparability checking* to avoid useless dominance tests.

Indeed, let p be a skyline point, then any point t such that $p \prec t$ can be pruned from the database in order to reduce the computation time. A *stop line* established from p, denoted by

L_p, is the set $\bigcup_{1\leq i\leq d} o_i(p)$ of the offsets of p in each dimensional subindex $I_i \in \mathcal{I}$. For a skyline point p, any point t such that $p \prec t$ can be pruned, and if `SDI` reaches p on each dimension, the computation can be safely terminated.

Lemma 2. *Let $\mathcal{I}$ be a dimension index of a d-dimensional database $\mathcal{D}$ and $p \in \mathcal{D}$ be an arbitrary skyline point. Let $o_i(p)$ denote the largest offset of any point x such that $x[i] = p[i]$ in the dimensional subindex $I_i \in \mathcal{I}$, if all offsets $o_i(p)$, $1 \leq i \leq d$, have been reached by following a top-down traversal on all dimensions, then the complete set of skyline points has been identified and the computation can be terminated.*

Proof. Let p and $t \in \mathcal{D} \setminus p$ be two skyline points in $\mathcal{D}$, we have: (1) $t \nsim p$; or (2) t and p have identical values on all dimensions. We denote $L_p = \bigcup_{1\leq i\leq d} o_i(p)$ the set of all offsets $o_i(p)$. In the first case, $t \nsim p \Rightarrow \exists k$ such that $p[k] \prec t[k]$, i.e. $o_k(p) < o_k(t)$, hence, if the index traversal reaches all offsets in L_p, t must have been identified at least in the dimension k. In the second case, we have $p[i] = t[i]$ on any dimension i. In both cases, if all offsets in L_p have been reached, all skyline points have been identified. □

`SaLSa` uses a sorting function to presort all points so that the last skyline point, called the *stop point*, can prune all the rest points. The main drawback of `SaLSa` is that the calculation of the stop point strongly depends on the sorting function and the find of an effective stop point cannot be guaranteed. We propose the notion of stop line that is based on the dimension index, of which the effectiveness can be guaranteed.

The *incomparability checking* is taken into account while a dominance test is proceeding. In our approach, we consider that a d-dimensional dominance test runs in $\mathcal{O}(d)$ time, so any point comparison better than $\mathcal{O}(d)$ time shall improve the efficiency of `SDI`. Indeed, to efficiently determine $s \nsim t$ in stead of testing $s \prec t$ in the case of $s \not\prec t$ is an essential time-costly task while comparing t with all existing skyline points in a dimensional subindex. In this paper, we propose a *weak* checking mechanism of the incomparability between a skyline point s and a testing point t in a dimensional subindex I_i as following.

Lemma 3. *Let $\mathcal{I}$ be a dimension index and s be a skyline point present in a dimensional subindex $I_i \in \mathcal{I}$. Given a point t such that $s[i] \prec t[i]$, we sufficiently have $s \nsim t$ if $max(s) > max(t)$, or $min(s) > min(t)$, or $sum(s) > sum(t)$.*

Proof. The sets L_s and L_p (see the proof of Lemma 2 for the definition) can be considered as the coordinates of two curves in a two-dimensional space. In Euclidean geometry, $max(s) > max(t)$, or $min(s) > min(t)$, or $sum(s) > sum(t)$ are sufficient conditions for the existence of at least

Algorithm 3: `SDI`

Input: Dimension index $\mathcal{I}$

Output: Skyline $\mathcal{S}$

```
1  while true do
2      I_best ← BestSubindex(I)
3      while T ← NextLocalSkyline(I_best) do
4          if T = null then
5              return S
6          foreach t ∈ T and t ∉ S do
7              if S_best ⊀ t then
8                  S_best ← S_best ∪ t
9                  S ← S ∪ t
10         if Found new skyline points then
11             Update StopLine
12             break
13     if StopLine is reached then
14         return S
```

one intersection of the curves formed by s and t since we have $s[i] \prec t[i]$, i.e. $o_i(s) < o_i(t)$, that is, according to the definition of dominance, $s \nsim t$. □

Note that the maximal value, the minimal value, and the sum of a point can be pre-calculated while constructing the dimension index, so Lemma 3 can efficiently determine $s \nsim t$. However, Lemma 3 shows in fact 3 sufficient conditions for $s \nsim t$, hence a dominance test is necessary to determine $s \nsim t$ in the cases that are not covered by Lemma 3, for which we call Lemma 3 a weak incomparability checking. The sketch of `SDI` is listed in Algorithm 3.

We can propose several extensions of `SDI`: (1) `SDI` computes the skyline in the categorical domains as long as the preference order $\prec$ can be defined; (2) `SDI` can be immediately adapted to subspace skyline computation by skipping unrelated dimensions; (3) `SDI` can be extended to the skyline maintenance by dynamically constructing the dimension index; (4) `SDI` can handle the top-k skyline query by finding the skyline points having the best positions in the dimension index.

2.2.2 Theoretical Analysis

We theoretical analyzed the performance of SDI. We denote d the dimensionality and N the cardinality of the data, and M the size of the skyline. In this analysis, we discuss SDI without duplicate values on any dimension, but if K duplicate values are present in a dimensional subindex, $\mathcal{O}(dK^2)$ shall be considered in assuming that BNL is applied to compute local skylines. Note that we consider $\mathcal{O}(d)$ time for a d-dimensional dominance test, which implies the tests of $s \prec t$ and $t \prec s$, hence, the dominance test within SDI is considered in $\mathcal{O}(d/2)$ time since $s \prec t$ is sufficient. We also note that the construction of dimension index requires $\mathcal{O}(dN \log N)$ time with respect to general $\mathcal{O}(N \log N)$ sorting algorithms on d dimensions, and we do not consider the two heuristics of stop line and incomparability checking.

The average time complexity of SDI is measured on M skyline points uniformly distributed in d dimensional sub-indexes.

The average time complexity of SDI is measured on M skyline points uniformly distributed in d dimensional sub-indexes.

Lemma 4. *Considering M skyline points,* SDI *computes the skyline of a d-dimensional database with the cardinality N in*

$$\mathcal{O}(dN \log N + \frac{M(2N - M - d)}{4}).$$

Proof. With M skyline points uniformly distributed on each dimension, $(N - M)/d$ non skyline points must be compared with M/d skyline points. For each dimension, in the worst case, $(M/d)(M/d-1)/2$ dominance tests are required by skyline points, $((N-M)/d)(M/d)$ dominance tests are required by non skyline points, and each dominance test cost $\mathcal{O}(d/2)$ time. Therefore, the average time complexity of SDI is $\mathcal{O}(((M/d)(M/d-1)/2+((N-M)/d)(M/d))(d/2)d)$, that is the result shown in Lemma 4. □

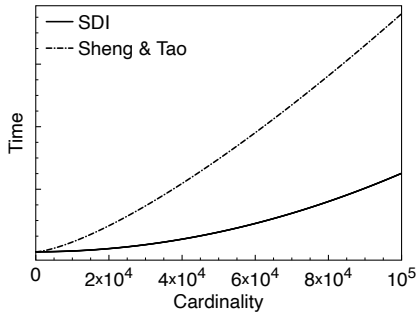
Lemma 5. *In the worst case, all N points in a d-dimensional database are skyline points.* SDI *computes the skyline in*

$$\mathcal{O}(dN \log N + \frac{N^2 - dN}{4}).$$

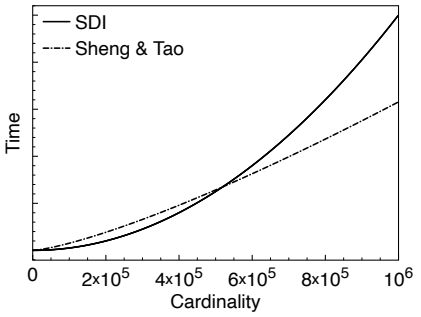
Proof. The proof is immediate if we replace M in Lemma 4 by N. □

Comparing with the best known worst-case time complexity $\mathcal{O}(N \log^{max(1,d-2)} N)$ proposed by Sheng and Tao [112], given $d > 2$, the following equation must be resolved:

$$N \log^{d-2} N > dN \log N + \frac{N^2 - dN}{4}.$$



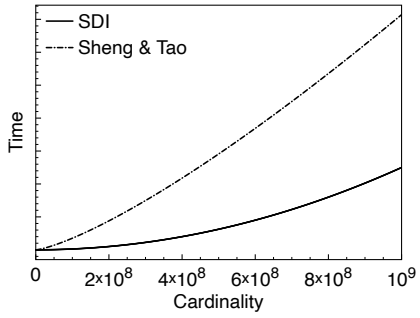
(a) $d = 6$, $N = 1 \times 10^5$.



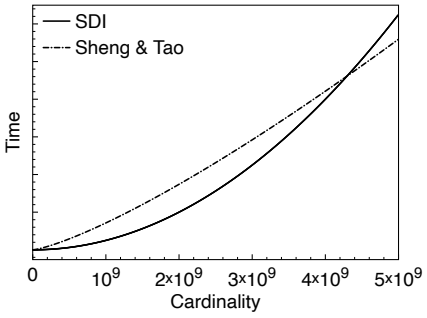
(b) $d = 6$, $N = 1 \times 10^6$.

Figure 2.3: Numerical simulation for complexity study in the worst case while $d = 6$.

The above equation belongs to transcendental equations that have no closed-form solutions. Our numerical simulation results presented in Figure 2.3 shows that while $d = 6$, `SDI` is better than the approach of Sheng and Tao for $N < 5 \times 10^5$; and while $d = 8$ shown in Figure 2.4 the compared approach beats `SDI` only if $N > 4 \times 10^9$. `SDI` performs better in high-dimensionality domains with respect to a reasonable data cardinality.



(c) $d = 8$, $N = 1 \times 10^9$.



(d) $d = 8$, $N = 5 \times 10^9$.

Figure 2.4: Numerical simulation for complexity study in the worst case while $d = 8$.

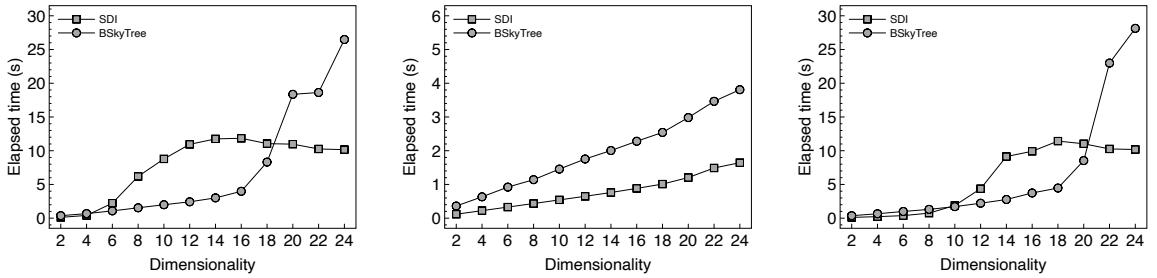
2.2.3 Experimental Evaluation

We experimentally evaluated the performance of `SDI` in comparison with `BSkyTree` implemented in SkyBench [22] on both synthetic datasets and real world datasets. We implemented `SDI`[4] in C++ standard and all executable binaries are compiled by `LLVM Clang` with `-O3` option. All experiments have been conducted on an Intel Core i5 2.8 GHz processor with 16GB 1600 MHz DDR3 RAM, running `macOS` 10.15.1 operating system. We note that all results are the average

[4] `https://github.com/skyline-sdi/sdi-bench`

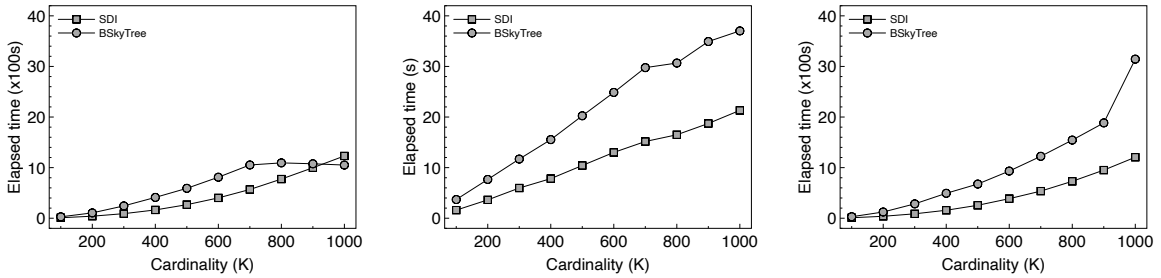
performance over 5 iterations.

First, the effects of (1) *dimensionality* and (2) *cardinality* of data have been evaluated. For (1), the cardinality is fixed to 100K and for (2), the dimensionality of data is fixed to 24. In our evaluation, only overall elapsed time, that is, the sum of data loading time, data structure construction time, and query time, has been studied.



(a) The AC dataset. (b) The CO dataset. (c) The UI dataset.

Figure 2.5: Performance evaluation on the effect of dimensionality ($N = 100$K).



(a) The AC dataset. (b) The CO dataset. (c) The UI dataset.

Figure 2.6: Performance evaluation on the effect of cardinality ($d = 24$).

`SDI` outperforms `BSkyTree` in low-dimensionality and high-dimensionality domains on AC data ($d \leq 4$ or $d > 20$) and UI ($d \leq 10$ or $d > 20$), but is less efficient than `BSkyTree` in other dimensionalities. In fact, we note that the stop line takes no advantage in AC data because of its *anti-correlated* characteristics, however, the stop line can efficiently determined with respect to the strong *correlation* in CO data, that is why `SDI` outperforms `BSkyTree`. Figure 2.6 shows the effect of cardinality on `SDI` with the highest dimensionality in our experiments. In high-dimensionality domains, `SDI` outperforms `BSkyTree` in most cases except in AC data.

Table 2.1 shows the performance comparison between `SDI` and `BSkyTree` on real datasets. `SDI` outperforms `BSkyTree` on `NBA` and `HOUSE` datasets but is much slower than `BSkyTree` on the `WEATHER` dataset because the huge number of duplicate dimension values in `WEATHER` makes

Dataset	d	N	$\lvert\mathcal{S}\rvert$	SDI	BSkyTree
HOUSE	6	127,931	5,774	306 ms	839 ms
NBA	8	17,264	1,796	45 ms	155 ms
WEATHER	15	566,268	26,713	18,680 ms	11,641 ms

Table 2.1: Performance evaluation on real datasets.

$\mathcal{O}(dK^2)$ an important factor.

Dataset	$d=2$	$d=4$	$d=6$	$d=8$	$d=10$	$d=12$
AC	36,731	5,022	2,227	240	30	27
CO	99,997	97,773	98,832	97,656	96,526	92,066
UI	99,849	88,011	63,260	30,576	12,381	8,110
Dataset	$d=14$	$d=16$	$d=18$	$d=20$	$d=22$	$d=24$
AC	13	25	0	10	0	0
CO	87,901	80,587	73,837	51,347	39,742	48,239
UI	3,794	297	933	35	3	8

Table 2.2: Pruned points by the stop line (N = 100K).

Table 2.2 lists the pruned points by the stop line in different synthetic datasets while N = 100K, that are relevant to our experimental results: the stop line does not fit AC data.

2.2.4 Conclusion

The main advantage of the SDI algorithm is that: (1) the skyline computation can be conducted on an arbitrary dimensional index; (2) any skyline point can be used to stop the computation process by outputting the complete skyline. However, an important note is that with the increasing of data cardinality, for instance to 1M or 10M points, SDI needs much time to build the dimension index, which will clearly slowdown the overall processing time. Indeed, we consider that for single-round skyline queries, to focus on total processing time is much important than to focus only on the query time without looking at data structure building time. Hence, in our particular perspective, we are interested to make SDI being a scalable framework for different skyline problems with respect to the paradigm: **building once, querying anytime and anyhow**. The main interests of this paradigm includes skyline maintenance in dynamic data (for instance, data streams or data updating) and dynamic skyline query in static data.

2.3 Maintaining Skyline in Streaming Data

The *skyline maintenance* problem in streaming data is much challenging because it requires instantaneous updates of the skyline regarding the arrival of new data and the expiration of too early data that are probably worthless and shall be discarded due to the time sensibility of data streams. In this section, we present the `RSS` (**R**ange **S**earch for **S**tream-Skyline Maintenance) algorithm for skyline maintenance in multi-dimensional data streams. Our analysis shows that the time complexity of `RSS` is bounded by a subset of the instant skyline, and our evaluation shows the efficiency of `RSS` on both of low and high dimensional data streams. We note that `RSS` is the first extension of the `SDI` algorithm presented in Section 4.2.

2.3.1 Sliding Window Based Skyline Maintenance

As the most useful technique in processing streaming data, a *sliding window* is usually considered while formulating stream based queries [102]. The model of window is generally *count-based* that covers a number of the most recent points at every time instant, or *time-based* that is bounded by a number of time units coinciding with timestamps of the stream. Figure 2.7 shows a paradigm of sliding window based skyline maintenance on the previous *price-distance* skyline illustration (Figure 2.1).

(a) Initial skyline. (b) Updated skyline.

Figure 2.7: A skyline maintenance example.

Let us consider a count-based window $W = 5$ and the arrival order of hotel points from a to f, then initially there are 4 skyline points $\{a, c, d, e\}$ as shown in Figure 2.7(a). At the instant while f arrives, the earliest point a must be discarded in order to keep the size of window. As a consequence, since a is the only point that dominates b, b becomes a skyline point when a is discarded; further, the incoming point f dominates current skyline points d and c, so d and c

must be removed from current skyline, and finally the updated skyline is therefore $\{b, c, f\}$, as shown in Figure 2.7(b).

In fact, dynamic updates of data make skyline maintenance difficult over data streams: (1) *an expired skyline point may release dominated points so that they become skyline points*; (2) *an incoming point dominates existing skyline points so that the latter shall be removed from the skyline.* Most of the existing skyline algorithms designed for static data do not fit such requirements imposed by streaming data. For instance, regarding each update of skyline invoked by incoming/expired points with respect to a sliding window, the straightforward algorithm `BNL` requires a full scan of the window, sorting-based algorithms require re-sorting all points in the window, and partitioning-based algorithms require rebuilding the partitioning structure. Hence, the dynamic update of skyline and the efficient data structures are two important issues of skyline maintenance on data streams [72, 73, 85, 120].

2.3.2 Dynamic Dimension Indexing

Our main contributions include:

- We construct a linked sorting structure for maintaining a dynamic dimension index that serve the sliding window.
- We show that with the proposed dynamic dimension indexing method, the dominance tests involved by the above operations (1) and (2) are limited to optimal subsets of current skyline points in the sliding window.
- We evaluate our proposed algorithm with both count-based and time-based sliding window, on both synthetic and real datasets, and show its efficiency on both low and high dimensional data streams.

Definition 12 (Dynamic Dimension Index)**.** *Let $\mathcal{D}$ be a database of d dimensions. The dimension index $\mathcal{I}$ of $\mathcal{D}$ is the ensemble of d ordered lists of index entries, where each index entry corresponds to an individual point, which contains the dimension value, a dimension link that points to the entry of the same point in the next dimension, and a header link that points to the point header. Each list $I_i \in \mathcal{I}$ is a sub-index of the dimension i, $1 \leq i \leq d$, in which the index entries are sorted by the dimension value on the preference order $\prec$.* ■

Figure 2.8 illustrates the dimension index data structure by following the links on all index entries of a point t_k, where the node h_k depicts the point header of t_k and the nodes $v_k^1, v_k^2, \ldots, v_k^d$ depict the index entries of t on dimensions $1, 2, \ldots, k$. Since all entries in a sub-index is sorted

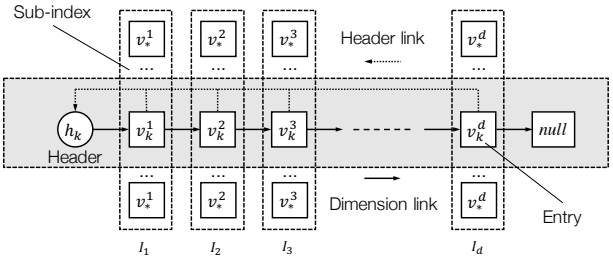


Figure 2.8: Dimension index data structure.

by dimension values, we note other point IDs as $*$ in our illustration, for instance, v_*^3 denotes the dimension value of a point $*$ on the dimension 3.

This design of linked entries allows fast point access from any dimension with a limited memory usage. For instance, let us consider 64-bit operating systems and 64-bit double-precision floating-point values, a d-dimensional point requires $24 \times d$ Bytes to store all index entries, including values, header links, and dimension links; a header stores a point ID as a 64-bit integer number, a 64-bit dominance list pointer (see Section 4, Figure 2.10), and a skyline flag, that requires 20 Bytes. Therefore, to store a point in our dimension index, totally $24 \times d + 20$ Bytes are required. Note the the space complexity of B-tree is $\mathcal{O}(N)$, where N is the number of nodes. If 2 64-bit pointers per node are required to maintain the tree, then totally $16 \times N \times (24 \times d + 20)$ Bytes are required by our proposed dynamic dimension index, that is, 100 MB of memory space is enough to handle a 10K window of 20-dimensional streaming data.

Based on Lemma 1, we present a dynamic dimension index updating method that allows efficient skyline maintenance. We illustrate our method with the example shown in Figure 2.9, where we assume that all values on any dimension are unique. Without loosing the generality, the symbol $\cdots$ signifies other irrelevant index entries related to this example.

Let t be an incoming point, we first locate the index entries of t on each sub-index with respect to the preference order $\prec$ (as the gray entries). Then we find a dimension *low* that contains the minimum number of skyline points s such that $s[lower] \prec t[lower]$, which we call the *lower-bounded* dimension of t where a *lower-bounded zone* can be detected by bounding the lower side from the top of the sub-index. We call the set of skyline points contained in the lower-bounded zone the *lower-bounded skyline*, to which we apply Lemma 1 to test whether t is a skyline point. For instance, in our example, the lower-bounded skyline is located in I_3, comparing with which t and c is incomparable so t is a skyline point. The incoming point t may dominate existing skyline points, Lemma 1 can also applied to filter such existing skyline points. We find a dimension *up* that contains the minimum number of skyline points u such that $t[upper] \prec u[upper]$, and we

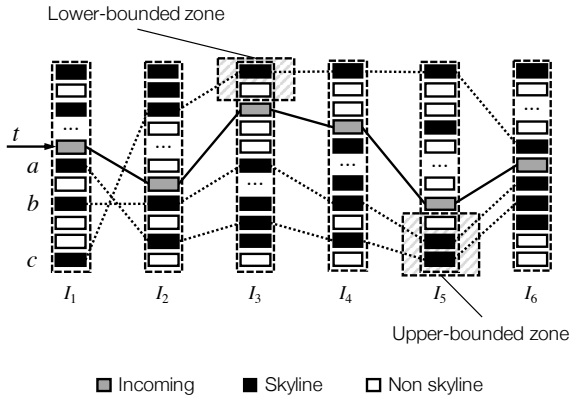


Figure 2.9: A example of dynamic dimension index.

call such a dimension *up* the *upper-bounded* dimension of t where a *upper-bounded zone* can be detected as well as the lower-bounded zone. For each point u contained in the *upper-bounded skyline*, if $t \prec u$, then u can be removed from the skyline. In our example, current skyline points a and b are dominated by t so they will be no longer in the skyline. Otherwise, if t is not a skyline point, no upper-bounded skyline detection is necessary.

While deleting a skyline point t, then for each non skyline point x contained in the lower-bounded zone of t such that $t \prec x$, the lower-bounded zone must be detected in order to test whether x is dominated by its lower-bounded skyline: if not, x shall be a new skyline point. Note that no dominance test is required while deleting a non skyline point.

In the algorithm `RSS`, the name of *Range Search* is based on the bounded search range of dominance tests. The update of skyline is invoked by each incoming point and can be described as the following:

1. When the window is not yet filled, perform incremental skyline computation with respect to incoming points.

2. When the window is filled, an incoming point invokes the collection of expired points with respect to the window mode: drop a single point for count-based window, or drop a set of points for time-based window.

3. For each expired skyline point, check whether any dominated points can be released to be new skyline points.

4. Discard expired points from the window.

5. Add the incoming point to the window by performing incremental skyline computation.

Algorithm 4: `RSS` (Range Search for Stream)

Input: Data stream $\mathcal{D}$, Window W

Output: Instant update of skyline $\mathcal{S}$

```
1  Dimension Index I = ∅
2  Skyline Index S = ∅
3  while t ← D do
4      U ← ExpiredPoints(I, W)
5      foreach u ∈ U do
6          if u ∈ S then
7              X ← DominatedPoints(I, u)
8              Disable u in I
9              foreach x ∈ X do
10                 if RangeSearch(I, x) then
11                     S ← S ∪ {x}
12             S ← S \ {u}
13         Remove u from I
14     if RangeSearch(I, t) then
15         R ← DominatedPoints(I, t)
16         foreach r ∈ R do
17             Remove r from I
18         S ← S ∪ {t}
19     Insert t to I
```

Algorithm 4 outlines `RSS`, which takes a data stream $\mathcal{D}$ and a window W as input, and updates the skyline index $\mathcal{S}$ while sliding the window W over $\mathcal{D}$ by accepting each incoming point t from the data stream.

First, `RSS` deletes expired points from the dimension index $\mathcal{I}$ (lines 5–17). `RSS` fetches the set U of all expired points with respect to the count-based or time-based window W (line 4). For each expired skyline point $u \in U$, `RSS` finds by *DominatedPoints* the set X of all points directly dominated by u (line 7) then disables u in $\mathcal{I}$ for not affecting following dominance tests. For each point $x \in X$, `RSS` calls *RangeSearch* (Algorithm 5) to determine new skyline points (lines 9–13). To finish point deletion, `RSS` removes u from the skyline index $\mathcal{S}$ if $u \in \mathcal{S}$ (line

14), and definitively removes u from $\mathcal{I}$ (line 16). Then, `RSS` inserts the incoming point into $\mathcal{I}$ (lines 18–25). `RSS` calls *RangeSearch* to determine whether t is a skyline point (line 18). If t is a skyline point, `RSS` calls *DominatedPoints* that returns the set R of the points directly dominated by t and removes each point $r \in R$ (lines 19–22). Finally, `RSS` adds t to $\mathcal{S}$ (line 23), and commit the insertion of t to $\mathcal{I}$ (line 25). Notice that if no expired point can be fetched, `RSS` directly proceeds point insertion.

In our design of dimension index (Figure 2.8), each sub-index is a B-tree of index entries with respect to the preference order $\prec$ on dimension values. Three additional data structures, including a *skyline index* that contains pointers to skyline point headers, a *header list* that contains all headers, and a *dominance index* that contains pointers to all directly dominated points, are used to maintain the dimension index. While inserting point to the dimension index, a header is first created to identify the point and appended to the header list with respect to the incoming order; then, a linked list of per-dimension index entries are created from the point with respect to dimension links, and each index entry corresponding to a dimension i will be inserted into the sub-index I_i.

The *RangeSearch* method is listed in Algorithm 5, which follows the dynamic dimension indexing described in Section 3 by determining lower-bounded and upper-bounded skylines for the point t within the dimension index $\mathcal{I}$. Given a point t, *RangeSearch* first calculates the lower-bounded dimension for t in order to perform dominance tests (line 2). According to Theorem 1, *RangeSearch* tests whether t is a local skyline in the set B of all points having the same dimension value (line 3) by the embedded `BNL` algorithm[5], which returns *true* if t is a local skyline; otherwise *RangeSearch* stops (lines 4–6). If t is a local skyline, *RangeSearch* further retrieves the lower-bounded skyline S for t (line 7), then tests t is a *true* skyline against S: if not, *RangeSearch* stops by returning *false* (lines 8–12). If t is found being a skyline point, *RangeSearch* calculates the upper-bounded dimension for t in order to eliminate skyline points eventually dominated by t (line 13). At this step, repeated dimension values shall also be taken into account: if t dominates any skyline point b having the same dimension values, then b must be removed from the skyline index (lines 14–20). *RangeSearch* retrieves the upper-bounded skyline S for t (line 21) removes all skyline points dominated by t (lines 22–27). *RangeSearch* returns *true* if t is a skyline point.

The update of dominance index is performed by the *UpdateDominanceList* call in *RangeSearch* while the skyline status of any point is modified, to make *DominatedPoints* efficient. A call

[5]Any skyline algorithm can be applied to compute local skylines, `BNL` is the simplest one with respect to a small number of repeated dimension values.

Algorithm 5: `Range Search`

Input: Dimension index $\mathcal{I}$, point t

Output: **true** if t is a skyline point

```
1  Skyline Index S
2  lower ← LowerBoundedDimension(I, t)
3  B ← GetBlock(t, I_lower)
4  if not BNL(B, t) then
5      return false
6  S ← LowerBoundedSkyline(I_lower, t)
7  foreach s ∈ S do
8      if s ≺ t then
9          return false
10 upper ← UpperBoundedDimension(I, t)
11 B ← GetBlock(t, I_upper)
12 foreach b ∈ B do
13     if b ∈ S and t ≺ b then
14         UpdateDominanceList(t, b)
15         S = S \ {b}
16 S ← UpperBoundedSkyline(I_upper, t)
17 foreach s ∈ S do
18     if t ≺ s then
19         UpdateDominanceList(t, s)
20         S = S \ {s}
21 return true
```

like $UpdateDominanceList(t, b)$ joins the dominance index of b to which of t. Figure 2.10 shows how dominance indexes are updated while an incoming point dominates existing skyline points, where a count-based window $W = 15$ is concerned. The update of dominance indexes can be illustrated by the following steps (let the gray band depict the skyline index and assume that the incoming point 33 is a skyline point): (a) the incoming point 33 makes the current earliest point 18 expired; (b) while removing the expired point 18, assume that point 25 becomes skyline point, and assume that the incoming point 33 dominates the skyline point 21; (c) point 21 is appended

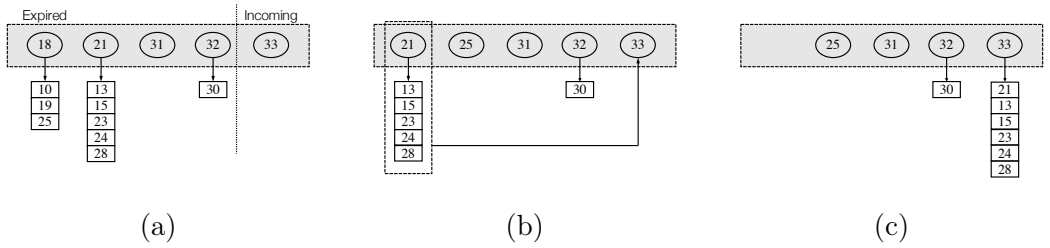


Figure 2.10: Dynamic update of dominance indexes.

to the point 33 and removed from the skyline index and its dominance entries are moved to point 33.

In order to determine the lower-bounded and upper-bounded dimensions for a point t, we use the following estimation formula instead of counting the exact sizes of lower-bounded and upper-bounded skylines.

$$I_{lower} = \arg\min_{I_i}(|\frac{v_t^i - \min(I_i)}{\max(I_i) - \min(I_i)}|) \tag{2.1}$$

$$I_{upper} = \arg\max_{I_i}(|\frac{v_t^i - \min(I_i)}{\max(I_i) - \min(I_i)}|) \tag{2.2}$$

The above approximate estimation of two bounded dimensions requires $\mathcal{O}(d)$ time, which is a trade-off between selecting the best dimension with the smallest bounded skyline and calculating in the least time.

2.3.3 Theoretical Analysis

Within a sliding windows W over d-dimensional streaming data, let M be the size of skyline and assume that dominance test requires $\mathcal{O}(d)$ time, we have the following analysis.

Lemma 6. *In the worst case,* `RSS` *inserts an incoming skyline point t in time*

$$\mathcal{O}\Big(d\log|W| + 2(d+M) + \frac{M}{d}(d + \frac{(|W|-M)(d+M)}{d})\Big).$$

Proof. The terms in the complexity expression is describes as following. First, to insert d index entries to d B-tree based sub-indexes, $\mathcal{O}(d\log|W|)$ time is required. Then, if t is a skyline point, $\mathcal{O}(2d)$ time is required to find the lower-bounded and upper-bounded dimensions, where the size of lower-bounded and upper-bounded skylines is both M/d. According to Lemma 1, M/d dominance tests are required by t with each bounded skyline, thus the total time is bounded in $\mathcal{O}(2d+2M)$. In the worst case, t dominates all M/d points in the upper-bounded skyline, then for

each upper-bounded skyline point, $\mathcal{O}(d)$ time is required to find the upper-bounded dimension, where $(|W| - M)/d$ non skyline points are dominated by each upper-bounded skyline point. Finally, for each of such non skyline points, $\mathcal{O}(d)$ time is required to find the lower-bounded dimension and the size of lower-bounded is M/d in the worst case, that is, $\mathcal{O}(d + M)$ time is required to test whether it will be a skyline point. □

Lemma 7. *In the worst case,* `RSS` *deletes an expired skyline point t in time*

$$\mathcal{O}\Big(d + \frac{(|W| - M)(d + M)}{d} + d \log |W|\Big).$$

Proof. To delete a skyline point, $\mathcal{O}(d)$ time is required to find the upper-bounded dimension. Then, as shown in the proof of Theorem 6, $(|W| - M)/d$ non skyline points in the upper-bounded zone shall be tested in the worst case, each test requires $\mathcal{O}(d + M)$ time. Finally, $\mathcal{O}(d \log |W|)$ time is require to remove d index entries from the B-trees. □

With the above analysis, it is clear that `RSS` requires $\mathcal{O}(d \log |W| + d + M)$ time to insert a non skyline incoming point and $\mathcal{O}(d \log |W|)$ time to delete a non skyline expired point. Furthermore, the higher the dimensionality d of data, the smaller the factor $(|W| - M)$. Indeed, our performance evaluation shown in Section 5 confirms the efficiency of `RSS` in high dimensional data streams.

2.3.4 Experimental Evaluation

We implemented `RSS` in C++, and all executable binaries were compiled by LLVM Clang with `-O3` option[6]. All experiments were conducted on an Intel Core i5 3.1 GHz processor with 16GB DDR3 RAM running macOS 10.14.5 operating system. Furthermore, all results reported are the average performance over 5 iterations except per incoming point processing time, based on the mentioned synthetic and real-world datasets in Section 2.1.

The dimensionality of synthetic datasets varies from 2 to 24 dimensions. Different window sizes are considered, $W \in \{1\text{K}, 2\text{K}, 4\text{K}, 8\text{K}, 16\text{K}, 32\text{K}\}$ points for count-based window and $W \in \{10, 20, 40, 80, 160, 320\}$ seconds for time-based window. Particularly for time-based window, the arrival speed of data stream is uniformly randomized between 100 and 1000 points per second. As a reference, Table 2.3 lists the mean per window skyline ratio (%) with respect to the dimensionality of synthetic data. All results are reported after the sliding window is filled, and subsequently, fixed window size and dimensionality is chosen to analyze the detail process of one incoming point.

[6] `https://github.com/skyline-sdi/sdi-rss`

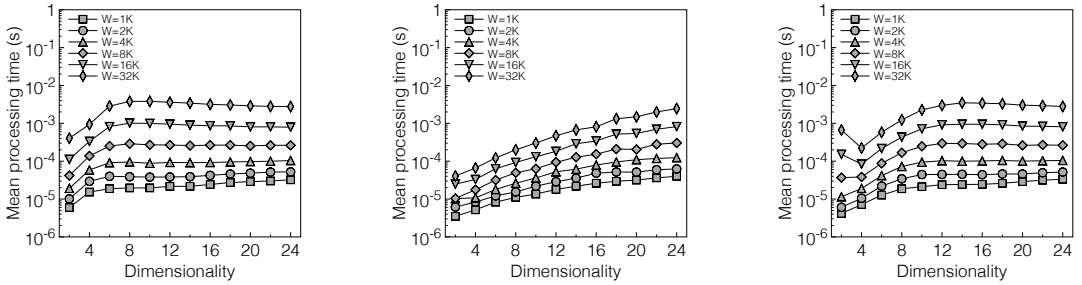
d	02	04	06	08	10	12	14	16	18	20	22	24
AC	1.10	23.46	61.00	83.35	93.45	97.21	98.90	99.48	99.73	99.89	99.95	99.98
CO	0.09	0.20	0.90	1.83	2.68	7.63	11.60	18.47	27.15	31.92	40.29	48.73
UI	0.27	2.84	13.37	33.78	56.57	75.43	89.10	95.07	98.40	99.51	99.85	99.94

(a) Count-based window.

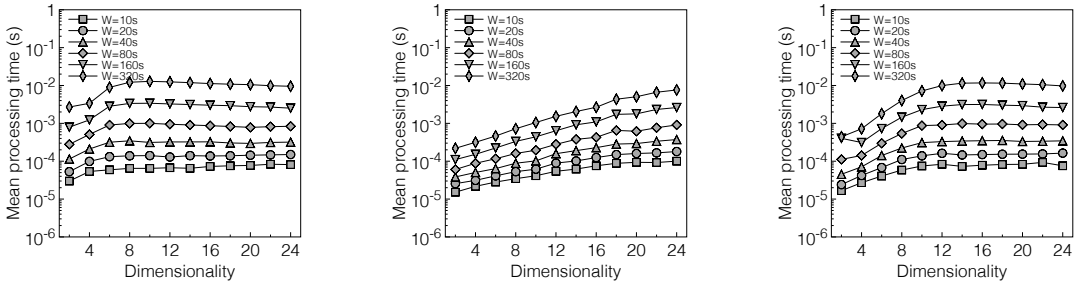
d	02	04	06	08	10	12	14	16	18	20	22	24
AC	0.82	19.97	56.87	80.65	92.00	96.46	98.56	99.32	99.64	99.85	99.93	99.97
CO	0.07	0.15	0.69	1.46	2.12	6.54	10.09	15.60	24.40	28.15	37.31	44.89
UI	0.19	2.33	11.46	30.45	52.72	71.95	87.42	94.20	97.94	99.35	99.79	99.92

(b) Time-based window.

Table 2.3: Mean per window skyline ratio (%) w.r.t. dimensionality.



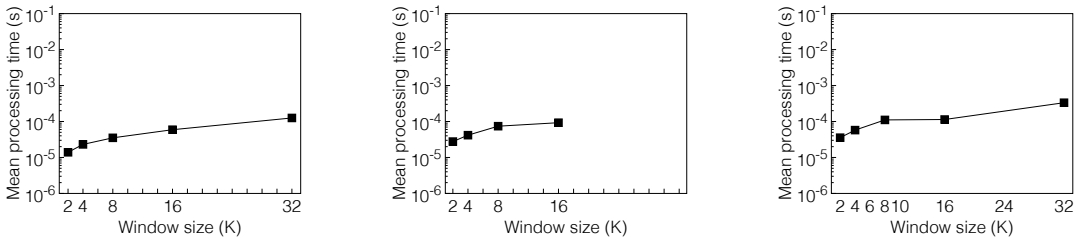
(a) AC data. (b) CO data. (c) UI data.



(d) AC data. (e) CO data. (f) UI data.

Figure 2.11: Mean skyline update time for count-based window (a, b, c) and time-based window (d, e, f) w.r.t dimensionality on synthetic data.

Figure 2.11 shows the mean processing time for synthetic data by varying the dimensionality from $d = 2$ to 24 stepped by 2. In Figure 2.11(c), there are irregular points when $d = 2$ and $W \geq$ 8K that the skyline updates spend more time but size is quite small. We investigate the detailed process when $d = 2$ and find out that the first expired tuple is a skyline in all



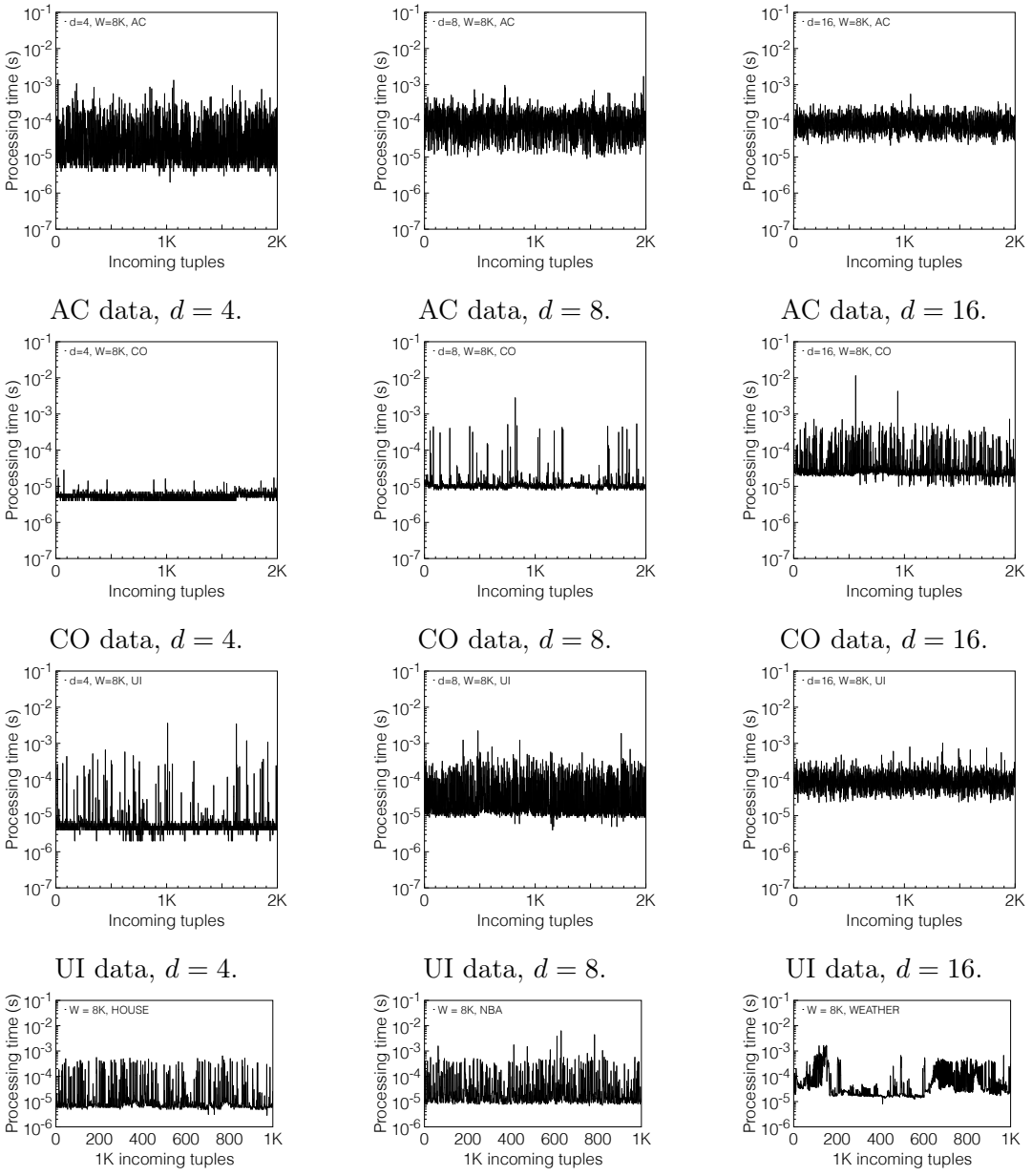
(a) HOUSE dataset. (b) NBA dataset. (c) WEATHER dataset.

Figure 2.12: Mean skyline update time for count-based window on real-world data.

cases. Theoretically, the dataset is uniformly distributed and independent on each dimensions, in addition the skyline size is very small in comparing with the window size, so the first expired tuple is likely to dominate approximately half-window (W/d) tuples. For $W = 32$K, the experiment shows that the first expired tuple dominates more than 15K tuples in the window and it spends more than 1s to check all its dominated non skyline tuples. Same situation happens to $d = 2$ and $W = 16$K as well as $d = 2$ and $W = 8$K, of which the first expired tuple dominates respectively about 8K and 4K tuples, and the processing time is respectively about 0.2s and 0.04s. Figure 2.11(f) also shows exceptional points as which is resulted from the same reason. Therefore, the mean processing time is affected by deleting the first expired tuple. Notice that except the first expired tuple, the processing time per new tuple is 10^{-5} on average. A common observation is that the performance reaches stable or keeps in one magnitude regarding the incremental of the dimensionality. For AC data, the performance is stable while $d \geq 8$ with both window mode, and $d \geq 12$ for UI data; however, the processing time keeps increasing for CO data. This observation confirms our theoretical analysis in the previous section.

Figure 2.12 shows the mean per window skyline update time on real-world data (the NBA dataset is too small), where RSS shows the same performance as well as on synthetic data. Due to the space limitation, we do not include the mean skyline update time for time-based window on real-world data, however the curves are very similar to which shown in Figure 2.12.

To investigate the per incoming tuple processing time, we fix the dimension to $d \in \{4, 8, 16\}$ and window size to $W = 8$K for count-based window (Figure 2.13) and time-based window (Figure 2.14) on synthetic data and real-world data (due to the space limitation, we do not include the time-based window processing time on real-world data, which is very similar to Figure 2.13). Although it reveals that the processing time mainly varying between 10^{-4} ~10^{-5}s, when one expiration is skyline, it has to check a large number of non skyline tuples which results in high spike, that explains also why the processing time on CO data is much stable (especially $d = 4$, for instance) than other data types. We also note that the irregular result on WEATHER



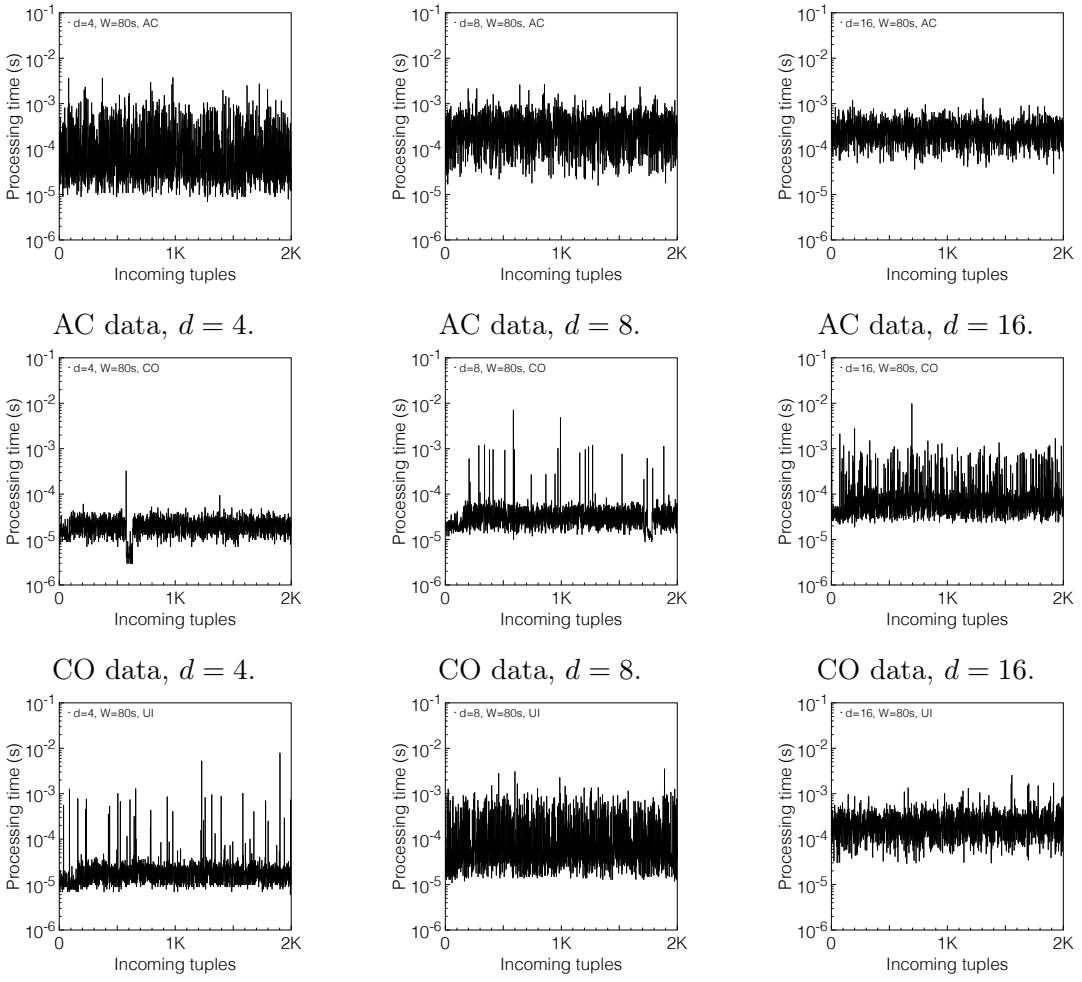
AC data, $d = 4$. AC data, $d = 8$. AC data, $d = 16$.

CO data, $d = 4$. CO data, $d = 8$. CO data, $d = 16$.

UI data, $d = 4$. UI data, $d = 8$. UI data, $d = 16$.

`HOUSE` data, $W = 8$K. `NBA` data, $W = 8$K. `WEATHER` data, $W = 8$K.

Figure 2.13: Incoming tuple processing time for count-based window on synthetic and real-world data.



AC data, $d = 4$. AC data, $d = 8$. AC data, $d = 16$.

CO data, $d = 4$. CO data, $d = 8$. CO data, $d = 16$.

UI data, $d = 4$. UI data, $d = 8$. UI data, $d = 16$.

Figure 2.14: Incoming tuple processing time for time-based window on synthetic data.

data (Figure 2.13) is raised by the huge number of repeated dimension values.

2.3.5 Conclusion

Our experimental results, which are detailed in the full paper [76], shows that the mean processing time is affected by deleting the first expired point if it is a skyline point, then the processing time per new point is 10^{-4} to 10^{-5} seconds in average. Although it reveals that the processing time mainly varying between 10^{-4}s to 10^{-5}s, when one expiration point is a skyline point, it has to check a large number of non skyline points which results in high spike. Another common observation is that the performance reaches stable or keeps in one magnitude regarding the incremental of the dimensionality. Our evaluation confirms our previous theoretical analysis. As a possible future work, we are interested in the adaption of `RSS` and `SDI` to distributed data streams and to column-based data streams.

2.4 Boosting Skyline Computation by Subset Queries

In this section, we present a subset-based approach that allows to integrate subspace-based incomparability to existing sorting-based skyline algorithms and can therefore significantly reduce the total number of dominance tests in large multidimensional datasets. Our theoretical and experimental studies show that the proposed subset approach boosts existing sorting-based skyline algorithms and makes them comparable to the state-of-the-art algorithms and even faster with uniform independent data.

2.4.1 Problem Statement

Godfrey et al. [46, 47] and Sheng and Tao [112] theoretically analyzed the time complexity of existing skyline algorithms and concluded that in the worst case, sorting-based algorithms finish in $\mathcal{O}(dN^2)$ time, however partitioning-based algorithms can finish in $\mathcal{O}(N \log^{d-2} N)$ time for any dimensionality $d > 2$. Godfrey et al. [46, 47] also gave a time complexity analysis for the average case under the *uniform independence* and *component independence* conditions. Essentially, the efficiency of a skyline algorithm heavily relies on how dominance tests are reduced. As the state-of-the-art skyline algorithms, `BSkyTree-S` and `BSkyTree-P` use a pivot point selection schema to map data points to incomparable regions, each can be considered as an optimized algorithm of sorting-based and partitioning-based categories.

Basically, given a d-dimensional space, a *subspace* [103] is a subset of all d dimensions, which is extended studied with several skyline problems such as *subspace skyline* [121, 104, 66] and

skycube [104, 11]. Let $\mathcal{P}$ be the dataset, $p \in \mathcal{P}$ be a skyline point, and $q \in \mathcal{P}$ be any point such that $p \neq q$, then, if p does not dominate q, there must exist a subspace where in each dimension the value of q is better than the value of p, that is, q dominates p in this subspace. We call such a subspace a *dominating subspace*. If all points dominated by the skyline point p have been pruned, each point that remains in the dataset must possess a dominating subspace where it dominates p. Indeed, the above skyline point p, also called a *pivot point*, is used by the `BSkyTree-S` and `BSkyTree-P` algorithms to partition points into incomparable regions in order to reduce the dominance tests.

We present a novel subspace-based skyline indexing approach to manage the incomparability between testing points and skyline points. Instead of partitioning the dataset, our method indexes the skyline points by dominating subspaces to reduce the dominance tests. We first show that the dominating subspace of a point can be merged with respect to multiple *pivot skyline points*, called *maximum dominating subspace* with respect to a given number of pivot points, then, we show that if a point q_1 dominates a point q_2, q_1's maximum dominating subspaces with respect to each pivot point must be a superset of q_2's maximum dominating subspaces. Based on the properties presented in this section, the sketch of the application of our method can be described as following:

1. Find multiple pivot skyline points to generate the maximum dominating subspace for each non-pruned point.
2. Run a skyline algorithm with the following new actions:
   (a) Once a skyline point is determined, put it to the proposed skyline index structure with respect to its maximum dominating subspace.
   (b) While testing a point with the current skyline, get only the set of comparable points from the skyline index structure with respect to the maximum dominating subspace.
3. Return the skyline.

Therefore, our proposed method does not concern concrete skyline computation algorithms because it is designed as a component like a container that allows to store (as *put* function) the skyline points and to retrieve (as a *get* function) a minimum number of skyline points to compare with a testing point. The paradigm of our method fits best sorting-based skyline algorithms that can progressively output the skyline points, however partitioning-based skyline algorithms cannot benefit much from our method because the data have already been partitioned and the dominance tests are limited in partitions, the double partitioning of data and skyline brings additional costs.

Therefore, the advantage of our method is to boost existing skyline algorithms. Several sorting-based skyline algorithms such as `SFS`, `SaLSa`, and `SDI` do not depend on any particular data structures (for instance, the lattice and SkyTree required by `BSkyTree-S` and `BSkyTree-P`, the ZB-tree required by `ZINC`, the R-tree required by `BBS`, and the B^+-tree required by `Index`, etc.), which can be easily implemented in any programming languages including `Python` and `PHP`, so to boost such algorithms has realistic interests to data industries.

The main result presented in this section is that the skyline indexing can be efficiently resolved by a **subset query**. The *subset query problem* is defined as: given a set $\mathcal{X}$ of distinct subsets of a universe D, for any set $Q \subseteq D$, return the set $\{Q' \in \mathcal{X} \mid Q \subset Q'\}$ that contains all supersets of Q. The set Q is called the *query set*. Indeed, given a testing point, its maximum dominating subspace can be considered as a query set, the task is to return all skyline points of which the maximum dominating subspaces are the supersets of the query set, hence, the required dominance tests can be limited in returned skyline points. In order to make our method efficient, we reversed the subset query problem, that is, to return the set $\{Q' \in \mathcal{X} \mid Q' \subset Q\}$. The subset query data structure and algorithms proposed in this section is hash map based (supported by the most of programming languages), which can add a skyline point in linear time with respect to the dimensionality d and can retrieve a set of skyline points with respect to a given subspace in $O((\frac{d}{2})^2)$ time in the average case for any $d > 2$. Particularly, in the case of $d = 2$, subset query is a binary problem so the usefulness of our proposed method is very limited.

Definition 13 (Subspace)**.** *Given a d-dimensional dataset $\mathcal{P}$, the set $D = \{1, 2, \ldots, d\}$ is the* space *of $\mathcal{P}$. Any subset $D' \subseteq D$ is a* subspace *of $\mathcal{P}$.* ■

Definition 14 (Dominating Subspace)**.** *Let p and q be two points. Let $D_{p \prec q}$ denote the subspace such that $\forall i \in D_{p \prec q} \Rightarrow p[i] < q[i]$ and $\forall i \notin D_{p \prec q} \Rightarrow q[i] \leq p[i]$, then $D_{p \prec q}$ is the* dominating subspace *of p with respect to q.* ■

According to Definition 14, given a dataset $\mathcal{P}$, let D be the space of $\mathcal{P}$ and $p, q \in \mathcal{P}$ be two points, we have:

- $D_{p \prec q} = \emptyset \Rightarrow q \prec p$ or $p = q$;
- $D_{p \prec q} = D \Rightarrow p \prec q$.

Lemma 8. *Given a dataset $\mathcal{P}$, let $p \in \mathcal{P}$ be a skyline point and $q_1, q_2 \in \mathcal{P}$, $q_1, q_2 \neq p$ be two arbitrary points such that $p \not\prec q_1$, $p \not\prec q_2$, and $q_1 \neq q_2$. $q_1 \nsim q_2$ if $D_{q_1 \prec p} \not\subseteq D_{q_2 \prec p}$ and $D_{q_2 \prec p} \not\subseteq D_{q_1 \prec p}$.* ■

Proof. p is a skyline point, so $D_{q_1 \prec p} \neq D$ and $D_{q_2 \prec p} \neq D$, that is, $D_{p \prec q_1} \neq D$ and $D_{p \prec q_2} \neq D$, which impose that $p \not\prec q_1$ and $p \not\prec q_2$. We have that $p \not\prec q_1 \iff \exists i, q_1[i] < p[i]$ and $p \not\prec q_2 \iff \exists i, q_2[i] < p[i]$, hence, $D_{q_1 \prec p} \not\subseteq D_{q_2 \prec p}$ implies that there is at least one dimension i such that $q_1[i] < p[i] \leq q_2[i]$ and $D_{q_2 \prec p} \not\subseteq D_{q_1 \prec p}$ implies that there is at least one dimension j such that $q_2[j] < p[j] \leq q_1[j]$. Thus, $q_1 \not\prec q_2$ and $q_2 \not\prec q_1$, that is, $q_1 \not\sim q_2$. □

More simply, Lemma 8 can be rewritten as:

$$|D_{q_1 \prec p} \cap D_{q_2 \prec p}| < min(|D_{q_1 \prec p}|, |D_{q_2 \prec p}|) \Rightarrow q_1 \not\sim q_2.$$

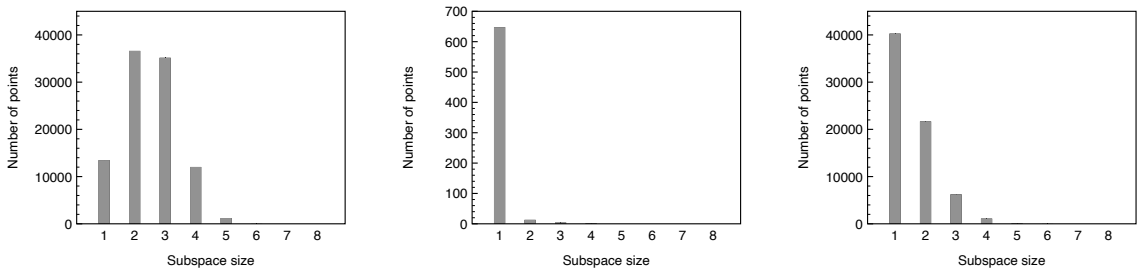
Lemma 9. *Given a dataset $\mathcal{P}$, let $p \in \mathcal{P}$ be a skyline point and $q_1, q_2 \in \mathcal{P}$, $q_1, q_2 \neq p$ be two arbitrary points such that $p \not\prec q_1$, $p \not\prec q_2$, and $q_1 \neq q_2$. $D_{q_1 \prec p} \not\supseteq D_{q_2 \prec p} \Rightarrow q_1 \not\prec q_2$.* ■

Proof. We have the same context as Lemma 8. If $D_{q_2 \prec p} \not\supseteq D_{q_1 \prec p}$, according to the proof of Lemma 8, there exists at least on dimension i such that $i \in D_{q_1 \prec p}$ and $i \notin D_{q_2 \prec p}$, that is, $q_2[i] < p[i] \leq q_1[i]$. Thus, $D_{q_1 \prec p} \not\supseteq D_{q_2 \prec p} \Rightarrow q_1 \not\prec q_2$. $D_{q_1 \prec p} \supseteq D_{q_2 \prec p}$ is a necessity of $q_1 \prec q_2$. □

Lemma 9 shows an inevitable constraint to partition points with respect to Lemma 8: if a point p is a skyline point in the set of points determined by the subspace D_x, then p must be compared with all skyline points in the set of points determined by any subspace $D_y \supset D_x$. Since a d-dimensional space contains $2^d - 2$ subspaces without $\emptyset$ neither the full space in our context, obviously, the dominance tests can be effectively reduced if all points can be distributed to as many incomparable subspaces as possible.

According to Lemma 8, given a dataset $\mathcal{P}$ in space D, if a skyline point $p \in \mathcal{P}$ is compared with each other point $q \in \mathcal{P}$, then every non-pruned point q can be attributed a dominating subspace $D_{q \prec p}$, where $D_{q \prec p} \neq \emptyset$ and $D_{q \prec p} \neq D$.

Figure 2.15 shows the distribution of non-pruned points in AC, CO, and UI synthetic datasets with 100K points of 8 dimensions, where the pivot point is the skyline point with the minimal Euclidean distance to the zero point. Due to page length limit, it is difficult to list the number of points for all $2^8 - 2$ subspaces, we show in Figure 2.15 the number of points with respect to subspace size. We see that the distribution of points is unbalanced, most of them are in small-size zones, far away from the 2^d level. Although recursive calls can be applied to each subspace to find more incomparable subspaces with respect to Lemma 8, Lemma 9 limits the immediate output of skyline points. Due to the number of $2^d - 2$ subspaces, we use the size (number of dimensions) of dominating subspaces as the criterion to evaluate the distribution of points. We see from Figure 2.15 that the distribution of points is unbalanced, most of them are in small-size



(a) 8-D 100K AC dataset. (b) 8-D 100K CO dataset. (c) 8-D 100K UI dataset.

Figure 2.15: Distribution of points with respect to subspace size where the pivot point is the skyline point with the minimal Euclidean distance to the zero point.

zones, far away from the 2^d level. Although recursive calls can be applied to each subspace to find more incomparable subspaces with respect to Lemma 8, Lemma 9 limits the immediate output of skyline points.

2.4.2 Subspace Union

We propose a subspace union method to distribute points to as many subspaces as possible , where the term *as many as possible* is controlled by a given threshold that finally affects the number of pivots points. Given a point, the dominating subspace can be merged from multiple pivot points, that is, a set of skyline points since all pivot points are skyline points, and we call such a merged subspace a *maximum dominating subspace*, where the term *maximum* means the maximum number of dimensions where the given point dominates the pivot points.

Definition 15 (Maximum Dominating Subspace)**.** *Let S be a set of skyline points in a dataset $\mathcal{P}$ of space D. For a point $q \in \mathcal{P}$, the union of dominating subspaces $D_{q\prec S} = \bigcup_{p\in S} D_{q\prec p}$, where $D_{q\prec S} \subseteq D$, is the* maximum dominating subspace *of q.* ■

With the above definition, we have the following extensions of Lemma 8 and Lemma 9, which are the bases of our results.

Lemma 10. *Given a dataset $\mathcal{P}$, let S be a set of skyline points of $\mathcal{P}$ and $q_1, q_2 \in \mathcal{P}$ be two arbitrary points such that $q_1, q_2 \notin S$, $S \not\prec q_1$, $S \not\prec q_2$, and $q_1 \neq q_2$. $q_1 \nsim q_2$ if $D_{q_1\prec S} \not\subseteq D_{q_2\prec S}$ and $D_{q_2\prec S} \not\subseteq D_{q_1\prec S}$.* ■

Proof. The proof is as the proof of Lemma 8. If $D_{q_1\prec S} \not\subseteq D_{q_2\prec S}$ and $D_{q_2\prec S} \not\subseteq D_{q_1\prec S}$, there must be at least one skyline point $p \in S$ on at least one dimension i where $q_1[i] < p[i] \leq q_2[i]$ or $q_2[i] < p[i] \leq q_1[i]$. Thus, $q_1 \nsim q_2$. □

Lemma 11. *Given a dataset $\mathcal{P}$, let S be a set of skyline points of $\mathcal{P}$ and $q_1, q_2 \in \mathcal{P}$ be two arbitrary points such that $q_1, q_2 \notin S$, $S \not\prec q_1$, $S \not\prec q_2$, and $q_1 \neq q_2$. $D_{q_1 \prec S} \not\supseteq D_{q_2 \prec S} \Rightarrow q_1 \not\prec q_2$.* ■

Proof. If $q_1 \prec q_2$, then $\forall i \in D, q_1[i] \leq q_2[i]$. If $D_{q_1 \prec S} \not\supseteq D_{q_2 \prec S}$, then there exists at least one skyline point $p \in S$ on at least one dimension i where $q_2[i] < p[i] \leq q_1[i]$. Thus, $q_1 \not\prec q_2$. □

In practice, it is difficult to determine the number of pivot skyline points, an optimal value depends on many factors including the number of all skyline points, which should be considered as unknown. Too few pivot points cannot distribute all points to a large number of subspaces but too many pivot points will clearly slow down our method. In general, we use a sorting-based process to select pivot points and merge dominating subspaces, the maximum dominating subspace is constructed in iteration. Each skyline point can assign a dominating subspace to a point $q \in \mathcal{P}$ and all dominated points will be pruned.

In each iteration, we determine the change of point number of each subspace size, that is, the number of points within the same subspace with the same size, which is limited by $d-1$ instead of all $2^d - 2$ subspaces. Let $\mathcal{P}$ be the dataset and D_q be the maximum dominating subspace of a point $q \in \mathcal{P}$, then, for each point q, the subspace D_q may be changed by current pivot point p as $D_q \cup D_{q \prec p}$. Hence, we propose a heuristic measure, the *stability threshold*, denoted by σ, to stop merging dominating subspaces. The stability threshold is the number of subspace sizes that do not change while iterating, which means that no new pivot points are necessary to continue to change the maximum dominating subspaces.

The Algorithm 6 merges dominating subspaces of each non-pruned point in a dataset. The algorithm stops while the stability threshold is reached.

First, in our algorithm we score each point in the dataset $\mathcal{P}$ by its Euclidean distance to the zero point (line 1). In this step, the sorting of all points is not necessary, which requires $\mathcal{O}(N \log N)$ time: assume that the stability threshold σ can be satisfied by k skyline points, the search of minimal score (line 9) can be done in $\mathcal{O}(kN)$ time in the worst case, where $k \ll N$. The algorithm runs in the iterative loop from line 5 to line 24 with respect to the stability measure $\sigma' < \sigma$. In each iteration, any time if all points have been pruned, then the algorithm returns the skyline S and the empty dataset $\mathcal{P}$ (line 7), so that the whole computation can be terminated. With pruning dominated points in the dataset, the point p having the minimal score is immediately a skyline point (line 9 and 10) and can be pruned from the dataset (line 11). Then, the point p will be compared with each point $q \in \mathcal{P}$ in the dataset (line 12 to line 22) by computing the dominating subspace $D_{q \prec p}$ (line 13). If $S_{q \prec p} = \emptyset$, then $p \not\prec q$ or $p = q$ (here we denote by $p = q$ that $\forall i \in D, p[i] = q[i]$), in any case, q will be pruned from $\mathcal{P}$ (line 18) and

Algorithm 6: Merge

Input: The dataset $\mathcal{P}$ and the stability threshold σ

Output: The initial skyline S of $\mathcal{P}$ and $\mathcal{P}$ with non-pruned points

```
1  Score each point q ∈ P by Euclidean distance to the zero point
2  Initialize the maximum dominating subspace D_q = ∅ for each point q ∈ P
3  S ← ∅
4  σ' ← 0
5  while σ' < σ do
6      if P = ∅ then
7          return S, P
8      p ← the point with the minimal score (which is a skyline point)
9      S ← S ∪ {p}
10     P ← P \ {p}
11     foreach q ∈ P do
12         Compute dominating subspace D_{q≺p}
13         if D_{q≺p} = ∅ then
14             if q = p then
15                 S ← S ∪ {q}
16             P ← P \ {q}
17             continue
18         D_q ← D_q ∪ D_{q≺p}
19     σ' ← compute the stability of point distribution
20 return S, P
```

q will be added to the skyline if $p = q$ (line 15 to line 17); otherwise, we merge the maximum dominating subspace $D_{q \prec p}$ of q (line 21). At the end of each iteration, the stability measure σ' will be updated (line 23). Finally, the algorithm returns the skyline S and the dataset $\mathcal{P}$ that contains non-pruned points only (line 25), that is $\forall q \in \mathcal{P}, S \not\prec q$.

In summary, Algorithm 6 is designed to merge the maximum dominating subspace of each point in order to distribute the points in the dataset to as much as possible subspaces, the stability threshold σ is set to control the algorithm. Because each merge procedure of the maximum dominating subspace requires the dominance tests against all non-pruned points in the dataset, the value of stability threshold is sensitive to the performance of our method. For small datasets,

the selection of stability threshold is less important because any skyline algorithm can finish in short time; for large datasets, the stability threshold can be tested from a random sample of the dataset. We also note that any measure can be applied to stop dominating subspace merging.

Since a maximum dominating space can be assigned to each point, this principle can be used to partition and index skyline points in order to reduce the dominance tests. Lemma 11 shows that $D_{q_1 \prec S} \not\supseteq D_{q_2 \prec S}$ is necessary to determine whether $q_1 \prec q_2$ while q_1 and q_2 have been attributed a subspace generated from a set of skyline points S.

Lemma 12. *Given a dataset $\mathcal{P}$, let S be a set of skyline points of $\mathcal{P}$ and $D_{q \prec S}$ be the superposed dominating subspace of a point $q \in \mathcal{P}$, where all points dominated by S have been pruned. Let p be a skyline point in $\mathcal{P}$, then, the dominance tests to determine whether a point $q \in \mathcal{P}$ is a skyline point can be done only with each skyline point p such that $D_{p \prec S} \supseteq D_{q \prec S}$, in the condition of presorting.* ■

Proof. In the condition of presorting like [9, 10, 24, 25, 77], it is enough to compare a point with all known skyline points to determine whether the testing point is in the skyline. The rest of the proof is as the proof of Lemma 11, if q_1 in Lemma 11 is a skyline point. □

2.4.3 Subset Query for Fast Dominance Tests

Our result shows that subspace based partitioning of skyline points can significantly reduce the dominance tests: a testing point is necessary to compare only with the skyline points with the maximum dominating subspace specified in Lemma 12. Therefore, if the skyline points can be stored and retrieved by a generic container, then this container can be used to improve any skyline algorithm by reducing the total number of dominance tests. So we have the following problem statements.

Problem 1: Design a data structure with which:

1. *each skyline point can be stored/indexed and partitioned by its maximum dominating subspace;*
2. *given a subspace D_q of a testing point q, all skyline points with any subspace $D' \supseteq D_q$ can be efficiently returned.*

Let $D_q^{\neg}$ denote the reversed maximum dominating subspace with respect to D of a point q, then the above problem is to find all subsets of $D_q^{\neg}$ in order to retrieve associated skyline points.

Problem 2: Given a set $\mathcal{X}$ of m subsets of a universe D, for any query set $Q \subseteq D$, return the set $\{Q' \in \mathcal{X} \mid Q' \subset Q\}$.

Many studies have been addressed to the subset query problem. The recent result [21] shows that the subset query can be accomplished in $\mathcal{O}(\frac{dM}{c})$ time. In our context, where d is the the dimensionality, M is the number of maximum dominating subspaces, and $c \leq M$ is a constant. We propose a very simple data structure to resolve our reversed subset query problem in $\mathcal{O}(\frac{d}{2})$ average time for adding a point and in $\mathcal{O}((\frac{d}{2})^2)$ average time for query, note that $d \ll M$.

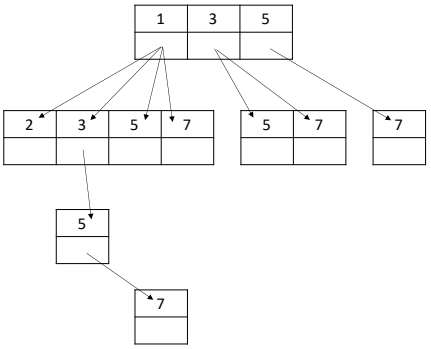


Figure 2.16: A map based data structure for subset query.

Figure 2.16 shows our data structure, which is a prefix tree like structure based on maps, where each value of the key-value pair of a map consists in two parts: a set of skyline points (IDs, references, or pointers, for instance) and a set of sub-maps (references or pointers, for instance). Each key-value pair is considered as a *node*. In considering the access cost, hash map is the best choice for constant insertion and retrieval of nodes, hence, the following description and analysis are all based on hash map. Indeed, any map implementation can be used to construct the proposed data structure, for instance, a sorted map, but in this case, the access will be no longer in constant time but in log time. The index size is the map-based prefix tree that contains all dimensions in concerned subspaces plus the total number of skyline points represented by IDs, references, or pointers.

In the example illustrated in Figure 2.16, the reversed maximum dominating subspaces are organized by the index of dimensions, where in this example we have the following subspaces

$$\{\{1,2\},\{1,3,5,7\},\{1,5\},\{1,7\},\{3,5\},\{3,7\},\{5,7\}\}$$

in the tree. Each node consists of the index of subspace and the set of all points (any point can

Algorithm 7: Store

Input: A point q with a subspace D_q

```
1 node ← root
2 foreach i ∈ D_q^¬ do
3     node ← node.get(i)
4 node.put(q)
```

be represented by a pointer, a reference, or its ID, a real copy is not necessary) assigned with this subspace. Given a query set $\{1, 3, 5\}$, we first locate the node 1, the retrieve all points associated with this node, and then, from the node 1, we locate the node 3 with retrieving all associated points, and the node 5. In order to find all subsets of $\{1, 3, 5\}$, the node 3 and the node 5 at the first level (root node) should also be accessed.

We propose the following two subset query algorithms to store and retrieve skyline points. We suppose that each point in the dataset has been attributed a maximum dominating subspace computed from the Algorithm 6, represented by a bit set (which is supported by many programming languages) or a binary vector of size d.

Algorithm 7 is quite simple. With respect to the data structure shown in Figure 2.16, we let the initial node be the root node (line 1). For each dimension in the reversed subspace $D^{\neg_q}$ (the index of the *true* bit with respect to a bit set or of the value 1 with respect to a binary vector), we get the last node by a simple loop (line 2 to line 4). We define that the $get(i)$ method (line 3) of a node returns the sub-node with the index value i, which can finish in constant time while using a hash map to implement the data structure, or in $\mathcal{O}(\log d)$ time if a sorted map is used. If the sub-node with the index value i does not exist, the $get(i)$ method create the sub-node. Finally, the point q is added to the last node (line 5) by the method $put(q)$.

Lemma 13. *Algorithm 7 finishes in $\mathcal{O}(1)$ time in the best case, in $\mathcal{O}(d-1)$ time in the worst case, and in $\mathcal{O}(\frac{d}{2})$ time in the average case.* ■

Proof. In the best case, $|D_q^{\neg}| = 1$, the algorithm finishes immediately. In the worst case, $|D_q^{\neg}| = d - 1$ because q is not dominated by any initial skyline points, therefore the algorithm requires $d - 1$ retrievals of sub-node, which requires $\mathcal{O}(d - 1)$ time with hash map based data structure. In the average case, the mean size of subspaces can be computed by dividing the sum of the total size of all subspaces (which is known as $d2^{d-1}$) by the total number of subspaces (which is 2^d), we have $\frac{d2^{d-1}}{2^d} = \frac{d}{2}$. Hence, this average time complexity is $\mathcal{O}(\frac{d}{2})$. □

Lemma 13 also shows that in the case of $d = 2$, Algorithm 7 finishes always in $\mathcal{O}(1)$, however,

Algorithm 8: Query

Input: A subspace D_q

Output: A set S of points such that $\forall p \in S, D_p not \subseteq D_q$

```
1 S ← root.points
2 foreach node ∈ root.nodes do
3     if node.index ∈ D_q^¬ then
4         query(D_q^¬, node, S)
5 return S
```

Algorithm 9: Recursive query

Input: A subspace D_q, a node, and a set S of points such that $\forall p \in S, D_p \not\subseteq D_q$

```
1 S ← S ∪ node.points
2 foreach node ∈ node.nodes do
3     if node.index ∈ D_q^¬ then
4         query(D_q^¬, node, S)
```

there will be only two independent nodes at the top level to store skyline points, therefore, our method can contribute very limited performance improvement.

Algorithm 8 presents a recursive retrieval of partitioned skyline points with respect to a given subspace D_q, all points distributed to any super set of D_q will be returned. Algorithm 9 corresponds to the *query* method in the line 4 of Algorithm 8. The skyline point set S is updated by each call of the method *query* (Algorithm 9), which is initialized by all points distributed to the root node (line 1 of Algorithm 8).

Lemma 14. *Given a subspace D_q, Algorithm 8 returns the set S such that $\forall p \in S, D_p \supseteq D_q$ in $\mathcal{O}(1)$ time in the best case, in*

$$\mathcal{O}(\frac{(d-1)(d-2)}{2})$$

time in the worst case, and in

$$\mathcal{O}(\frac{(d/2)(d/2-1)}{2})$$

time in the average case. The average time complexity of Algorithm 8 can be considered as $\mathcal{O}((\frac{d}{2})^2)$. ■

Proof. Let $|D_q^{\neg}| = n$, then $\frac{n(n-1)}{2}$ tests must be done with respect to the data structure shown in Figure 2.16 to retrieve all subsets of $D_q^{\neg}$, that is, all super sets of D_q. In the best case, $n = 1$,

1 test returns the subset of $D_q^{\neg}$; in the worst case, $n = d - 1$, hence $\mathcal{O}(\frac{(d-1)(d-2)}{2})$ time is required. In the average case, as shown in the proof of Lemma 13, $n = \frac{d}{2}$, therefore, Algorithm 8 terminates in $\mathcal{O}(\frac{(d/2)(d/2-1)}{2})$ time, which can be considered in $\mathcal{O}((\frac{d}{2})^2)$ time complexity. □

We note that the dimensionality d of the dataset is much less than the cardinality N of the dataset, the $\mathcal{O}((\frac{d}{2})^2)$ average subset query time can be ignored in comparison with $\mathcal{O}(dN^2)$ time or $\mathcal{O}(N \log^{(d-2)} N)$ time. However, our method is not suitable to directly partition points in a dataset because $2^d - 2$ subspaces can be generated in the worst case and in this case $\mathcal{O}((\frac{d}{2})^2 N)$ time is required to retrieve comparable points. Hence, we propose our method to partition skyline points only, and the dominance tests can be performed in general ways. Therefore, the best application of our result is to boost sorting-based skyline algorithms.

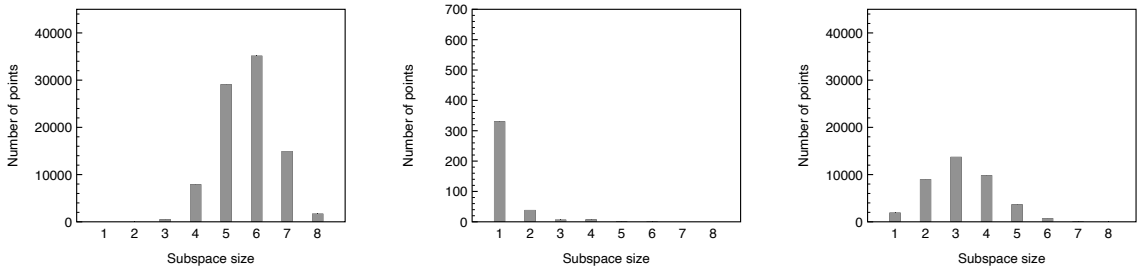
To evaluate the proposed method, we applied it to `SFS`, `SaLSa`, and `SDI` algorithms without changing their original designs, where the main function of our method is to store and to retrieve skyline points. The code is implemented in C++ with the C++11 standard[7] and tested on AMD Epyc 7702 2GHz CPU with 512 GB RAM.

The evaluation metrics are based on the *mean dominance test number* [65] and the *elapsed processor time*, where the mean dominance test number is defined as the ratio of the total number of dominance tests on the total number of points. All elapsed processor time results, in milliseconds, are based on the mean time of 10 runs, all data have been loaded into the main memory before counting.

First, we study the effect of the stability threshold σ, where $1 < \sigma \leq d$ (d is the dimensionality of dataset). Note that it is meaningless to set $\sigma = 1$ because the objective of our method is to balance the distribution of points among subspaces. The results detailed in the full paper [69] show that low-value stability thresholds can effectively reduce the mean number of dominance tests however there is no exact corresponding changes in elapsed processor time on CO and UI (with `SDI-Subset`) datasets. The main reason is that AC data require a huge number of dominance tests so that the time variance among different dominance tests can be statistically neutralized, however CO and UI data require much less dominance tests so the different dominance test time makes sense. Indeed, in comparison with Figure 2.15, Figure 2.17 can show that while $\sigma = 3$, the most of points in AC data are distributed in high subspaces but that in CO and UI data are distributed in low subspaces, which requires less skyline index accesses and the index is much smaller. We have the similar results on other AC/CO/UI datasets with different dimensionality

[7]The source code of our method is available at `https://github.com/dominiquehli/skyline-subset`. We thank Jongwuk Lee for the source code of `BSkyTree-S` and `BSkyTree-P`.

and cardinality, where the fastest σ for `SDI-Subset` is around $d/3$. Therefore, in the reported performance evaluations, the stability threshold σ is set to rounded $d/3$.



(a) 8-D 100K AC dataset. (b) 8-D 100K CO dataset. (c) 8-D 100K UI dataset.

Figure 2.17: Distribution of points with respect to subspace size where the stability threshold is set to 3.

2.4.4 Experimental Evaluation

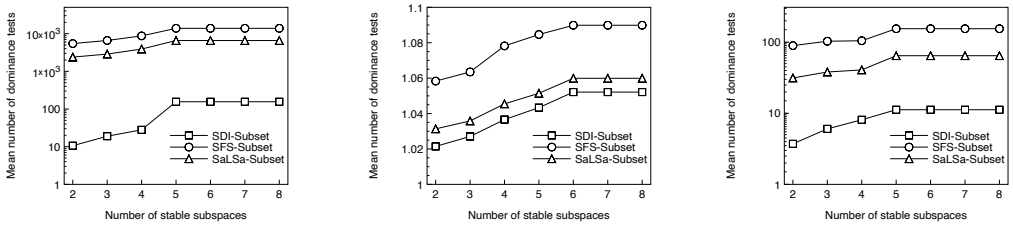
In this section, we report the experimental results of our method. We applied our method to `SFS`, `SaLSa`, and `SDI` algorithms without changing their original designs, the main function of our method is to store and to retrieve skyline points. The code is implemented in C++ with the C++11 standard[8] and tested on AMD Epyc 7702 2GHz CPU with 512 GB RAM[9].

The evaluation metrics are based on the *mean dominance test number* [65] and the *elapsed processor time*, where the mean dominance test number is defined as the ratio of the total number of dominance tests on the total number of points. All elapsed processor time results, in milliseconds, are based on the mean time of 10 runs, all data have been loaded into the main memory before counting.

Effect of Stability Threshold First, we study the effect of the stability threshold σ, where $1 < \sigma \leq d$ (d is the dimensionality of dataset). Note that it is meaningless to set $\sigma = 1$ because the objective of our method is to balance the distribution of points among subspaces. Figure 2.18 and Figure 2.19 show that low-value stability thresholds can effectively reduce the mean number of dominance tests however there is no exact corresponding changes in elapsed processor time on CO and UI (with `SDI-Subset`) datasets. Note that the format of Y axis of AC and

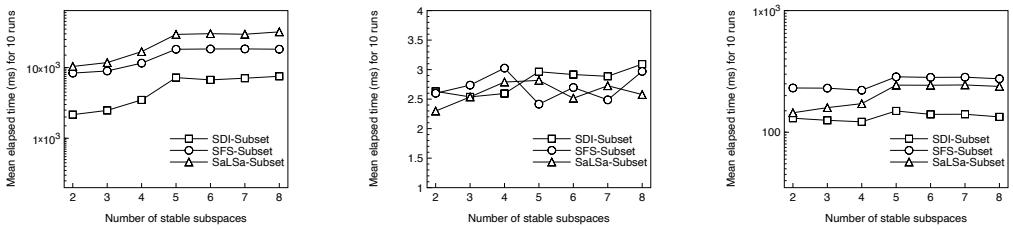
[8]The source code of our method is available at `https://github.com/dominiquehli/skyline-subset`. We thank Jongwuk Lee for the source code of `BSkyTree-S` and `BSkyTree-P`.

[9]The authors benefited from the use of the cluster at the Centre de Calcul Scientifique en réegion Centre-Val de Loire, France.



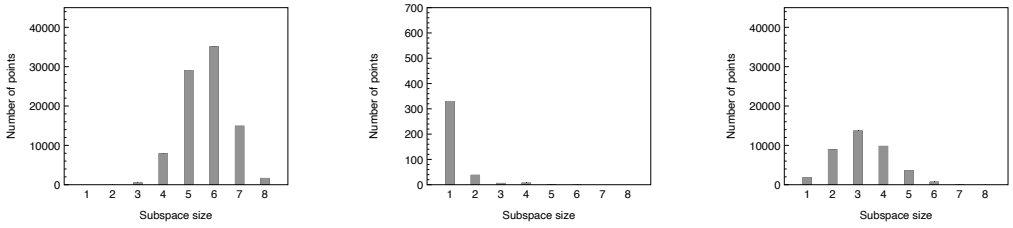
(a) 8-D 100K AC dataset. (b) 8-D 100K CO dataset. (c) 8-D 100K UI dataset.

Figure 2.18: Mean dominance test numbers with respect to stable subspaces on 8-D 100K datasets.



(a) 8-D 100K AC dataset. (b) 8-D 100K CO dataset. (c) 8-D 100K UI dataset.

Figure 2.19: Elapsed processor time (ms) for 10 runs with respect to stable subspaces 8-D 100K datasets.



(a) 8-D 100K AC dataset. (b) 8-D 100K CO dataset. (c) 8-D 100K UI dataset.

Figure 2.20: Distribution of points with respect to subspace size where the stability threshold is set to 3.

Dimensionality	2-D	4-D	6-D	8-D	10-D	12-D	16-D	20-D	24-D	
AC datasets	55	4806	40423	95898	139770	166641	190360	197142	199048	
CO datasets	3	20	57	148	252	1303	4473	9582	20891	
UI datasets	14	382	3275	13046	37379	77200	154827	190501	198742	
Cardinality	100K	200K	300K	400K	500K	600K	700K	800K	900K	1M
AC datasets	55969	95898	131632	164204	193488	221317	247651	273151	297131	320138
CO datasets	135	148	189	203	219	238	260	259	202	208
UI datasets	55969	95898	131632	164204	193488	221317	247651	273151	297131	320138

Table 2.4: Skyline size of synthetic datasets.

UI datasets is logarithmic and that of CO is linear. The main reason is that AC data require a huge number of dominance tests so that the time variance among different dominance tests can be statistically neutralized, however CO and UI data require much less dominance tests so the different dominance test time makes sense. Indeed, in comparison with Figure 2.15, Figure 2.20 can show that while $\sigma = 3$, the most of points in AC data are distributed in high subspaces but that in CO and UI data are distributed in low subspaces, which requires less skyline index accesses and the index is much smaller. We have the similar results on other AC/CO/UI datasets with different dimensionality and cardinality, where the fastest σ for `SDI-Subset` is around $d/3$. Therefore, in the reported performance evaluations, the stability threshold σ is set to rounded $d/3$.

Effect of Data Type We now report the improvements of `SFS`, `SaLSa`, and `SDI` algorithms with the boost of our method, where the state-of-the-art algorithms `BSkyTree-S` and `BSkyTree-P` are used as the baseline. Two groups of tests have been conducted in order to study the effect of the data dimensionality and the effect of the data cardinality on synthetic AC, CO, and UI datasets. In the first group, the cardinality of all datasets is fixed to 2×10^5 (200K) points, the dimensionality varies from 2-D to 24-D; in the second group, the dimensionality of all datasets is fixed to 8-D, the cardinality varies from 10^5 (100K) to 10^6 (1M) points. Table 2.4 lists the skyline size of all synthetic datasets.

Because the scale of obtained values is huge, our results are presented as numbers listed in Table 2.5 — Table 2.16, where the algorithms with `-Subset` suffix are boosted by our method. Furthermore, we introduce the *performance gains* metric into the presented tables, which is calculated as the ration of any value obtained without boosting on the value boosted by our method. If there is no performance gain, we mark it as "–".

From Table 2.5 to Table 2.8, we study the the performance of our method on AC data. It is not surprising that `BSkyTree-P` is the absolute winner on AC data, and the boost of our method is very limited. We can preview this result from the previous analysis. However, in high-dimensional data, for instance 20-D and 24-D datasets, our method can boost `SFS`, `SaLSa`, and `SDI` up to 30 to 40 times. The reason is that the 200K points have been distributed till to 2^{20} and 2^{24} subspaces, that is also why `SDI-Subset` wins `BSkyTree-P` in 16-D and 24-D.

The effectiveness of our method on CO data is studied from Table 2.9 to Table 2.12. Except `SaLSa` and `SDI`, all other methods require at least on scan of the full dataset. We can see that there is almost no performance gain in many datasets, as already shown in Figure 2.20: it is difficult to distribute CO data. We also see that `SDI` is the winner in mean dominance test numbers because of its early stop mechanism, as `SaLSa`. However `SDI` needs much more time to

finish the computation in comparison with BSkyTree-S because it must first build the dimension index of the full dataset.

Table 2.13 to Table 2.16 prove the performance of our method. We note the in Table 2.13, our method cannot reduce any mean dominance test numbers on the datasets of which the dimensionality is less than 6 in comparison with SaLSa and SDI because of the same reason described with CO data. On 6-D dataset, SaLSa-Subset is the winner, lightly faster then SDI-Subset. However, from 6-D data, the boosted methods SaLSa-Subset and SDI-Subset run faster than BSkyTree-P. From 8-D datasets[10], SDI-Subset becomes the fastest algorithm on UI data.

Dimensionality	2-D	4-D	6-D	8-D	10-D	12-D	16-D	20-D	24-D
SFS	3.66836	141.496	4568.9	23648.6	49406.2	69797.2	90707.7	97192	99059.0
SFS-Subset	3.66836	301.28	2010.01	4884.64	10568.8	15764.2	11408.7	2059.39	2182.93
Gain	—	—	× 2.27	× 4.84	× 4.67	× 4.43	× 7.95	× 47.19	× 45.38
SaLSa	**1.05371**	41.3559	1780.08	10883.6	25035.1	37825.5	53188.3	58872	61164.7
SaLSa-Subset	3.66836	210.186	702.753	1871.68	4751.66	7716.71	6154.24	1195.77	1264.72
Gain	—	—	× 2.53	× 5.81	× 5.27	× 4.90	× 8.64	× 49.23	× 48.36
SDI	4.1517	57.3677	611.713	1775.54	2908.51	3544.81	3975.69	3748.68	3538.47
SDI-Subset	3.66836	42.8875	147.725	260.142	601.158	911.765	655.527	**132.627**	**125.826**
Gain	× 1.13	× 1.34	× 4.14	× 6.83	× 4.84	× 3.89	× 6.06	× 28.26	× 28.12
BSkyTree-S	2.77378	62.2657	1037.81	2642.78	3251.29	2770.63	1570.63	1375.05	803.409
BSkyTree-P	2.96938	**20.8456**	**70.0527**	**153.205**	**240.783**	**263.435**	**382.818**	359.256	466.324

Table 2.5: Mean dominance test numbers on synthetic 200K AC dataset with respect to the dimensionality from 2-D to 24-D.

Dimensionality	2-D	4-D	6-D	8-D	10-D	12-D	16-D	20-D	24-D
SFS	32.6347	316.748	11519.5	79646.1	200778	290403	438638	513578	495081.0
SFS-Subset	14.4623	794.527	6220.12	18914.7	35340.6	55567.1	39347.6	20817.5	16630.1
Gain	× 2.26	—	× 1.85	× 4.21	× 5.68	× 5.23	× 11.15	× 24.67	× 29.77
SaLSa	31.942	149.111	5897.42	39627.7	134451	174043	294857	338326	410082.0
SaLSa-Subset	17.4618	785.225	7097.27	25528.3	50362.1	93603	98041.8	25432.1	17157.5
Gain	× 1.83	—	—	× 1.55	× 2.67	× 1.86	× 3.01	× 13.30	× 23.90
SDI	48.2699	427.674	7512.17	27096.3	42502.4	52725.2	60998.5	57874.9	52973.4
SDI-Subset	15.7008	538.322	1579.89	5197.69	10514.9	14936.4	12372.4	**12074.1**	**9632.24**
Gain	× 3.07	—	× 4.75	× 5.21	× 4.04	× 3.53	× 4.93	× 4.79	× 5.50
BSkyTree-S	**8.2063**	382.894	12472.8	36874.9	59113.1	58579	52648.1	51648.3	40451.4
BSkyTree-P	8.7188	**82.9531**	**696.73**	**2139.81**	**4062.37**	**5389.26**	**10650**	17095.5	29731.9

Table 2.6: Elapsed processor time (ms) on synthetic 200K AC dataset with respect to the dimensionality from 2-D to 24-D.

Finally, our experimental results show that BSkyTree-P is the best choice for AC data, BSkyTree-S is always the winner on CO data and low-dimensional data (for instance, $d < 6$), and the boosted methods, particularly SDI-Subset, is best for UI data. Besides, the usefulness

[10]The 16-D dataset is exceptional. We have tested different 16-D UI datasets from 100K to 1M points, all results are very similar to that of AC datasets.

Cardinality	100K	200K	300K	400K	500K	600K	700K	800K	900K	1M
SFS	16094.3	23648.6	29744	34746.4	38630.9	42150.5	45269.6	48205	50733.8	53040.0
SFS-Subset	5448.8	4884.64	6506.3	9785.86	12398	12136.6	14463.9	15638.7	16222.5	18461.7
Gain	× 2.95	× 4.84	× 4.57	× 3.55	× 3.12	× 3.47	× 3.13	× 3.08	× 3.13	× 2.87
SaLSa	7686.93	10883.6	13400.6	15422.7	16974.9	18335.3	19523.3	20675.1	21621.6	22470.2
SaLSa-Subset	2380.6	1871.68	2471.55	3748.81	4780.51	4588.65	5453.09	5776.88	5950.9	6818.11
Gain	× 3.23	× 5.81	× 5.42	× 4.11	× 3.55	× 4	× 3.58	× 3.58	× 3.63	× 3.30
SDI	1197.53	1775.54	2285.43	2715.21	3057.96	3651.34	3679.31	3955.05	4206.04	4439.36
SDI-Subset	380.506	260.142	347.751	592.091	773.557	706.018	991.316	1005.2	1111.23	1323.31
Gain	× 3.15	× 6.83	× 6.57	× 4.59	× 3.95	× 5.17	× 3.71	× 3.93	× 3.79	× 3.35
BSkyTree-S	1840.34	2642.78	3294.33	3825.6	4226.94	4585.38	4924.65	5224.82	5495.08	5746.91
BSkyTree-P	**138.385**	**153.205**	**165.693**	**172.064**	**179.472**	**184.118**	**185.679**	**189.311**	**191.758**	**191.702**

Table 2.7: Mean dominance test numbers on synthetic 8-D AC dataset with respect to the cardinality from 100K to 1M.

Cardinality	100K	200K	300K	400K	500K	600K	700K	800K	900K	1M
SFS	19460.6	79646.1	217338	397901	443378	569681	1606980	2139990	2393550	2710520.0
SFS-Subset	5626.8	18914.7	28654.9	54100.5	96181.3	115770	166344	223040	269236	374428.0
Gain	× 3.46	× 4.21	× 7.58	× 7.35	× 4.61	× 4.92	× 9.66	× 9.59	× 8.89	× 7.24
SaLSa	17124.8	39627.7	217466	420102	572934	433723	931318	1159820	1112750	1204340.0
SaLSa-Subset	7106.61	25528.3	28876.9	60517.9	106613	130710	174473	230938	283176	398255.0
Gain	× 2.41	× 1.55	× 7.53	× 6.94	× 5.37	× 3.32	× 5.34	× 5.02	× 3.93	× 3.02
SDI	5348.27	27096.3	44927.6	84776.9	131919	199842	248619	316241	386424	459770.0
SDI-Subset	1443.94	5197.69	5854.34	11830	20698.5	23743.9	37005.5	43266.8	70195.4	94795.2
Gain	× 3.70	× 5.21	× 7.67	× 7.17	× 6.37	× 8.42	× 6.72	× 7.31	× 5.50	× 4.85
BSkyTree-S	8754.06	36874.9	65591.6	113072	169633	232899	302621	379215	450423	503851.0
BSkyTree-P	**855.191**	**2139.81**	**3231.9**	**4597**	**6280.92**	**7493.29**	**9186.88**	**10759.3**	**12452.9**	**13986.8**

Table 2.8: Elapsed processor time (ms) on synthetic 8-D AC dataset with respect to the cardinality from 100K to 1M.

Dimensionality	2-D	4-D	6-D	8-D	10-D	12-D	16-D	20-D	24-D
SFS	1	1.00095	1.01652	1.09495	1.21073	6.14055	53.3143	233.428	1095.44
SFS-Subset	1	1.00035	1.01202	1.04543	1.06489	2.51762	9.72743	18.7451	45.6721
Gain	—	× 1	× 1	× 1.05	× 1.14	× 2.44	× 5.48	× 12.45	× 23.98
SaLSa	0.000015	0.00032	0.01269	0.054905	0.09861	1.74134	18.7257	91.0814	474.549
SaLSa-Subset	1	1.00005	1.00721	1.02232	1.03373	1.46619	3.07304	7.96234	20.3894
Gain	—	—	—	—	—	× 1.19	× 6.09	× 11.44	× 23.27
SDI	**0**	**0.00014**	**0.003715**	**0.02436**	**0.03205**	**1.08491**	8.20914	21.5959	52.8217
SDI-Subset	1	1.00004	1.00556	1.01909	1.0234	1.30619	**1.86312**	**3.4836**	**6.7105**
Gain	—	—	—	—	—	—	× 4.41	× 6.20	× 7.87
BSkyTree-S	1.00011	1.00383	1.07056	1.98896	8.24004	3.38524	13.6984	19.9785	33.2862
BSkyTree-P	1.0001	1.00379	1.07219	1.98422	8.22813	3.14621	13.0169	17.1449	33.0747

Table 2.9: Mean dominance test numbers on synthetic 200K CO dataset with respect to the dimensionality from 2-D to 24-D.

Dimensionality	2-D	4-D	6-D	8-D	10-D	12-D	16-D	20-D	24-D
SFS	20.4586	29.0367	38.259	40.8126	40.8972	58.4573	204.267	783.707	3411.88
SFS-Subset	2.0127	2.838	5.11	8.6771	7.2286	37.1798	331.794	1179.47	4011.02
Gain	× 10.16	× 10.23	× 7.49	× 4.70	× 5.66	× 1.57	—	—	—
SaLSa	25.8268	27.3071	29.3963	30.6528	30.7115	40.746	97.8189	353.89	1760.31
SaLSa-Subset	2.109	3.6676	4.2471	6.4295	7.0722	20.3191	140.599	474.649	1540.02
Gain	× 12.25	× 7.45	× 6.92	× 4.77	× 4.34	× 2.01	—	—	× 1.14
SDI	37.8009	91.3422	135.347	178.336	221.828	276.807	440.341	659.926	1064.64
SDI-Subset	2.4486	3.9579	3.9521	6.3952	7.6565	20.9283	**95.6926**	**266.429**	**892.259**
Gain	× 15.44	× 23.08	× 34.25	× 27.89	× 28.97	× 13.23	× 4.60	× 2.48	× 1.19
BSkyTree-S	**0.0732**	**0.1962**	**0.2866**	**0.8575**	**2.0497**	**13.768**	135.577	281.453	933.818
BSkyTree-P	0.8813	1.4264	2.1908	6.1156	11.9934	23.1115	124.2	330.976	1209.79

Table 2.10: Elapsed processor time (ms) on synthetic 200K CO dataset with respect to the dimensionality from 2-D to 24-D.

Cardinality	100K	200K	300K	400K	500K	600K	700K	800K	900K	1M
SFS	1.15714	1.09495	1.11023	1.0878	1.09627	1.10625	1.1115	1.10019	1.0526	1.05208
SFS-Subset	1.05833	1.04543	1.04546	1.03052	1.03156	1.0359	1.03737	1.03598	1.01502	1.0148
Gain	× 1.09	× 1.05	× 1.06	× 1.06	× 1.06	× 1.07	× 1.07	× 1.06	× 1.04	× 1.04
SaLSa	0.09037	0.0549	0.0568	0.042585	0.041912	0.0379	0.0393	0.03806	0.01542	0.015499
SaLSa-Subset	1.03135	1.02232	1.02243	1.01315	1.01379	1.01574	1.01626	1.01845	1.00788	1.00828
Gain	—	—	—	—	—	—	—	—	—	—
SDI	**0.034**	**0.0243**	**0.0398**	**0.0213**	**0.0236**	**0.0205**	**0.02096**	**0.022**	**0.00476**	**0.0048**
SDI-Subset	1.0257	1.0191	1.0163	1.0098	1.001	1.0123	1.0122	1.0149	1.0058	1.0057
Gain	—	—	—	—	—	—	—	—	—	—
BSkyTree-S	2.37996	1.98896	1.19064	1.74626	1.85352	1.15658	1.7801	1.95128	1.02078	1.10242
BSkyTree-P	2.35791	1.98422	1.19132	1.74511	1.8546	1.15933	1.78536	1.95586	1.02092	1.10346

Table 2.11: Mean dominance test numbers on synthetic 8-D CO dataset with respect to the cardinality from 100K to 1M.

Cardinality	100K	200K	300K	400K	500K	600K	700K	800K	900K	1M
SFS	10.4544	40.8126	47.7369	66.9709	106.479	122.469	122.189	139.056	155.7	171.368
SFS-Subset	2.2779	8.6771	13.5451	13.7252	31.7945	21.079	21.1029	22.3662	42.2938	32.2714
Gain	× 4.59	× 4.70	× 3.52	× 4.88	× 3.35	× 5.81	× 5.79	× 6.22	× 3.68	× 5.31
SaLSa	14.084	30.6528	44.4031	59.58	74.613	91.1816	103.349	119.546	134.693	146.871
SaLSa-Subset	1.935	6.4295	7.4033	8.454	10.7574	14.6004	15.0803	17.5672	20.1752	21.2981
Gain	× 7.28	× 4.77	× 6	× 7.05	× 6.94	× 6.25	× 6.85	× 6.81	× 6.68	× 6.90
SDI	79.3928	178.336	256.307	343.177	442.941	536.764	628.812	724.978	810.822	913.288
SDI-Subset	2.8573	6.3952	10.4126	11.9223	23.9418	17.4361	20.745	25.2528	34.5934	31.4668
Gain	× 27.79	× 27.89	× 24.62	× 28.78	× 18.50	× 30.78	× 30.31	× 28.71	× 23.44	× 29.02
BSkyTree-S	**0.7982**	**0.8575**	**1.5527**	**2.0601**	**3.6962**	**3.2064**	**3.434**	**3.8724**	**2.0915**	**2.0029**
BSkyTree-P	3.9025	6.1156	6.1724	13.3343	16.8883	14.1379	22.9712	28.8244	16.5354	21.6284

Table 2.12: Elapsed processor time (ms) on synthetic 8-D CO dataset with respect to the cardinality from 100K to 1M.

Dimensionality	2-D	4-D	6-D	8-D	10-D	12-D	16-D	20-D	24-D
SFS	1.00995	2.07385	38.3914	459.212	3568.61	15015.3	60021.5	90753.8	98751.0
SFS-Subset	1.00995	1.59669	17.9581	106.088	710.948	3082.02	26693.7	11347	2034.05
Gain	—	× 1.30	× 2.14	× 4.33	× 5.02	× 4.87	× 2.25	× 8	× 48.55
SaLSa	0.00975	**0.532165**	12.3735	167.818	1430.21	6809.19	32449.3	53197.5	58806.3
SaLSa-Subset	1.00995	1.39582	7.60971	43.9786	273.812	1364.84	14261.8	6486.06	1194.34
Gain	—	—	× 1.63	× 3.82	× 5.22	× 4.99	× 2.28	× 8.20	× 49.24
SDI	**0.00602**	0.54185	12.7038	75.1358	371.175	1305.16	2884.92	3712.77	3484.04
SDI-Subset	1.00995	1.11015	**3.02379**	**10.2973**	**47.508**	185.514	1512.53	645.27	**114.729**
Gain	—	—	× 4.20	× 7.30	× 7.81	× 7.04	× 1.91	× 5.75	× 30.37
BSkyTree-S	1.01232	7.0461	31.7958	133.137	332.591	717.365	1137.52	936.944	558.125
BSkyTree-P	1.01279	7.21108	25.7719	79.2428	93.519	**158.027**	**529.538**	**582.166**	432.831

Table 2.13: Mean dominance test numbers on synthetic 200K UI dataset with respect to the dimensionality from 2-D to 24-D.

Dimensionality	2-D	4-D	6-D	8-D	10-D	12-D	16-D	20-D	24-D
SFS	23.1289	40.7084	120.397	1268.57	11673.8	68903.9	313820	472481	559174.0
SFS-Subset	3.3954	12.6331	99.9347	696.141	2832.31	16612.6	122205	46537.1	13353.8
Gain	× 6.81	× 3.22	× 1.20	× 1.82	× 4.12	× 4.15	× 2.57	× 10.15	× 41.87
SaLSa	20.8913	25.4035	72.0775	608.382	5749.55	32369	235476	321931	355698.0
SaLSa-Subset	3.0395	9.6911	**59.2101**	403.433	2367.69	18146.9	249288	100806	17245.0
Gain	× 6.87	× 2.62	× 1.22	× 1.51	× 2.43	× 1.78	—	× 3.19	× 20.63
SDI	35.8351	82.9728	265.237	988.093	5087.94	23951.7	52147.6	75153.3	60557.6
SDI-Subset	2.3276	11.2018	83.3118	**337.974**	**755.396**	**2447.71**	31524	**12808.6**	**9872.25**
Gain	× 15.40	× 7.41	× 3.18	× 2.92	× 6.74	× 9.79	× 1.65	× 5.87	× 6.13
BSkyTree-S	**0.1235**	**7.6218**	104.15	896.382	5651.38	15539.5	42210.9	38429	32554.3
BSkyTree-P	1.155	24.1729	95.2192	434.162	1146.56	2879.16	**13397.5**	27845.5	32266.7

Table 2.14: Elapsed processor time (ms) on synthetic 200K UI dataset with respect to the dimensionality from 2-D to 24-D.

Cardinality	100K	200K	300K	400K	500K	600K	700K	800K	900K	1M
SFS	460.313	459.212	499.667	531.194	522.63	507.032	522.836	528.471	520.552	507.664
SFS-Subset	89.5718	106.088	129.523	122.864	120.545	114.264	141.636	144.664	139.262	139.63
Gain	× 5.14	× 4.33	× 3.86	× 4.32	× 4.34	× 4.44	× 3.69	× 3.65	× 3.74	× 3.64
SaLSa	175.827	167.818	176.903	185.034	179.431	171.196	176.839	178.822	174.434	167.308
SaLSa-Subset	31.4026	43.9786	38.6649	39.2604	36.4398	34.823	38.6292	39.7039	37.9282	37.959
Gain	× 5.60	× 3.82	× 4.58	× 4.71	× 4.92	× 4.92	× 4.58	× 4.50	× 4.60	× 4.41
SDI	70.879	75.1358	86.6211	96.2773	97.7019	101.154	103.341	105.916	105.649	105.536
SDI-Subset	**8.84229**	**10.2973**	**9.57091**	**9.85854**	**9.60977**	**9.08958**	**9.54601**	**9.87711**	**8.79965**	**8.82683**
Gain	× 8.02	× 7.30	× 9.05	× 9.77	× 10.17	× 11.13	× 10.83	× 10.72	× 12.01	× 11.96
BSkyTree-S	140.838	133.137	133.913	131.735	118.279	111.169	88.9518	88.8225	162.526	100.47
BSkyTree-P	85.5219	79.2428	79.191	72.7893	61.2489	52.4952	32.0429	36.6836	99.3828	38.4963

Table 2.15: Mean dominance test numbers on synthetic 8-D UI dataset with respect to the cardinality from 100K to 1M.

Cardinality	100K	200K	300K	400K	500K	600K	700K	800K	900K	1M
SFS	569.967	1268.57	2519.03	3879.07	4406.86	5569.82	6462.62	9452.43	7414.07	8824.39
SFS-Subset	213.963	696.141	1213.75	1679.07	2010.92	2396.66	3437.86	4159.97	4627.68	4730.24
Gain	× 2.66	× 1.82	× 2.08	× 2.31	× 2.19	× 2.32	× 1.88	× 2.27	× 1.60	× 1.87
SaLSa	320.66	608.382	1001.58	1423.56	1707.41	1943.35	2388.99	2749.06	2987.54	3272.88
SaLSa-Subset	139.294	403.433	585.847	799.217	877.945	1069.97	1547.82	1848.41	1622.37	1709.46
Gain	× 2.30	× 1.51	× 1.71	× 1.78	× 1.94	× 1.82	× 1.54	× 1.49	× 1.84	× 1.91
SDI	361.305	988.093	1263.2	1879.79	2403.84	2938.16	3639.35	4111.78	4371.11	4802.71
SDI-Subset	**99.4546**	**337.974**	**386.365**	**481.068**	**550.036**	**546.609**	**898.925**	**1049.89**	**1082.65**	**1220.89**
Gain	× 3.63	× 2.92	× 3.27	× 3.91	× 4.37	× 5.38	× 4.05	× 3.92	× 4.04	× 3.93
BSkyTree-S	380.996	896.382	1157.3	1599.01	1944.32	2368.51	2818.7	3104.79	3777.96	4034.15
BSkyTree-P	210.467	434.162	600.422	790.853	916.168	1012.09	1041.63	1189.88	2039.03	1428.89

Table 2.16: Elapsed processor time (ms) on synthetic 8-D UI dataset with respect to the cardinality from 100K to 1M.

of our method is also limited in low-dimensionality domains, such as 2-D to 4-D datasets. The main reason is that the number of subspaces is not enough to boost SFS, SaLSa, and SDI: there is no subspace in 2-D dataset and only $2^4 - 2 = 14$ subspaces can be generated to distribute points. In the presented experiments, any tested algorithm can finish the skyline computation of a 4-D UI dataset with 2×10^5 (200K) points in millisecond-level, so our method can not give additional performance gain. However, on larger dataset, for instance, on a 4-D UI dataset with 10^6 (1M) points, our extended experiments show that all boosted methods, SFS-Subset, SaLSa-Subset, and SDI-Subset perform better than BSkyTree-S and BSkyTree-P, where the skyline contains 423 points, as shown in Figure 2.17 (DT: Mean dominance test numbers; RT: Elapsed processor time).

Method	**DT**	**RT**
SFS	1.38777	193.684 ms
SFS-Subset	1.39038	74.731 ms
Performance Gain	–	×2.59
SaLSa	0.298219	169.865 ms
SaLSa-Subset	1.17633	49.637 ms
Performance Gain	–	×3.42
SDI	0.388056	574.287 ms
SDI-Subset	1.03643	64.737 ms
Performance Gain	–	×8.87
BSkyTree-S	8.96764	111.378 ms
BSkyTree-P	9.06151	120.388 ms

Table 2.17: Results on 4-D UI dataset with 1M points.

Results on Real Datasets We also tested our method on the three real world datasets `HOUSE` (6-D, 127,931 points, 5,774 skyline points), `NBA` (8-D, 17,264 points, 1,796 skyline points), and `WEATHER` (15-D, 566,268 points, 26,713 skyline points) [22] as listed from Table 2.18 to Table 2.20 (DT: Mean dominance test numbers; RT: Elapsed processor time).

Method	**DT**	**RT**	σ
`SFS`	173.446	298.147 ms	
`SFS-Subset`	135.977	245.348 ms	4
Performance Gain	×1.28	×1.22	
`SaLSa`	94.4504	211.248 ms	
`SaLSa-Subset`	73.2253	249.887 ms	4
Performance Gain	×1.29	–	
`SDI`	21.6086	214.724 ms	
`SDI-Subset`	18.9641	111.7 ms	4
Performance Gain	×1.14	×1.92	
`BSkyTree-S`	60.3785	287.097 ms	
`BSkyTree-P`	13.1706	56.219 ms	

Table 2.18: The HOUSE dataset.

Method	**DT**	**RT**	σ
`SFS`	149.09	29.167 ms	
`SFS-Subset`	104.749	23.84 ms	2
Performance Gain	×1.42	×1.22	
`SaLSa`	115.763	26.005 ms	
`SaLSa-Subset`	74.4728	24.236 ms	2
Performance Gain	×1.55	×1.07	
`SDI`	17.817	24.488 ms	
`SDI-Subset`	18.5333	22.376 ms	2
Performance Gain	–	×1.04	
`BSkyTree-S`	54.5101	26.769 ms	
`BSkyTree-P`	39.7169	17.681 ms	

Table 2.19: The NBA dataset.

Our experimental results show that our method can boost `SFS`, `SaLSa`, and `SDI`, where all stability threshold σ values have been manually adjusted. We note and analyzed the relatively limited (< 2) effectiveness on these three datasets. `HOUSE` is an AC type dataset, we have already

Method	DT	RT	σ
SFS	10793.9	43279.4 ms	
SFS-Subset	9530.47	43231.2 ms ms	3
Performance Gain	$\times$1.42	1.0001 = –	
SaLSa	3080.55	14349.4 ms	
SaLSa-Subset	2630	26169.5 ms	3
Performance Gain	$\times$1.17	–	
SDI	537.066	10653.4 ms	
SDI-Subset	525.12	10617.3 ms	3
Performance Gain	$\times$1.02	$\times$1.04	
BSkyTree-S	3116.63	43495.9 ms	
BSkyTree-P	561.723	11694.2 ms	

Table 2.20: The WEATHER dataset.

shown the limits of our method with such a data type. NBA is a small dataset with only 17,264 points dataset, SaLSa cannot be boosted at all in terms of elapsed processor time because the I/O of our proposed skyline index requires additional processor time. WEATHER dataset consists of 15 dimensions, so SDI works well because it was designed for high-dimensionality domains. Besides, there are a large number of duplicate values in several dimensions in the WEATHER dataset, and this factor causes that there may be a lot of skyline points in one single node of our proposed skyline index, which will affect the performance of the skyline index. However, both SDI and SDI-Subset perform better then BSkyTree-P.

2.4.5 Conclusion

In this section, we present a subset approach to efficient skyline computation, which is designed to boost sorting-based skyline algorithms. We proposed a subspace union method to assign all points a maximum dominating subspace, with which the dominance tests between skyline points and testing points are only necessary between comparable subspaces. In order to efficiently retrieve skyline points for dominance tests with respect to maximum dominating subspaces, we proposed a subset query method to index the skyline. Our theoretical analysis and experimental results show that the skyline computation can be boosted by our subspace union and subset query method. The particular perspective of the presented work includes: (1) extending the proposed method to AC and CO data; (2) developing a cost model to improve the stability threshold in order to find the best number of pivot points; (3) adapting the proposed method to updating

data such as data streams.

2.5 Perspectives

The skyline query processing related problems is very similar to the pattern mining problems, we are facing to: (1) how to compute the skyline more faster, for which there are many algorithms have been developed; (2) how to return less results, for which the skycube and subspace query is well studied; (3) how to return more useful results, for which the most objective measure is the top-k queries, dominance-based or subspace-based. Note that the most of skyline algorithms are based on in-memory computation, therefore, in the big data context, it is hard to compute the above different skylines in large datasets even a few MapReduce-based methods exist [88, 61, 101, 133, 135].

I have many perspectives in skyline query processing, which are based on the `SDI` framework (Section 2.2) and my `Subset` approach (Section 2.4):

- Make the general skyline computation more faster. The main idea is to adapt my `Subset` approach to boost also partition-based skyline algorithms because in general, partition-based algorithms run much faster than sorting-based algorithms, even we have shown that `SaLSa+Subset` and `SDI+Subset` outperform the state-of-the-art algorithms on UI data.

- Make the subspace skyline computation more scalable. As introduced in Introduction, the notion of Skycube has been proposed to pre-compute skylines in lattice-based subspaces, of which there are $2^d - d - 1$ subspaces if we do not consider single-dimension skylines (just sorting the values). However, Skycube does not fit the requirements of the Big Data of today: for instance, given a huge dataset of 32 dimensions, a cube of 2,294,967,263 nodes should be created to store subspace skylines. I am interested in designing a fast on-demand subspace skyline algorithms based on `SDI+Subset`, with the fusion (bottom-up) or division (top-down) of cached results.

- Make the top-k subspace skyline computation more scalable and more efficient. As previously mentioned, there are currently two definitions of top-k skylines: dominance-based and subspace-based. We have shown that with the increase of dimensionality of data, the size of skyline increases. For instance, while 90% of points are skyline points in a dataset, dominance-based top-k skyline makes non sense because almost all points are in the skyline and they dominate only a small part of points. Hence, for my part, I feel subspace-based top-k skyline computation is a topic with exciting challenges on-top of `SDI+Subset`.

- Design efficient MapReduce-based skyline algorithms for huge high-dimensional datasets stored in HDFS/HBase[11]/Hive[12] clusters), where in-memory computation cannot fit the volume of data. This topic is an in working research with Mostafa Bamha in the LIFO Laboratory at the University of Orléans, of which the first result (to submit) shows that `SDI` can be well adapted to compute the skyline directly from HBase. The subspace and the top-k skyline queries are also interesting with the MapReduce programming model.

- Besides, in big relational SQL databases, in-memory computation of skylines is unrealistic. Furthermore, with the growth of data analysis outsourcing, the privacy protection in skyline computation with three-party (the data source, the computing power, and the analytic) is a critical topic. Based on the `SDI` framework, a new method for the skyline computation without dominance tests is in development (to submit), with the team of Professor Yuqi Huang at the Zhejiang University (China) and the team of Associate Professor Min Bao at the Zhejiang Sci-Tech University (China). On this topic, we will be looking for an international funding between France and China to produce more results.

The above perspectives are expected to give new solutions to important skyline problems, including big data in external-memory computation or by there-party solutions.

[11] `https://hbase.apache.org`
[12] `https://hive.apache.org`

Conclusion and Future Work

The work presented in this HDR Thesis presents the evolution of my research interests. Since my Master internship, my PhD, and my post-doc, I have made contributions on the traditional area on pattern mining, which is to propose new algorithms to mine existing patterns, particularly sequential patterns, unexpected sequential patterns, and their applications in basic machine learning problems. Since 2016, I have started to investigate the pattern mining with MapReduce in Big Data, and since 2018 I have focused my interest on the applications of pattern mining to different problems in different domains related to machine learning (Chapter 1). Since 2019, my research focus has been turned to skyline query processing problems (Chapter 2). In this thesis I gave perspectives in each of pattern mining (Chapter 1, Section 5) and skyline query processing (Chapter 2, Section 5), the future research direction that I intend to follow.

Undoubtedly, I will continue to apply pattern mining to machine learning problems and to use machine learning to improve the quality of pattern mining, within or without the context of Big Data. Thus, my priorities in pattern mining are the applications of patterns in industrial domains and to mine frequent sequence patterns in large scale datasets with MapReduce and Spark frameworks.

Currently, my highest priority focuses on different skyline query processing problems in large scale datasets with high cardinality and high dimensionality. `SDI` is a scalable framework for efficient skyline computation in high-dimensionality domains, with which subspace, top-k, and dynamic queries can be built and can be enhanced by the subset approach. Another research direction on skyline computation is to use skyline to serve machine learning, including feature selection and model construction. Indeed, skyline, or the Pareto Front, has been studied in the feature selection problem [41, 56], but existing methods rest in low-dimensionality domains, mostly to find non-dominated feature-combinations with respect to the learning effectiveness. With proposed high-dimensional skyline computation framework, it is an interesting direction to find the skyline in a huge number of features with a ranking mechanism such as LIME, in order to directly output the optimized feature set. Furthermore, I am very interested in using skyline

to combine small models to a hybrid one for a concrete learning problem, instead of using a big model to resolve all learning problems. For instance, in a concrete learning problem, all training instances (cardinality) can be evaluate by different models (dimensionality), the subspace and top-k skyline computation allows to filter the best combination of models.

Publications Since 2010

Refereed Journal Publications

1. Yuqi Huang, Dominique H. Li, Haoyi Niu, Donatello Conte: Visual identification of oscillatory two-phase flow with complex flow patterns. Measurement 186 (2021): 110148, 14 pages [58]

2. Lamine Diop, Cheikh Talibouya Diop, Arnaud Giacometti, Dominique H. Li, Arnaud Soulet: Sequential pattern sampling with norm-based utility. Knowledge and Information Systems, 62(5):2029–2065, 2020 [35]

3. Khanh-Chuong Duong, Mostafa Bamha, Arnaud Giacometti, Dominique H. Li, Arnaud Soulet, Christel Vrain: MapFIM+: Memory Aware Parallelized Frequent Itemset Mining In Very Large Datasets. Transactions on Large-Scale Data-and Knowledge-Centered Systems, pages 200–225, 2018 [38]

4. Sandra de Amo, Mouhamadou Saliou Diallo, Cheikh Talibouya Diop, Arnaud Giacometti, Dominique H. Li, Arnaud Soulet: Contextual preference mining for user profile construction. Contextual preference mining for user profile construc- tion. Information Systems, 49:182–199, 2015 [28]

5. Arnaud Giacometti, Dominique H. Li, Patrick Marcel, Arnaud Soulet: 20 years of pattern mining: a bibliometric survey. ACM SIGKDD Explorations Newsletter, 15(1): 41-50, 2014

Refereed Conference Publications

6. Dominique H. Li: Subset Approach to Efficient Skyline Computation. EDBT 2023, pages 391–403 [69]

7. Luong Phat Nguyen, Julien Mille, Dominique H. Li, Donatello Conte, Nicolas Ragot: Efficient dynamic texture classification with probabilistic motifs. ICPR 2022, pages 564–570 [93]

8. Rui Liu, Dominique H. Li: Indexation dynamique pour la maintenance du skyline dans les flux de données. EGC 2021: 389-396

9. Rui Liu, Dominique H. Li: Dynamic Dimension Indexing for Efficient Skyline Maintenance on Data Streams. DASFAA 2020: 272-287 [76]

10. Rui Liu, Dominique H. Li: Efficient Skyline Computation in High-Dimensionality Domains. EDBT 2020: 459-462 [77]

11. Luong Phat Nguyen, Julien Mille, Dominique H. Li, Donatello Conte, Nicolas Ragot: Trajectory Extraction and Deep Features for Classification of Liquid-gas Flow under the Context of Forced Oscillation. VISAPP 2020: 17-26 [92]

12. Rui Liu, Dominique H. Li: Calcul efficace du skyline basé sur l'indexation dimensionnelle. EGC 2020: 285-292

13. Donatello Conte, Dominique H. Li, Yuqi Huang, Haoyi Niu: Une application de classification vidéo en mécanique des fluides diphasiques oscillatoires. EGC 2020: 293-300

14. Lamine Diop, Cheikh Talibouya Diop, Arnaud Giacometti, Dominique H. Li, Arnaud Soulet: Découverte de motifs à la demande dans une base de données distribuée. EGC 2019: 21-32

15. Lamine Diop, Cheikh Talibouya Diop, Arnaud Giacometti, Dominique H. Li, Arnaud Soulet: Sequential Pattern Sampling with Norm Constraints. ICDM 2018: 89-98 [33]

16. Lamine Diop, Cheikh Talibouya Diop, Arnaud Giacometti, Dominique H. Li, Arnaud Soulet: Echantillonnage de motifs séquentiels sous contrainte sur la norme. EGC 2018: 35-46

17. Khanh-Chuong Duong, Mostafa Bamha, Arnaud Giacometti, Dominique H. Li, Arnaud Soulet, Christel Vrain: MapFIM: Memory Aware Parallelized Frequent Itemset Mining in Very Large Datasets. DEXA (1) 2017: 478-495 [37]

18. Arnaud Giacometti, Dominique H. Li, Arnaud Soulet: Balancing the Analysis of Frequent Patterns. PAKDD (1) 2014: 53-64

19. Arnaud Giacometti, Dominique H. Li, Arnaud Soulet: Construction de profils de préférences contextuelles basée sur l'extraction de motifs séquentiels. EGC 2014: 419-430

20. Arnaud Giacometti, Dominique H. Li, Arnaud Soulet: 20 ans de découverte de motifs : une étude bibliographique quantitative. EGC 2013: 133-144

21. Sandra de Amo, Mouhamadou Saliou Diallo, Cheikh Talibouya Diop, Arnaud Giacometti, Dominique H. Li, Arnaud Soulet: Mining Contextual Preference Rules for Building User Profiles. DaWaK 2012: 229-242 [29]

Bibliography

[1] Rakesh Agrawal, Tomasz Imieliński, and Arun Swami. Mining association rules between sets of items in large databases. In *SIGMOD*, pages 207–216, 1993.

[2] Rakesh Agrawal, Ralf Rantzau, and Evimaria Terzi. Context-sensitive ranking. In *SIGMOD*, pages 383–394, 2006.

[3] Rakesh Agrawal and Ramakrishnan Srikant. Mining sequential patterns. In *ICDE*, pages 3–14, 1995.

[4] Rakesh Agrawal, Ramakrishnan Srikant, et al. Fast algorithms for mining association rules. In *VLDB*, volume 1215, pages 487–499, 1994.

[5] Chowdhury Farhan Ahmed, Syed Khairuzzaman Tanbeer, Byeong-Soo Jeong, and Young-Koo Lee. Efficient tree structures for high utility pattern mining in incremental databases. *IEEE Transactions on Knowledge and Data Engineering*, 21(12):1708–1721, 2009.

[6] Vincent Andrearczyk and Paul F Whelan. Convolutional neural network on three orthogonal planes for dynamic texture classification. *Pattern Recognition*, 76:36–49, 2018.

[7] Alejandro Barredo Arrieta, Natalia Díaz-Rodríguez, Javier Del Ser, Adrien Bennetot, Siham Tabik, Alberto Barbado, Salvador García, Sergio Gil-López, Daniel Molina, Richard Benjamins, et al. Explainable artificial intelligence (xai): Concepts, taxonomies, opportunities and challenges toward responsible ai. *Information fusion*, 58:82–115, 2020.

[8] Anthony Bagnall, Jason Lines, Aaron Bostrom, James Large, and Eamonn Keogh. The great time series classification bake off: a review and experimental evaluation of recent algorithmic advances. *Data mining and knowledge discovery*, 31(3):606–660, 2017.

[9] Ilaria Bartolini, Paolo Ciaccia, and Marco Patella. Salsa: Computing the skyline without scanning the whole sky. In *CIKM*, pages 405–414, 2006.

[10] Ilaria Bartolini, Paolo Ciaccia, and Marco Patella. Efficient sort-based skyline evaluation. *ACM Transactions on Database Systems*, 33(4):1–49, 2008.

[11] Kenneth S Bøgh, Sean Chester, Darius Šidlauskas, and Ira Assent. Hashcube: A data structure for space-and query-efficient skycube compression. In *CIKM*, pages 1767–1770, 2014.

[12] Stephan Borzsony, Donald Kossmann, and Konrad Stocker. The skyline operator. In *ICDE*, pages 421–430, 2001.

[13] Jean-Yves Bouguet. Pyramidal implementation of the affine lucas kanade feature tracker description of the algorithm. *Intel Corporation*, 5(1-10):4, 2001.

[14] Craig Boutilier, Ronen I Brafman, Carmel Domshlak, Holger H Hoos, and David Poole. Cp-nets: A tool for representing and reasoning withconditional ceteris paribus preference statements. *Journal of artificial intelligence research*, 21:135–191, 2004.

[15] John Emmett Bush and Alexander Louis London. Design data for" cocktail shaker" cooled pistons and valves. *SAE Transactions*, pages 446–459, 1966.

[16] Shengze Cai, Shichao Zhou, Chao Xu, and Qi Gao. Dense motion estimation of particle images via a convolutional neural network. *Experiments in Fluids*, 60(4):1–16, 2019.

[17] A. B. Chan and N. Vasconcelos. Probabilistic kernels for the classification of auto-regressive visual processes. In *CVPR*, pages 846–851, 2005.

[18] A. B. Chan and N. Vasconcelos. Classifying video with kernel dynamic textures. In *CVPR*, pages 1–6, 2007.

[19] Chee-Yong Chan, HV Jagadish, Kian-Lee Tan, Anthony KH Tung, and Zhenjie Zhang. Finding k-dominant skylines in high dimensional space. In *SIGMOD*, pages 503–514, 2006.

[20] Chee-Yong Chan, HV Jagadish, Kian-Lee Tan, Anthony KH Tung, and Zhenjie Zhang. On high dimensional skylines. In *EDBT*, pages 478–495, 2006.

[21] Moses Charikar, Piotr Indyk, and Rina Panigrahy. New algorithms for subset query, partial match, orthogonal range searching, and related problems. In *International Colloquium on Automata, Languages, and Programming*, pages 451–462, 2002.

[22] Sean Chester, Darius Šidlauskas, Ira Assent, and Kenneth S Bøgh. Scalable parallelization of skyline computation for multi-core processors. In *ICDE*, pages 1083–1094, 2015.

[23] Jan Chomicki. Preference formulas in relational queries. *ACM Transactions on Database Systems*, 28(4):427–466, 2003.

[24] Jan Chomicki, Parke Godfrey, Jarek Gryz, and Dongming Liang. Skyline with presorting. In *ICDE*, volume 3, pages 717–719, 2003.

[25] Jan Chomicki, Parke Godfrey, Jarek Gryz, and Dongming Liang. Skyline with presorting: Theory and optimizations. In *IIPWM*, pages 595–604, 2005.

[26] William W. Cohen, Robert E. Schapire, and Yoram Singer. Learning to order things. *Journal of Artificial Intelligence Research*, 10:243–270, 1999.

[27] Tomás Crivelli, Bruno Cernuschi-Frias, Patrick Bouthemy, and Jian-Feng Yao. Motion textures: modeling, classification, and segmentation using mixed-state markov random fields. *SIAM Journal on Imaging Sciences*, 6(4):2484–2520, 2013.

[28] Sandra De Amo, Mouhamadou Saliou Diallo, Cheikh Talibouya Diop, Arnaud Giacometti, Dominique Li, and Arnaud Soulet. Contextual preference mining for user profile construction. *Information Systems*, 49:182–199, 2015.

[29] Sandra de Amo, Mouhamadou Saliou Diallo, Cheikh Talibouya Diop, Arnaud Giacometti, Haoyuan D Li, and Arnaud Soulet. Mining contextual preference rules for building user profiles. In *DaWaK*, pages 229–242. Springer, 2012.

[30] Jeffrey Dean and Sanjay Ghemawat. Mapreduce: simplified data processing on large clusters. *Communications of the ACM*, 51(1):107–113, 2008.

[31] Xiwen Deng, Jilin Lei, Jun Wen, Zhigao Wen, and Lizhong Shen. Numerical investigation on the oscillating flow and uneven heat transfer processes of the cooling oil inside a piston gallery. *Applied Thermal Engineering*, 126:139–150, 2017.

[32] Xiwen Deng, Jilin Lei, Jun Wen, Zhigao Wen, and Lizhong Shen. Multi-objective optimization of cooling galleries inside pistons of a diesel engine. *Applied Thermal Engineering*, 132:441–449, 2018.

[33] Lamine Diop, Cheikh Talibouya Diop, Arnaud Giacometti, Dominique Li, and Arnaud Soulet. Sequential pattern sampling with norm constraints. In *ICDM*, pages 89–98, 2018.

[34] Lamine Diop, Cheikh Talibouya Diop, Arnaud Giacometti, Dominique Li, and Arnaud Soulet. Découverte de motifs à la demande dans une base de données distribuée. In *EGC*, volume 79, 2019.

[35] Lamine Diop, Cheikh Talibouya Diop, Arnaud Giacometti, Dominique Li, and Arnaud Soulet. Sequential pattern sampling with norm-based utility. *Knowledge and Information Systems*, 62(5):2029–2065, 2020.

[36] Lamine Diop, Cheikh Talibouya Diop, Arnaud Giacometti, and Arnaud Soulet. Pattern sampling in distributed databases. In *ADBIS*, pages 60–74. Springer, 2020.

[37] Khanh-Chuong Duong, Mostafa Bamha, Arnaud Giacometti, Dominique Li, Arnaud Soulet, and Christel Vrain. Mapfim: Memory aware parallelized frequent itemset mining in very large datasets. In *DEXA*, pages 478–495. Springer, 2017.

[38] Khanh-Chuong Duong, Mostafa Bamha, Arnaud Giacometti, Dominique Li, Arnaud Soulet, and Christel Vrain. Mapfim+: Memory aware parallelized frequent itemset mining in very large datasets. *Transactions on Large-Scale Data-and Knowledge-Centered Systems*, pages 200–225, 2018.

[39] Elias Egho, Dominique Gay, Marc Boullé, Nicolas Voisine, and Fabrice Clérot. A user parameter-free approach for mining robust sequential classification rules. *Knowledge and Information Systems*, 52:53–81, 2017.

[40] Martin Ester, Hans-Peter Kriegel, Jörg Sander, Xiaowei Xu, et al. A density-based algorithm for discovering clusters in large spatial databases with noise. In *SIGKDD*, pages 226–231, 1996.

[41] Chao Feng, Chao Qian, and Ke Tang. Unsupervised feature selection by pareto optimization. In *AAAI*, pages 3534–3541, 2019.

[42] Basura Fernando, Elisa Fromont, and Tinne Tuytelaars. Mining mid-level features for image classification. *International Journal of Computer Vision*, 108(3):186–203, 2014.

[43] Chuancong Gao, Jianyong Wang, Yukai He, and Lizhu Zhou. Efficient mining of frequent sequence generators. In *WWW*, pages 1051–1052, 2008.

[44] Arnaud Giacometti, Dominique Li, and Arnaud Soulet. Construction de profils de préférences contextuelles basée sur l'extraction de motifs séquentiels. In *EGC*, pages 419–430, 2014.

[45] A. Gilbert, J. Illingworth, and R. Bowden. Fast realistic multi-action recognition using mined dense spatio-temporal features. In *ICCV*, pages 925–931, 2009.

[46] Parke Godfrey, Ryan Shipley, and Jarek Gryz. Maximal vector computation in large data sets. In *VLDB*, volume 5, pages 229–240, 2005.

[47] Parke Godfrey, Ryan Shipley, and Jarek Gryz. Algorithms and analyses for maximal vector computation. *The VLDB Journal*, 16(1):5–28, 2007.

[48] Wesley Nunes Gonçalves and Odemir Martinez Bruno. Dynamic texture analysis and segmentation using deterministic partially self-Avoiding walks. *Expert Systems with Applications*, 40(11):4283–4300, 2013.

[49] Wesley Nunes Gonçalves, Bruno Brandoli Machado, and Odemir Martinez Bruno. A complex network approach for dynamic texture recognition. *Neurocomputing*, 153:211–220, 2015.

[50] Su Guanghui, Jia Dounan, Kenji Fukuda, and Guo Yujun. Theoretical and experimental study on density wave oscillation of two-phase natural circulation of low equilibrium quality. *Nuclear Engineering and Design*, 215(3):187–198, 2002.

[51] David Gunning, Mark Stefik, Jaesik Choi, Timothy Miller, Simone Stumpf, and Guang-Zhong Yang. Xai—explainable artificial intelligence. *Science robotics*, 4(37):eaay7120, 2019.

[52] Isma Hadji and Richard P Wildes. A spatiotemporal oriented energy network for dynamic texture recognition. In *ICCV*, pages 3066–3074, 2017.

[53] Jiawei Han, Jian Pei, Behzad Mortazavi-Asl, Helen Pinto, Qiming Chen, Umeshwar Dayal, and Meichun Hsu. Prefixspan: Mining sequential patterns efficiently by prefix-projected pattern growth. In *ICDE*, pages 215–224, 2001.

[54] Jiawei Han, Jian Pei, Yiwen Yin, and Runying Mao. Mining frequent patterns without candidate generation: A frequent-pattern tree approach. *Data mining and knowledge discovery*, 8:53–87, 2004.

[55] Xixian Han, Bailing Wang, Jianzhong Li, and Hong Gao. Ranking the big sky: efficient top-k skyline computation on massive data. *Knowledge and Information Systems*, 60(1):415–446, 2019.

[56] Amin Hashemi, Mohammad Bagher Dowlatshahi, and Hossein Nezamabadi-pour. An efficient pareto-based feature selection algorithm for multi-label classification. *Information Sciences*, 581:428–447, 2021.

[57] Stefan Holland, Martin Ester, and Werner Kießling. Preference mining: A novel approach on mining user preferences for personalized applications. In *PKDD*, pages 204–216. Springer, 2003.

[58] Yuqi Huang, Dominique H Li, Haoyi Niu, and Donatello Conte. Visual identification of oscillatory two-phase flow with complex flow patterns. *Measurement*, 186:110148, 2021.

[59] Thorsten Joachims. Optimizing search engines using clickthrough data. In *SIGKDD*, pages 133–142, 2002.

[60] Werner Kießling and Gerhard Köstler. Preference sql—design, implementation, experiences. In *VLDB*, pages 990–1001. Elsevier, 2002.

[61] Jia-Ling Koh, Chia-Ching Chen, Chih-Yu Chan, and Arbee LP Chen. Mapreduce skyline query processing with partitioning and distributed dominance tests. *Information Sciences*, 375:114–137, 2017.

[62] Donald Kossmann, Frank Ramsak, and Steffen Rost. Shooting stars in the sky: An online algorithm for skyline queries. In *VLDB*, pages 275–286, 2002.

[63] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. *Advances in neural information processing systems*, 25:1097–1105, 2012.

[64] Jongwuk Lee and Seung-won Hwang. Bskytree: scalable skyline computation using a balanced pivot selection. In *EDBT*, pages 195–206, 2010.

[65] Jongwuk Lee and Seung-won Hwang. Scalable skyline computation using a balanced pivot selection technique. *Information Systems*, 39:1–21, 2014.

[66] Jongwuk Lee and Seung-won Hwang. Toward efficient multidimensional subspace skyline computation. *The VLDB Journal*, 23(1):129–145, 2014.

[67] Ken CK Lee, Wang-Chien Lee, Baihua Zheng, Huajing Li, and Yuan Tian. Z-sky: an efficient skyline query processing framework based on z-order. *The VLDB Journal*, 19(3):333–362, 2010.

[68] Thomas Leung and Jitendra Malik. Representing and recognizing the visual appearance of materials using three-dimensional textons. *International Journal of Computer Vision*, 43(1):29–44, 2001.

[69] Dominique H Li. Subset approach to efficient skyline computation. In *EDBT*, pages 391–403, 2023.

[70] Jinyan Li, Haiquan Li, Limsoon Wong, Jian Pei, and Guozhu Dong. Minimum description length principle: Generators are preferable to closed patterns. In *AAAI*, pages 409–414, 2006.

[71] Wenmin Li, Jiawei Han, and Jian Pei. CMAR: Accurate and efficient classification based on multiple class-association rules. In *ICDM*, pages 369–376, 2001.

[72] Xuemin Lin, Hongjun Lu, Jian Xu, and Jeffrey Xu Yu. Continuously maintaining quantile summaries of the most recent n elements over a data stream. In *ICDE*, pages 362–373, 2004.

[73] Xuemin Lin, Yidong Yuan, Wei Wang, and Hongjun Lu. Stabbing the sky: Efficient skyline computation over sliding windows. In *ICDE*, pages 502–513, 2005.

[74] Bin Liu and Chee-Yong Chan. Zinc: Efficient indexing for skyline computation. *VLDB Endowment*, 4(3):197–207, 2010.

[75] Bing Liu, Wynne Hsu, Yiming Ma, et al. Integrating classification and association rule mining. In *SIGKDD*, volume 98, pages 80–86, 1998.

[76] Rui Liu and Dominique Li. Dynamic dimension indexing for efficient skyline maintenance on data streams. In *DASFAA*, pages 272–287, 2020.

[77] Rui Liu and Dominique H Li. Efficient skyline computation in high-dimensionality domains. In *EDBT*, pages 459–462, 2020.

[78] Tianshu Liu. Openopticalflow: an open source program for extraction of velocity fields from flow visualization images. *Journal of Open Research Software*, 5(1), 2017.

[79] David Lo, Hong Cheng, Jiawei Han, Siau-Cheng Khoo, and Chengnian Sun. Classification of software behaviors for failure detection: A discriminative pattern mining approach. In *SIGKDD*, page 557–566, 2009.

[80] Zongqing Lu, Weixin Xie, Jihong Pei, and JianJun Huang. Dynamic texture recognition by spatio-temporal multiresolution histograms. In *IEEE Workshops on Applications of Computer Vision*, volume 2, pages 241–246, 2005.

[81] Bruce D. Lucas and Takeo Kanade. An iterative image registration technique with an application to stereo vision. In *IJCAI*, pages 674–679, 1981.

[82] Jizu Lv, Peng Wang, Minli Bai, Gang Li, and Ke Zeng. Experimental visualization of gas–liquid two-phase flow during reciprocating motion. *Applied Thermal Engineering*, 79:63–73, 2015.

[83] Jian Ma, Zhi-Ping Fan, Yan-Ping Jiang, Ji-Ye Mao, and Louis Ma. A method for repairing the inconsistency of fuzzy preference relations. *Fuzzy Sets and Systems*, 157(1):20–33, 2006.

[84] Florent Masseglia, Fabienne Cathala, and Pascal Poncelet. The psp approach for mining sequential patterns. In *PKDD*, pages 176–184, 1998.

[85] Michael Morse, Jignesh M Patel, and William I Grosky. Efficient continuous skyline computation. *Information Sciences*, 177(17):3411–3437, 2007.

[86] Michael Morse, Jignesh M Patel, and Hosagrahar V Jagadish. Efficient skyline computation over low-cardinality domains. In *VLDB*, pages 267–278, 2007.

[87] Kyriakos Mouratidis, Keming Li, and Bo Tang. Marrying top-k with skyline queries: Relaxing the preference input while producing output of controllable size. In *Proceedings of the 2021 International Conference on Management of Data*, pages 1317–1330, 2021.

[88] Kasper Mullesgaard, Jens Laurits Pederseny, Hua Lu, and Yongluan Zhou. Efficient skyline computation in mapreduce. In *EDBT*, pages 37–48, 2014.

[89] Benjamin Negrevergne, Alexandre Termier, Jean-François Méhaut, and Takeaki Uno. Discovering closed frequent itemsets on multicore: Parallelizing computations and optimizing memory accesses. In *HPCS*, pages 521–528, 2010.

[90] Benjamin Negrevergne, Alexandre Termier, Marie-Christine Rousset, and Jean-François Méhaut. Paraminer: a generic pattern mining algorithm for multi-core architectures. *Data Mining and Knowledge Discovery*, 28:593–633, 2014.

[91] Randal C Nelson and Ramprasad Polana. Qualitative recognition of motion using temporal texture. *CVGIP: Image understanding*, 56(1):78–89, 1992.

[92] Luong Phat Nguyen, Julien Mille, Dominique Li, Donatello Conte, and Nicolas Ragot. Trajectory extraction and deep features for classification of liquid-gas flow under the context of forced oscillation. In *VISAPP*, pages 17–26, 2020.

[93] Luong Phat Nguyen, Julien Mille, Dominique H Li, Donatello Conte, and Nicolas Ragot. Efficient dynamic texture classification with probabilistic motifs. In *2022 26th International Conference on Pattern Recognition (ICPR)*, pages 564–570. IEEE, 2022.

[94] Thanh Tuan Nguyen, Thanh Phuong Nguyen, Frédéric Bouchara, and Ngoc Son Vu. Volumes of Blurred-Invariant Gaussians for Dynamic Texture Classification. *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, 11678 LNCS:155–167, 2019.

[95] Thanh Tuan Nguyen, Thanh Phuong Nguyen, Frédéric Bouchara, and Xuan Son Nguyen. Momental directional patterns for dynamic texture recognition. *Computer Vision and Image Understanding*, 194:102882, 2020.

[96] F. Nielsen. K-mle: A fast algorithm for learning statistical mixture models. In *ICASSP*, pages 869–872, 2012.

[97] S. Nowozin, G. Bakir, and K. Tsuda. Discriminative subsequence mining for action classification. In *ICCV*, pages 1–8, 2007.

[98] S. Nowozin, K. Tsuda, T. Uno, T. Kudo, and G. Bakir. Weighted substructure mining for image analysis. In *CVPR*, pages 1–8, 2007.

[99] Dimitris Papadias, Yufei Tao, Greg Fu, and Bernhard Seeger. An optimal and progressive algorithm for skyline queries. In *SIGMOD*, pages 467–478, 2003.

[100] Dimitris Papadias, Yufei Tao, Greg Fu, and Bernhard Seeger. Progressive skyline computation in database systems. *ACM Transactions on Database Systems*, 30(1):41–82, 2005.

[101] Yoonjae Park, Jun-Ki Min, and Kyuseok Shim. Efficient processing of skyline queries using mapreduce. *IEEE Transactions on Knowledge and Data Engineering*, 29(5):1031–1044, 2017.

[102] Kostas Patroumpas and Timos Sellis. Window specification over data streams. In *EDBT*, pages 445–464, 2006.

[103] Jian Pei, Wen Jin, Martin Ester, and Yufei Tao. Catching the best views of skyline: A semantic approach based on decisive subspaces. In *VLDB*, pages 253–264, 2005.

[104] Jian Pei, Yidong Yuan, Xuemin Lin, Wen Jin, Martin Ester, Qing Liu, Wei Wang, Yufei Tao, Jeffrey Xu Yu, and Qing Zhang. Towards multidimensional subspace skyline analysis. *ACM Transactions on Database Systems*, 31(4):1335–1381, 2006.

[105] Verónika Peralta, Dimitre Kostadinov, and Mokrane Bouzeghoub. APMD-workbench: A benchmark for query personalization. In *CIRSE Workshop*, pages 38–41, 2009.

[106] Fabíola SF Pereira and Sandra de Amo. Evaluation of conditional preference queries. *Journal of Information and Data Management*, 1(3):503–503, 2010.

[107] Renaud Péteri and Dmitry Chetverikov. Dynamic texture recognition using normal flow and texture regularity. In *Iberian Conference on Pattern Recognition and Image Analysis*, pages 223–230, 2005.

[108] Gwangbum Pyun, Unil Yun, and Keun Ho Ryu. Efficient frequent pattern mining based on linear prefix tree. *Knowledge-Based Systems*, 55:125–139, 2014.

[109] Xianbiao Qi, Chun-Guang Li, Guoying Zhao, Xiaopeng Hong, and Matti Pietikäinen. Dynamic texture and scene classification by transferring deep image features. *Neurocomputing*, 171:1230–1241, 2016.

[110] Lucas C Ribas, Wesley N Goncalves, and Odemir M Bruno. Dynamic texture analysis with diffusion in networks. *Digital Signal Processing*, 92:109–126, 2019.

[111] Marco Tulio Ribeiro, Sameer Singh, and Carlos Guestrin. "why should i trust you?" explaining the predictions of any classifier. In *SIGKDD*, pages 1135–1144, 2016.

[112] Cheng Sheng and Yufei Tao. Worst-case i/o-efficient skyline algorithms. *ACM Transactions on Database Systems*, 37(4):1–22, 2012.

[113] Jianbo Shi and Carlo Tomasi. Good features to track. In *CVPR*, pages 593–600, 1994.

[114] Josef Sivic and Andrew Zisserman. Video google: A text retrieval approach to object matching in videos. In *ICCV*, volume 3, pages 1470–1470, 2003.

[115] S. Soatto, G. Doretto, and Ying Nian Wu. Dynamic textures. In *ICCV*, pages 439–446, 2001.

[116] Ruihua Song, Qingwei Guo, Ruochi Zhang, Guomao Xin, Ji-Rong Wen, Yong Yu, and Hsiao-Wuen Hon. Select-the-best-ones: A new way to judge relative relevance. *Information processing & management*, 47(1):37–52, 2011.

[117] Guanghui Su, Dounan Jia, Kenji Fukuda, and Yujun GUO. Theoretical study on density wave oscillation of two-phase natural circulation under low quality conditions. *Journal of Nuclear Science and Technology*, 38(8):607–613, 2001.

[118] Kian-Lee Tan, Pin-Kwang Eng, Beng Chin Ooi, et al. Efficient progressive skyline computation. In *VLDB*, volume 1, pages 301–310, 2001.

[119] Syed Khairuzzaman Tanbeer, Chowdhury Farhan Ahmed, Byeong-Soo Jeong, and Young-Koo Lee. Cp-tree: a tree structure for single-pass frequent pattern mining. In *PAKDD*, pages 1022–1027, 2008.

[120] Yufei Tao and Dimitris Papadias. Maintaining sliding window skylines on data streams. *IEEE Transactions on Knowledge and Data Engineering*, 18(3):377–391, 2006.

[121] Yufei Tao, Xiaokui Xiao, and Jian Pei. Subsky: Efficient computation of skylines in subspaces. In *ICDE*, pages 65–65, 2006.

[122] Yufei Tao, Xiaokui Xiao, and Jian Pei. Efficient skyline and top-k retrieval in subspaces. *IEEE Transactions on Knowledge and Data Engineering*, 19(8):1072–1088, 2007.

[123] Deepshikha Tiwari and Vipin Tyagi. Dynamic texture recognition based on completed volume local binary pattern. *Multidimensional Systems and Signal Processing*, 27(2):563–575, 2016.

[124] Deepshikha Tiwari and Vipin Tyagi. A novel scheme based on local binary pattern for dynamic texture recognition. *Computer Vision and Image Understanding*, 150:58 – 65, 2016.

[125] Deepshikha Tiwari and Vipin Tyagi. Dynamic texture recognition using multiresolution edge-weighted local structure pattern. *Computers & Electrical Engineering*, 62:485 – 498, 2017.

[126] Takeaki Uno, Masashi Kiyomi, Hiroki Arimura, et al. Lcm ver. 2: Efficient mining algorithms for frequent/closed/maximal itemsets. In *FIMI Workshop*, 2004.

[127] Akrivi Vlachou and Michalis Vazirgiannis. Ranking the sky: Discovering the importance of skyline points through subspace dominance relationships. *Data & Knowledge Engineering*, 69(9):943–964, 2010.

[128] Chen Wang, Mingsheng Hong, Jian Pei, Haofeng Zhou, Wei Wang, and Baile Shi. Efficient pattern-growth methods for frequent tree pattern mining. In *PAKDD*, pages 441–451, 2004.

[129] Chunyu Wang, John Flynn, Yizhou Wang, and Alan Yuille. Recognizing actions in 3d using action-snippets and activated simplices. In *Thirtieth AAAI Conference on Artificial Intelligence*, 2016.

[130] Chunyu Wang, Yizhou Wang, and Alan L Yuille. Mining 3d key-pose-motifs for action recognition. In *CVPR*, pages 2639–2647, 2016.

[131] Jianyong Wang and Jiawei Han. Bide: Efficient mining of frequent closed sequences. In *ICDE*, pages 79–90, 2004.

[132] Peng Wang, Jizu Lv, Minli Bai, Gang Li, and Ke Zeng. The reciprocating motion characteristics of nanofluid inside the piston cooling gallery. *Powder Technology*, 274:402–417, 2015.

[133] Wenlu Wang, Ji Zhang, M-T Sun, and W-S Ku. Efficient parallel spatial skyline evaluation using mapreduce. In *EDBT*, 2017.

[134] Yong Wang and Shiqiang Hu. Chaotic features for dynamic textures recognition. *Soft Computing*, 20(5):1977–1989, 2016.

[135] Heri Wijayanto, Wenlu Wang, Wei-Shinn Ku, and Arbee LP Chen. Lshape partitioning: Parallel skyline query processing using mapreduce. *IEEE Transactions on Knowledge and Data Engineering*, 34(7):3363–3376, 2020.

[136] Nic Wilson. Extending cp-nets with stronger conditional preference statements. In *AAAI*, pages 735–741, 2004.

[137] Douglas R. Woodall. Properties of preferential election rules. *Voting matters*, 3:8–15, 1994.

[138] Zhengzheng Xing, Jian Pei, and Eamonn Keogh. A brief survey on sequence classification. *ACM SIGKDD Explorations Newsletter*, 12(1):40–48, 2010.

[139] Y. Xu, Yuhui Quan, H. Ling, and H. Ji. Dynamic texture classification using dynamic fractal analysis. In *ICCV*, pages 1219–1226, 2011.

[140] Yong Xu, Sibin Huang, Hui Ji, and Cornelia Fermüller. Scale-space texture description on sift-like textons. *Computer Vision and Image Understanding*, 116(9):999 – 1013, 2012.

[141] Xifeng Yan and Jiawei Han. gspan: Graph-based substructure pattern mining. In *ICDM*, pages 721–724, 2002.

[142] Xifeng Yan, Jiawei Han, and Ramin Afshar. Clospan: Mining: Closed sequential patterns in large datasets. In *SDM*, pages 166–177, 2003.

[143] Man Lung Yiu and Nikos Mamoulis. Multi-dimensional top-k dominating queries. *The VLDB Journal*, 18(3):695–718, 2009.

[144] Xiaoli Yu, Dong Yi, Yuqi Huang, Yiji Lu, and Anthony Paul Roskilly. Experimental investigation of two-phase flow and heat transfer performance in a cooling gallery under forced oscillation. *International Journal of Heat and Mass Transfer*, 132:1306–1318, 2019.

[145] J. Yuan, Y. Wu, and M. Yang. Discovery of collocation patterns: from visual words to visual phrases. In *CVPR*, pages 1–8, 2007.

[146] J. Yuan, M. Yang, and Y. Wu. Mining discriminative co-occurrence patterns for visual recognition. In *CVPR*, pages 2777–2784, 2011.

[147] Mohammed J Zaki. Spade: An efficient algorithm for mining frequent sequences. *Machine learning*, 42:31–60, 2001.

[148] Mohammed Javeed Zaki. Scalable algorithms for association mining. *IEEE transactions on knowledge and data engineering*, 12(3):372–390, 2000.

[149] Shiming Zhang, Nikos Mamoulis, and David W Cheung. Scalable skyline computation using object-based space partitioning. In *SIGMOD*, pages 483–494, 2009.

[150] G. Zhao and M. Pietikainen. Dynamic texture recognition using local binary patterns with an application to facial expressions. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 29(6):915–928, 2007.

[151] Guoying Zhao and Matti Pietikäinen. Dynamic texture recognition using volume local binary patterns. In *Dynamical Vision*, pages 165–177, 2007.

[152] Feida Zhu, Xifeng Yan, Jiawei Han, and Philip S Yu. gprune: a constraint pushing framework for graph pattern mining. In *PAKDD*, pages 388–400, 2007.

Résumé

Introduction

La découverte efficace d'informations intéressantes à partir de données est un problème important dans de nombreux domaines d'application et de recherche. Avec le développement rapide des travaux de recherche en science des données et en apprentissage automatique, de nombreuses méthodes ont été proposées. Mes principaux intérêts de recherche portent sur les applications et les algorithmes de la fouille de motifs pour l'apprentissage automatique et sur le problème de traitement des requêtes skyline.

J'ai commencé à travailler activement sur la fouille de données dès le début de son stage de Master, en particulier sur la fouille de motifs séquentiels et ses applications. Après avoir obtenu mon doctorat en 2009 à l'Université de Montpellier, sur le thème de l'extraction de motifs séquentiels inattendus sous la direction du Professeur Pascal Poncelet et du Professeur Anne Laurent, j'ai effectué un post-doc d'un an à l'Université de Pise en Italie sous la direction du Professeur Giuseppe Attardi sur l'application des motifs à l'apprentissage actif dans l'analyse des dépendances en TAL. Depuis que je suis devenu Maître de Conférences à l'Université de Tours en 2010, j'ai intégré l'équipe de recherche BdTln au sein du Laboratoire LIFAT (ancien Laboratoire LI) de l'Université de Tours. Mes intérêts de recherche portent particulièrement sur les applications et les algorithmes de la fouille de motifs, de l'apprentissage automatique basé sur la fouille de motifs (depuis 2018) et le traitement des requêtes skyline (depuis 2019). Tous les résultats présentés dans ce mémoire d'habilitation sont postérieurs à 2010.

En 2010, dans le cadre du projet collaboratif international PQUERY avec l'Universida de Federal de Uberlândia (Brésil), l'Universidad de la República (Uruguay) et l'Université de Tours au sein du programme régional STIC-AmSud, j'ai commencé ma première participation à la recherche sur la fouille de motifs fréquents. L'objectif de ce projet de recherche est de générer automatiquement des profils d'utilisateurs à partir de données relationnelles sous forme de règles interprétables, où l'exploration de motifs fréquents est utilisée pour trouver des règles d'association

afin de déterminer conditionnellement des règles de préférence. Avec ce projet, Mouhamadou Saliou Diallo, (soutenu en 2014) est le premier doctorant que j'ai encadré avec mes collèges Arnaud Giacometti et Arnaud Soulet dans la même équipe de recherche BdTln sur le thème de la construction de profils utilisateurs par extraction et classement des motifs fréquents comme règles de préférence.

En effet, la fouille de motifs fréquents est un problème bien étudié depuis près de 30 ans après l'introduction des algorithmes `Apriori` pour l'extraction de *motifs ensemblistes* (*itemset patterns*) fréquents dans les données transactionnelles [1] telles que `GSP` [4], `Eclat` [148], `LCM` [126], `FP-Tree` [54], `CP-Tree` [119], ou `LP-Tree` [108], et *motifs séquentiels* (*sequential patterns*) dans des *données séquentielles* [3], comme `PSP` [84] , ou `SPADE` [147], `PrefixSpan` [53]. La notion de motifs fréquents ne se limite pas aux motifs ensemblistes et aux motifs séquentiels. Par exemple, les définitions et les problèmes d'extraction de motifs d'arbres et de motifs de graphes sont également étudiés en profondeur [128, 5, 141, 152] (qui sont hors de mes champs de recherche). D'autre part, pour améliorer l'efficacité des algorithmes *en mémoire* ci-dessus pour l'extraction des motifs fréquents, il existe plusieurs directions :

- Concevoir des algorithmes plus efficaces tels que `Eclat`, `LCM`, `FP-Tree`, `PSP`, `SPADE` et `PrefixSpan`.
- Renvoyer un sous-ensemble de motifs fréquents, par exemple, les *motifs clos* concernés par `CloSpan` [142] et `BIDE` [131], ou les *générateurs de motifs* concernés par `Gr-growth` [70] et `FEAT` [43].
- Extraire des motifs fréquents en calcul parallèle avec des processeurs multicœurs, tels que `PLCM` [89] et `ParaMiner` [90].

De 2016 à 2018, j'ai rejoint l'encadrement du chercheur post-doc Khanh-Chuong Duong dans le projet de recherche régional GIRAFON (Grands graphes : interrogation, fouille et analyse) en collaboration avec le Laboratoire LIFO de l'Université d'Orléans. Sous notre encadrement, Khanh-Chuong Duong a pu développer deux nouveaux algorithmes basés sur MapReduce [37, 38] pour extraire des motifs ensemblistes fréquents dans de très grands volumes de données. Par ailleurs, depuis 2018, j'ai participé à l'encadrement du doctorant Lamine Diop (soutenu en 2021) avec Arnaud Giacometti, Arnaud Soulet, et Cheikh Talibouya Diop (Université Gaston Berger de Saint-Louis, Sénégal) dont le sujet est d'échantillonner des motifs séquentiels afin de réduire le temps de calcul et d'éviter de renvoyer un grand nombre de motifs. En fait, la méthode d'échantillonnage [33, 35] proposée par Lamine Diop a été adaptée pour extraire des motifs ensemblistes fréquents dans des données distribués [34, 36].

Depuis 2017, j'ai officiellement commencé à concentrer mes recherches sur les applications de la fouille de motifs dans les données de séquences spatio-temporelles à différents problèmes d'apprentissage automatique, en particulier dans l'apprentissage de vidéos. Tout d'abord, j'ai lancé une collaboration internationale avec l'équipe du Professeur Yuqi Huang de l'Université du Zhejiang (Chine) et mon collège Donatello Conte dans l'équipe de recherche RFAI du laboratoire LIFAT pour identifier des vidéos d'écoulement diphasique par des motifs de séquence et des méthodes d'apprentissage automatique [58], qui devrait être utile à l'application d'un financement international de la recherche entre la France et la Chine à l'avenir. Dans un second temps, j'ai participé à l'encadrement de Luong Phat Nguyen (soutenu en 2021) avec mes collèges Nicolas Ragot, Donatello Conte, et Julien Mille dans l'équipe de recherche RFAI sur l'extraction de trajectoire et sur l'utilisation de motifs séquentiels probabilistes pour la classification de vidéo en texture dynamique [92, 93]. Les résultats obtenus dans les travaux de recherche ci-dessus montrent que les techniques d'extraction de motifs basées sur des séquences spatio-temporelles peuvent être appliquées efficacement aux problèmes d'apprentissage vidéo. Par ailleurs, dans mon encadrement actuel (depuis 2022) du doctorant Thomas Kastner avec mon collège Hubert Cardot dans l'équipe de recherche RFAI du Laboratoire LIFAT, nous avons proposé une nouvelle forme de séquence pour simplifier la représentation des intervalles du temps dans les séquences pour l'apprentissage des anomalies dans de grands volumes de données.

De fin 2018 à mi 2019, j'ai supervisé le stage recherche de Master de l'étudiante international Erasmus Mundus Rui Liu sur le traitement des requêtes skyline [12], dont les résultats ont obtenu le prix du *Best Master Thesis Award Erasmus Mundus 2019 in Computer Science*. Dans le mémoire de Rui Liu, nous avons proposé un nouveau cadre `SDI` [77] pour le calcul efficace de skyline dans des données de grande dimension et nous avons développé avec succès l'algorithme `RSS` [76], le premier algorithme au-dessus de `SDI`, pour maintenir le skyline dans les données de flux. Avec un tel succès depuis 2020, outre la fouille de motifs et l'apprentissage automatique, mes intérêts de recherche se concentrent sur plusieurs problèmes de traitement des requêtes skyline, notamment les requêtes générales, les requêtes dynamiques, les requêtes sous-espace et les requêtes top-k. Mon résultat de recherche le plus récent est d'utiliser des techniques de sous-ensembles pour améliorer les algorithmes de calcul de skyline [69].

En fait, au cours des dernières années, j'ai profondément ressenti qu'il existe plusieurs similitudes entre les problèmes de la fouille de motifs et les problèmes de requête skyline :

- Renvoyez les résultats plus rapidement. Pour ce problème, de nombreux algorithmes efficaces de skyline (détaillés au Chapitre 2) ont été proposés au cours de plus de deux décodages, y compris ceux basés sur MapReduce pour le traitement des données à grande

échelle.

- Renvoyer des résultats moins mais utiles. En tant que motifs fermés et générateurs de motifs dans la fouille de données, les requêtes skyline de sous-espace [103, 121, 104, 66], les requêtes de skyline top-k [20, 19, 55, 87] et leur combinaison [122, 143, 127] sont toujours activement étudiés.

- Les applications de skyline à l'apprentissage automatique, qui est l'une des perspectives les plus importantes de mon travail futur.

Par conséquent, je considère que l'objectif final de la fouille de motifs et du traitement des requêtes Skyline peut être considéré comme la recherche de sous-ensembles intéressants dans les données, ces sous-ensembles pouvant être un ensemble de motifs (pour la fouille de motifs) ou un ensemble de tuplets (pour le traitement de requêtes skyline). De plus, le terme *intéressant* peut être objectif, par exemple vérifié par des méthodes d'apprentissage automatique, ou subjectif, par exemple simplement examiné par des experts du domaine.

Le reste de ce mémoire HDR est organisé comme suit. Dans le Chapitre 1, je présente mes supervisions et travaux de recherche avec mes collègues sur les algorithmes de fouille de motifs et les applications des motifs en apprentissage automatique. Mon encadrement et mes travaux de recherche sur les problèmes de calcul de skyline sont présentés dans le Chapitre 2. Enfin, je conclus et présente mes perspectives pour de futurs travaux de recherche.

Applications d'fouille de motifs

Dans cette section, je présente les applications de la fouille de motifs dans plusieurs problèmes d'apprentissage automatique. Pour clarifier la notion de la *Fouille de motifs* (*Pattern Mining*) dans cette section, nous utilisons les termes *motif ensembliste* (*itemset pattern*) et *motifs séquentiel* (*sequence pattern*) dans ce mémoire pour distinguer les définitions originales proposées dans [1] et [3], et qui permet d'unifier la notion de *fouille de motifs*.

Construction de profil Le sujet de l'amélioration des systèmes de bases de données avec des requêtes de préférences contextuelles a suscité beaucoup d'intérêt dans la communauté des bases de données sur la construction de *profils d'utilisateurs*, qui a une variété d'applications allant du commerce électronique aux moteurs de recherche personnalisés où la prédiction des préférences de l'utilisateur sont essentielles dans la conception du modèle de système. De plus, la prédiction des préférences de l'utilisateur peut être considérée comme un problème d'apprentissage.

Cependant, peu de travaux ont été consacrés jusqu'à présent au sujet de l'*élicitation des préférences*, c'est-à-dire dans la manière dont les préférences de l'utilisateur sont obtenues pour créer le profil. Nous présentons notre approche basée sur des règles `ProfMiner` qui consiste à extraire un profil utilisateur à partir d'un ensemble d'échantillons de préférences utilisateur. Dans notre cadre, un *profil* est spécifié par un ensemble de *règles de préférences contextuelles* vérifiant les propriétés de justesse et de concision [29, 44, 28], sous forme de $X \rightarrow I^+ \succ I^-$, dont X est un itemset qui représente le contexte, et I^+ et l'itemset préféré à l'itemset I^- sous ce contexte.

Nous supposons que nos données sont constituées de comparaisons par paires et ne discutons pas de la manière dont l'utilisateur a informé les choix, sachant que différentes stratégies peuvent être appliquées [116]. Notre approche suppose également que la comparaison des *paires d'objets* peut exprimer les préférences de l'utilisateur par rapport à des contextes particuliers, ainsi, *règles de préférence contextuelles* [2] peuvent être générées par des méthodes d'extraction de motifs à partir d'un *base de données de préférences*.

Par conséquent, l'extraction de la règle de préférence contextuelle peut être résolue par une exploration fréquente des modèles avec pré-traitement et post-traitement :

- *Par l'extraction de motifs ensemblistes* : dans ce cas, la préférence de l'utilisateur sur deux tuples peut être combinée à une transaction en ajoutant un préfixe à chaque élément pour distinguer les éléments communs (le contexte X), les éléments préférés (I^+) et les éléments non préférés (I^-), alors, les règles de préférence peuvent être générées.

- *Par l'extraction de motifs ensemblistes* : dans ce cas, la préférence de l'utilisateur sur deux tuples peut être représentée par une séquence

  $$\langle (\text{éléments communs})(\text{éléments préférés})(\text{éléments non préférés}) \rangle ,$$

  où un motif séquentiel est directement une règle de préférence.

Une fois qu'un ensemble de règle de préférence contextuelle a été généré, nous les trions d'abord par la *confiance* puis par le *support* concernés dans la fouille de règles d'association. Nous avons proposé plusieurs méthodes qui permettent de comparer deux tuples pour voir si un est préféré à l'autre par rapport au profil de l'utilisateur.

Un ensemble d'expériences sur une base de données réelle des préférences des utilisateurs concernant les films a montré l'efficacité de la méthode. Plus intéressant, notre approche est la première à construire un profil utilisateur lisible basé sur la notion de règles de préférence contextuelles. Au cours de la dernière décennie, les techniques d'extraction de préférences se sont révélées très utiles dans la recherche de bases de données concernant la personnalisation des

requêtes. Des modèles de préférences extraits d'un échantillon donné de préférences utilisateur peuvent être stockés dans des référentiels qui seront interrogés afin de personnaliser la réponse à une requête SQL proposée par l'utilisateur. Les techniques d'extraction de préférences peuvent également être très utiles dans le développement de systèmes de recommandation pour les applications de commerce électronique.

Identification des motifs visuels d'écoulement diphasique Avec l'équipe du Professeur Yuqi Huang de l'Université du Zhejiang, nous avons proposé une application d'apprentissage vidéo basée sur des séquences pour identifier et classer des motifs visuels d'écoulement diphasiques oscillatoires complexes dans le contexte de la galerie de refroidissement des moteurs à combustion interne. Note qu'il s'agit de la première approche d'apprentissage vidéo dans ce domaine.

Notre étude et nos résultats montrent que dans le contexte dynamique (vidéos), les méthodes d'apprentissage automatique peuvent identifier des motifs d'écoulement diphasiques complexes, et la précision de classification moyenne obtenue dans nos expériences montre que notre méthode présentée permet de trouver les différences entre différents motifs de flux, qui sont représentés par différents nombres de Reynolds (le nombre de Reynolds permet de généraliser et de mesurer les écoulements diphasiques dans une galerie de refroidissement). Parce que les conditions expérimentales de notre étude sont proches des conditions réelles de travail, la clarté des résultats de visualisation est en fait affectée. Par conséquent, les méthodes d'apprentissage automatique sont introduites pour analyser et classer les vidéos capturées, en recherchant la relation entre les nombres de Reynolds et l'état du flux biphasique interne, et pour fournir des données de vérification plus stables et fiables à analyser avec les résultats CFD (Computational Fluid Dynamics).

Basée sur des techniques de traitement d'image et d'apprentissage automatique qui permettent de convaincre les résultats de visualisation d'un écoulement diphasique oscillatoire avec des modèles d'écoulement complexes, la méthode présentée peut être résumée en 3 étapes :

1. Extraire des motifs d'écoulement biphasés à partir de points de flux optique tracés de segments vidéo.

2. Représenter les motifs d'écoulement diphasiques comme un ensemble de séquences de décalages discrétisés sur l'axe X/Y 2D.

3. Caractériser les motifs d'écoulement à deux phases par des vecteurs de caractéristiques constitués de n-grammes (les n-grammes peuvent être considérer comme les motifs séquentiels avec les éléments consécutifs) générés à partir de séquences décalées.

Nous considérons que les descripteurs de motifs de flux à deux phases peuvent être caractérisés par les caractéristiques utilisées par les approches de classification vidéo, car ces caractéristiques permettent de classer les données vidéo et leurs performances peuvent être efficacement évaluées par la précision des résultats de classification. Noter que bien que les réseaux neuronaux profonds, tels que, généralement, les réseaux neuronaux convolutifs (CNN) puissent accomplir avec précision les tâches de classification vidéo, les itérations à l'intérieur du réseau multicouche rendent impossible l'exploitation des fonctionnalités apprises dans d'autres applications, telles que CFD dans le contexte de notre recherche.

La chaîne de traitement des données de notre méthode est représentée sur la Figure 1.8. Tout d'abord, à l'étape (1), un clip vidéo capturé est converti en une séquence d'images ; puis, à l'étape (2) suivante, selon la méthode développée par [113], on extrait par détection de flux optique un ensemble de points mobiles qui représentent les mouvements des interfaces de phase ; à l'étape (3), nous représentons les points mobiles tracés par des décalages consécutifs sur l'axe 2D X/Y, ainsi, de telles séquences de décalage décrivent des schémas d'écoulement à deux phases ; enfin, nous extrayons les caractéristiques évaluées par classification en tant que descripteurs de modèles d'écoulement à deux phases à l'étape (4). La qualité des descripteurs de modèles d'écoulement à deux phases extraits est mesurée par une validation croisée N fois de la classification vidéo, comme le montre la Figure 1.10. Traditionnellement, chaque clip vidéo est divisé en N segments indépendants sans chevauchement, dont $N-1$ segments sont utilisés comme ensemble d'apprentissage et 1 segment est utilisé comme ensemble de test pour chaque pli. Les descripteurs de modèle de flux pour la formation et le test du classificateur sont discrétisés uniquement sur la base des données de formation (qui seront détaillées dans la section suivante) à l'étape (1), puis à l'étape (2), nous construisons des vecteurs de caractéristiques par segment afin que toute classification des méthodes peuvent être utilisées pour évaluer des descripteurs de modèle de flux comme aux étapes (3) et (4).

Les valeurs contiguës contenues dans les séquences décalées empêchent la collecte de valeurs communes, ce qui est une nécessité pour présenter des séquences décalées. Une manière générale de discrétiser les attributs vectoriels numériques (c'est-à-dire les vecteurs de décalage) consiste à utiliser une méthode de clustering pour générer des attributs symboliques par rapport à des contraintes prédéfinies (par exemple, comme k-`means`, le nombre d'attributs) donc qu'un nombre limité d'attributs d'entités peut couvrir toutes les valeurs contiguës.

Il n'y a pas de différence définitive entre les résultats obtenus par les méthodes k-`means` et Agglomerative, par conséquent, nous avons finalement adopté k-`means` pour convertir les vecteurs de décalage en attributs de décalage discrétisés k car de sa simplicité et de sa large utilisation.

Par conversion basée sur k-`means`, faire en sorte qu'une séquence décalée discrétisée en séquence symbolique génère des caractéristiques communes peut être extraite.

Nous proposons la stratégie suivante pour classer les données vidéo représentées par un ensemble de séquences décalées. Soit $\mathcal{C}$ l'ensemble de toutes les classes, S_k^n un ensemble de séquences par rapport aux deux paramètres k et n, h un classifieur tel que pour chaque séquence $s \in S_k^n$, $h(s) = C$ où $C \in \mathcal{C}$ est une classe. Étant donné un classificateur h, on définit le *score niveau séquence* d'une classe C par rapport à un ensemble de séquences S_k^n, noté $\sigma_h(C, S_k^n)$, comme le nombre de toutes les séquences $s \in S_k^n$ classées dans C, c'est-à-dire

$$\sigma_h(C, S_k^n) = \frac{|\{s \in S_k^n \mid h(s) = C\}|}{|S_k^n|} \times 100.$$

Une classe C est appelée la *classe dominante* d'un ensemble de séquences S_k^n par rapport à un classificateur h si pour toutes les autres classes $C' \in \mathcal{C}$ nous avons $\sigma_h(C, S_k^n) > \sigma_h(C', S_k^n)$; sinon on dit qu'il n'y a pas de classe dominante donc l'ensemble de séquences S_k^n ne peut pas être classifié par le classifieur h. Dans le contexte typique de la classification, un score est une précision au niveau de la séquence d'une classe, qui peut être calculée à partir de la matrice de confusion, et la tâche de calcul consiste à trouver les meilleures valeurs (k, n) par paires qui maximisent le nombre de vidéos correctement classées.

Dans notre approche, les méthodes de vision par ordinateur et d'apprentissage automatique ont été appliquées de manière innovante à l'identification des motifs de flux diphasiques oscillants en fournissant une analyse plus précieuse pour les résultats des expériences de visualisation. Notre classification a évalué les résultats expérimentaux a montré la performance du travail présenté, ce qui peut aider à obtenir une tendance de mouvement plus claire de l'écoulement diphasique oscillatoire et fournir une vérification pour la simulation CFD. Les vecteurs obtenus en combinant des images avec la méthode du flux optique pourraient être pris comme référence pour comparer avec les vecteurs vitesses issues de la simulation numérique. La méthode fournit une nouvelle stratégie pour la détection de champ d'écoulement diphasique sans contact et peut analyser en profondeur et creuser les résultats de visualisation de l'écoulement diphasique oscillatoire avec des modèles d'écoulement complexes avec une fiabilité et une valeur d'application élevées. La méthode de recherche introduite peut être largement appliquée dans la visualisation de l'écoulement *multiphase* qui est un domaine clé à développer sur la recherche fondamentale des systèmes de transfert de chaleur. Dans nos travaux futurs, nous nous intéressons à la compréhension et à la prédiction de schémas d'écoulement diphasiques complexes par des modèles probabilistes.

Classification des textures dynamiques Nous proposons d'aborder la classification vidéo de texture dynamique comme un problème d'extraction de motifs de séquences. En effet, les

textures dynamiques sont des motifs spatio-temporels répétitifs caractérisés par des mouvements non rigides et complexes [115]. L'extraction de caractéristiques et de motifs spatio-temporels peut permettre la caractérisation et la classification de vidéos de texture dynamique par des approches d'apprentissage en profondeur.

Les premières méthodes spatio-temporelles artisanales basées sur les caractéristiques dépendent du flux optique [91, 107, 80, 27]. Par exemple, dans le travail de [27], une carte de mouvement/pas de mouvement est construite et combinée avec des modèles statistiques à états mixtes. Cependant, les méthodes basées sur le flux optique, qui se prêtent à l'extraction de champs de mouvement lisse, et peuvent donc ne pas bien représenter les textures dynamiques, qui sont généralement constituées de mouvements chaotiques dans plusieurs directions. L'apparence évolutive des textures dans le temps a été explicitement modélisée dans [115], en s'appuyant sur un système dynamique linéaire (LDS). La méthode utilise les paramètres du modèle comme entités en entrée. Une autre famille bien connue de caractéristiques artisanales sont les extensions spatio-temporelles de la méthode Local Binary Pattern (LBP), telles que CVLBP [123] ou 2D LBP calculées avec 3 plans orthogonaux (LBP-TOP) [150]. [94] propose une combinaison de filtres gaussiens et de modèles LBP, où les descripteurs LBP sont calculés à la fois à partir de volumes flous et de la différence 3D des volumes gaussiens. Récemment, une fonctionnalité appelée motif directionnel momentané (MDP) [95] a été développée.

Outre les approches d'apprentissage automatique, l'exploration de modèles, en particulier les méthodes d'extraction de motifs séquentiels, a également été utilisée pour résoudre divers problèmes de vision par ordinateur, tels que la classification d'images [98, 145, 146] ou la reconnaissance d'action [45, 97, 130] . Les méthodes présentées utilisent une affectation stricte des observations d'entrée aux éléments et aux modèles, ce qui peut conduire à des modèles instables et à des décisions sensibles qui ne sont pas adaptées à la nature floue des textures dynamiques. Afin d'éviter de tels problèmes, [130] propose un cadre pour la reconnaissance d'action où les modèles sont des séquences d'affectations souples, plutôt que des ensembles d'éléments eux-mêmes.

Nous proposons une nouvelle représentation des données de texture dynamique, avec la classification vidéo comme objectif final, basée sur l'extraction de motifs séquentiels. Le cadre proposé utilise un algorithme d'extraction de motifs séquentiels pour découvrir des sous-séquences fréquentes de patchs d'image, qui sont appelés motifs clés, inspiré de [130].

Dans la phase d'apprentissage, tout d'abord, un ensemble de symboles, correspondant à des patchs d'image représentatifs, est extrait par regroupement non supervisé comme principaux modes de distribution des patchs. Ensuite, pour chaque classe vidéo, un ensemble de motifs clés est construit : cette extraction traite à la fois des séquences déterministes (hard-assigned) et des

séquences probabilistes (soft-assigned). Enfin, les ensembles de motifs clés des différentes classes sont fusionnés en un ensemble global, dans lequel chaque classe est équitablement représentée. Dans la phase d'inférence, des séquences probabilistes soft-assignées sont extraites de la vidéo à classer. La correspondance entre ces séquences probabilistes et les motifs clés de l'ensemble unifié crée un vecteur de caractéristiques envoyé au classificateur final. L'analyse expérimentale étudie l'utilisation de cette représentation basée sur des motifs pour la classification vidéo avec une étude approfondie des paramètres, en tenant compte de l'impact sur le nombre de motifs clés et de l'impact sur les performances de classification. Nous soulignons le fait que la méthode proposée ne suit pas la tendance des réseaux de neurones convolutifs, et plus généralement des approches d'apprentissage profond.

Considérons la tâche de classification vidéo dans les classes C, nous introduisons un cadre efficace basé sur une nouvelle méthode d'exploration de *p-séquences*, qui sont des séquences de vecteurs de probabilité. La méthode proposée se décompose en trois étapes principales :

- Regrouper non-supervisé de patchs à l'aide d'un modèle de mélange gaussien, et construction de p-séquences.
- Extraire de motifs clés à partir de p-séquences pour chaque classe par extraction de motifs de séquence.
- Construire de vecteurs de caractéristiques basés sur des supports probabilistes de l'union des motifs clés et de la classification à l'aide d'un SVM avec un noyau χ^2.

Notre méthode proposée a été testée sur les jeux de données de référence **Traffic** et **UCLA** tout en la comparant aux méthodes de pointe. En général, les performances de notre approche proposée sont comparables aux méthodes de l'état de l'art. Comme la précision globale des expériences, différents ensembles de paramètres sont testés afin de mesurer l'impact de chaque paramètre pour chaque module. Les deux paramètres les plus importants dans notre approche sont la taille du patch σ et le nombre de clusters K. Une taille de patch plus petite de 8 avec un plus grand nombre de clusters de 50 affiche les meilleures performances. De plus, une contrainte d'écart maximum a également contribué à la performance de l'approche. Un écart plus petit génère moins de motifs qu'un écart plus grand (une contrainte d'écart maximum de 3 et 5) mais avec une précision légèrement meilleure, ce qui signifie que l'ensemble des motifs clés extraits utilisant un écart de 3 est plus significatif et plus discriminant que avec un écart de 5. En ce qui concerne le seuil de support minimum (directement lié au nombre de motifs), augmenter progressivement cette valeur peut nous aider à atteindre une valeur optimale et à obtenir le meilleur résultat.

Perspectives Dans le sujet de cette partie, ma perspective principale dans la fouille de motifs est centrée sur les applications des modèles aux problèmes d'apprentissage automatique en tant que *fonctionnalités efficaces* avec *apprentissage explicable* [111, 51, 7]. Bien que les motifs fréquents soient souvent utilisés comme fonctionnalités dans les problèmes d'apprentissage automatique supervisés ou non supervisés [75, 39, 35, 58, 93], cependant, avec la variation du seuil de support minimum, le nombre de motifs fréquents renvoyés (y compris modèles fermés ou générateurs de modèles) est incontrôlable : un seuil trop élevé rend le processus de fouille renvoyant trop peu de motifs et un seuil trop bas provoque trop de motifs, les deux cas ont de graves impacts pour construire un modèle d'apprentissage efficace mais petit. Par conséquent, la fouille et/ou la sélection de motifs *intéressants* ne sont pas moins importants que la fouille de motifs plus rapide ou moins rapide.

Pour atteindre la perspective ci-dessus, je suis intéressé par l'utilisation d'apprentissage explicables telles que `LIME` [111] pour construire un système de mesure d'intérêt objectif pour l'extraction de motifs itérative basée sur les préfixes sans interactions humaines. À l'inverse, ces modèles intéressants sont capables d'expliquer comment la décision d'apprentissage a été prise par les motifs. Cette perspective est également coopérée avec l'équipe du professeur Yuqi Huang de l'Université de Zhejiang (Chine) en génie énergétique et avec l'équipe du chercheur Dr. Jisheng Zhao de l'Université de New South Wales (Australie) en mécanique des fluides pour résoudre les problèmes d'apprentissage reliés.

D'autre part, mes perspectives en minage de motifs couvrent également le minage de motifs de séquences fréquentes dans le contexte du Big Data, avec le modèle de programmation MapReduce [30] ou Apache Spark[13] cadre. Il y a une recherche en cours en collaboration avec Mostafa Bamha au Laboratoire LIFO de l'Université d'Orléans. L'idée de base est d'étudier l'utilisation et le coût des bases de données de séquences projetées par préfixe dans un environnement distribué, typiquement en HDFS (Hadoop Distributed File System in the Apache Hadoop Framework[14]).

Vers le traitement efficace des requêtes skyline

Introduction Le *problème de traitement des requêtes skyline* a reçu une attention intensive de la part de la communauté des bases de données depuis la première introduction de l'*opérateur skyline* [12] en 2001.

Étant donné un ensemble de points multidimensionnels, la *requête skyline* renvoie le *skyline* qui est l'ensemble de tous les points non dominés. Un point p_i est dit *non-dominé* s'il n'existe

[13] `https://spark.apache.org`

[14] `https://hadoop.apache.org`

aucun autre point p_j tel que p_j soit meilleur que p_i dans toutes les dimensions par rapport à un ordre de préférence défini par l'utilisateur. La figure 2.1 montre un exemple de skyline très couramment utilisé dans la littérature : supposons un ensemble d'hôtels où nous voulons sélectionner ceux dont le prix (axe Y) et la distance de la plage (axe X) sont minimisés), puis les hôtels $\{a, c, e, h, m\}$ forment la ligne de skyline car aucun autre hôtel ne peut être meilleur qu'eux en termes de prix et de distance de la plage.

La façon la plus simple de calculer le skyline est la comparaison par paires basée sur des boucles imbriquées : pour chaque point p_i dans l'ensemble de données, comparez p_i avec chaque autre point p_j, si p_i domine p_j, puis supprimez p_j ; si p_j domine p_i, alors supprimez le point p_i et cassez la boucle imbriquée ; sinon, continuez la boucle imbriquée en conservant à la fois p_i et p_j. Une telle procédure de boucle imbriquée produit finalement l'ensemble de tous les points non dominés en un temps $\mathcal{O}(dN^2)$ où d est le nombre de dimensions en chaque point et N est le nombre de points dans les données, si l'on considère $\mathcal{O}(d)$ temps pour tester la relation de dominance (*dominance test*) entre deux points de dimensions d.

Les tests de dominance représentent le coût majeur du calcul de skyline. Afin de résoudre efficacement le problème de calcul de skyline, de nombreux algorithmes ont été conçus et développés sur la base de la réduction des testicules de dominance, qui peuvent être classés en deux classes [65] : basé sur le tri (tel que `BNL` [12], `Index` [118], `SFS` [24, 25], `LESS` [46, 47], `SaLSa` [9, 10], `BSkyTree-S` [64, 65]) et basé sur le partitionnement (comme `D&C` [12], `NN` [62], `BBS` [99, 100], `LS` [86], `OSPS` [149], `ZSearch` [67] et `BSkyTree-P` [64, 65]). En outre, des techniques d'indexation sont également appliquées aux algorithmes de skyline, tels que `Index`, `BBS` et `ZINC` [74] avec différents mécanismes.

Le défi que nous avons relevé dans le cadre du stage Erasmus Mundus BDMA International Master de Rui Liu, supervisée par moi-même, est de développer un nouveau framework évolutif et extensible `SDI` pouvant être utilisé dans différents problèmes de skyline, en général le calcul de skyline en données statiques [77] ou dans les flux de données [76], et peut être étendu aux requêtes de sous-espace, aux requêtes top-k, et pour résoudre tous ces problèmes de skyline dans le Big Data, comme décrit dans la perspective. Avec les résultats présentés, le mémoire de Master de Rui Liu a été sélectionnée comme meilleur prix Erasmus Mundus 2019 en informatique. Dans ce travail, détaillé dans [77] et [76], nous avons d'abord proposé un framework basé sur le tri et l'indexation `SDI` pour un calcul efficace de skyline dans des domaines de grande dimension, puis nous avons étendu avec succès ce framework à flux de données.

En étudiant les algorithmes de skyline existants, j'ai proposé indépendamment une nouvelle approche basée sur sous-ensemble pour améliorer le calcul de skyline basé sur le tri, détaillé

dans [69], qui fait de SDI et SaLSa les algorithmes de skyline les plus rapides sur des données indépendantes uniformes .

SDI : un cadre évolutif pour le calcul skyline Nous présentons un algorithme basé sur l'indexation des dimensions pour le calcul de skyline. Nous montrons d'abord que les tests de dominance nécessaires à la détermination d'un point de skyline peuvent être suffisamment bornés à un sous-ensemble de skyline courant, puis proposons l'algorithme SDI (Scalable Dimension Indexing), dont la complexité temporelle est mieux que le meilleur algorithme connu dans les domaines de haute dimensionnalité avec une cardinalité raisonnablement faible. Notre évaluation des performances sur des ensembles de données synthétiques et réels montre l'efficacité de SDI dans les domaines de haute dimensionnalité ainsi que dans les domaines de faible dimensionnalité.

En indexant toutes les dimensions, il suffit de tester un point uniquement avec les points de skyline existants sur une dimension arbitraire au lieu de l'ensemble complet des points de skyline. Nous montrons également que tout point de skyline peut être utilisé comme une *stop line* qui traverse les dimensions indexées pour arrêter le calcul, ce qui est beaucoup plus performant que le calcul de *stop point* mentionné dans SaLSa [9, 10]. De plus, SDI adopte la *vérification d'incomparabilité faible* pour prendre en compte l'incomparabilité entre les points, qui est la caractéristique la plus importante des derniers algorithmes de skyline. Notre analyse montre que la pire complexité temporelle de SDI est meilleure que la plus connue [112] dans les domaines à haute dimensionnalité avec une cardinalité raisonnablement faible, et notre évaluation des performances montre que SDI surpasse l'état de Algorithme BSkyTree à la pointe de la technologie[15] sur des données de grande dimension, mais moins efficace que BSkyTree sur des données de faible dimension.

En effet, nous pouvons proposer plusieurs extensions de SDI : (1) SDI calcule le skyline dans les domaines catégoriels tant que l'ordre de préférence $\prec$ peut être défini ; (2) SDI peut être immédiatement adapté au calcul de skyline du sous-espace en sautant des dimensions non liées ; (3) SDI peut être étendu à la maintenance de skyline en construisant dynamiquement l'index de dimension ; (4) SDI peut gérer la requête skyline de top-k en trouvant les points skyline ayant les meilleures positions dans l'index de dimension.

Le principal avantage de l'algorithme SDI est que : (1) le calcul de skyline peut être effectué sur un indice dimensionnel arbitraire ; (2) n'importe quel point de skyline peut être utilisé pour arrêter le processus de calcul en produisant skyline complet. Cependant, une remarque importante est qu'avec l'augmentation de la cardinalité des données, par exemple à 1M ou 10M

[15] L'algorithme BSkyTree que nous avons comparé est bien l'algorithme BSkyTree-S inclus dans le code source de SkyBench [22] : https://github.com/sean-chester/SkyBench

points, SDI a besoin de beaucoup de temps pour construire l'index de dimension, ce qui ralentira clairement le temps de traitement global. En effet, nous considérons que pour les requêtes skyline à un tour, se concentrer sur le temps de traitement total est beaucoup plus important que de se concentrer uniquement sur le temps de requête sans regarder le temps de construction de la structure de données. Par conséquent, dans notre perspective particulière, nous sommes intéressés à faire de SDI un cadre évolutif pour différents problèmes d'horizon par rapport au paradigme : **construire une fois, interroger à tout moment et de toute façon**. Les principaux intérêts de ce paradigme incluent la maintenance de skyline dans les données dynamiques (par exemple, les flux de données ou la mise à jour des données) et la requête d'horizon dynamique dans les données statiques.

Maintenir skyline dans les flux de données Le problème de la *maintenance de skyline* dans les flux de données est très difficile car il nécessite des mises à jour instantanées de skyline concernant l'arrivée de nouvelles données et l'expiration de données trop précoces qui sont probablement sans valeur et doivent être supprimées en raison de la sensibilité temporelle des données ruisseaux. Nous présentons l'algorithme RSS (**R**ange **S**earch for **S**tream-Skyline Maintenance) pour la maintenance de skyline dans les flux de données multidimensionnels. Notre analyse montre que la complexité temporelle de RSS est limitée par un sous-ensemble de skyline instantané, et notre évaluation montre l'efficacité de RSS sur les flux de données de faible et de grande dimension. Notons que RSS est la première extension de l'algorithme SDI.

En tant que technique la plus utile dans le traitement des données en continu, une *fenêtre glissante* est généralement considérée lors de la formulation de requêtes basées sur le flux [102]. Le modèle de fenêtre est généralement *count-based* qui couvre un certain nombre de points les plus récents à chaque instant, ou *time-based* qui est limité par un certain nombre d'unités de temps coïncidant avec les horodatages du flux. La figure 2.7 montre un paradigme de maintenance de skyline basée sur une fenêtre glissante sur l'illustration précédente de skyline *prix-distance* (Figure 2.1).

En fait, les mises à jour dynamiques des données rendent la maintenance de skyline difficile sur les flux de données : (1) *un point skyline expiré peut libérer des points dominés afin qu'ils deviennent des points skyline* ; (2) *un point entrant domine les points skyline existants de sorte que ces derniers doivent être supprimés de skyline.* La plupart des algorithmes skyline existants conçus pour les données statiques ne répondent pas aux exigences imposées par les données en continu. Par exemple, en ce qui concerne chaque mise à jour de skyline invoquée par les points entrants/expirés par rapport à une fenêtre glissante, l'algorithme simple BNL nécessite un balayage complet de la fenêtre, les algorithmes basés sur le tri nécessitent un nouveau tri

de tous les points de la fenêtre, et les algorithmes basés sur le partitionnement nécessitent la reconstruction de la structure de partitionnement. Par conséquent, la mise à jour dynamique de skyline et les structures de données efficaces sont deux problèmes importants de la maintenance de skyline sur les flux de données [72, 73, 85, 120].

Nos principales contributions incluent :

- Nous construisons une structure de tri liée pour maintenir un index de dimension dynamique qui sert la fenêtre glissante.
- Nous montrons qu'avec la méthode d'indexation de dimension dynamique proposée, les tests de dominance impliqués par les opérations (1) et (2) ci-dessus sont limités à des sous-ensembles optimaux de points skyline actuels dans la fenêtre glissante.
- Nous évaluons notre algorithme proposé avec une fenêtre glissante basée sur le nombre et sur le temps, sur des ensembles de données synthétiques et réels, et montrons son efficacité sur des flux de données de faible et de haute dimension.

Nos résultats expérimentaux, qui sont détaillés dans l'article complet [76], montrent que le temps de traitement moyen est affecté par la suppression du premier point expiré s'il s'agit d'un point skyline, alors le temps de traitement par nouveau point est de 10^{-4} à 10^{-5} secondes en moyenne. Bien qu'il révèle que le temps de traitement varie principalement entre 10^{-4}s et 10^{-5}s, lorsqu'un point d'expiration est un point skyline, il doit vérifier un grand nombre de points non-skyline, ce qui entraîne en pic haut. Une autre observation courante est que la performance atteint une stabilité ou se maintient dans une amplitude en ce qui concerne l'incrément de la dimensionnalité. Notre évaluation confirme notre analyse théorique. En tant que travail futur possible, nous nous intéressons à l'adaptation de `RSS` et `SDI` aux flux de données distribués et aux flux de données basés sur des colonnes.

Accélérer le calcul skyline par une approche de sous-ensemble Nous présentons une approche basée sur les sous-ensembles qui permet d'intégrer l'incomparabilité basée sur les sous-espaces aux algorithmes existants de skyline basés sur le tri et peut donc réduire considérablement le nombre total de tests de dominance dans de grands ensembles de données multidimensionnels. Nos études théoriques et expérimentales montrent que l'approche de sous-ensemble proposée renforce les algorithmes existants de skyline basés sur le tri et les rend comparables aux algorithmes de pointe et encore plus rapides avec des données indépendantes uniformes.

Godfrey et al. [46, 47] et Sheng et Tao [112] ont analysé théoriquement la complexité temporelle des algorithmes existants de skyline et ont conclu que dans le pire des cas, les algorithmes

basés sur le tri se terminent en $\mathcal{O}(dN^2)$ temps , cependant les algorithmes basés sur le partitionnement peuvent se terminer en $\mathcal{O}(N \log^{d-2} N)$ temps pour toute dimensionnalité $d > 2$. Godfrey et al. [46, 47] a également donné une analyse de la complexité temporelle pour le cas moyen sous les conditions d'*indépendance uniforme* et d'*indépendance des composants*. Essentiellement, l'efficacité d'un algorithme de skyline dépend fortement de la façon dont les tests de dominance sont réduits. Comme les algorithmes skyline de l'état de l'art, `BSkyTree-S` et `BSkyTree-P` utilisent un schéma de sélection de points pivots pour mapper des points de données à des régions incomparables, chacun peut être considéré comme un algorithme optimisé de tri et de partitionnement.

Ainsi, nous présentons une nouvelle approche d'indexation de skyline basée sur le sous-espace pour gérer l'incomparabilité entre les points de test et les points skyline. Au lieu de partitionner le jeu de données, notre méthode indexe les points skyline en dominant les sous-espaces pour réduire les tests de dominance. Nous montrons d'abord que le sous-espace dominant d'un point peut être fusionné par rapport à plusieurs *pivot skyline points*, appelés *maximum dominating subspace* par rapport à un nombre donné de points pivots, puis, nous montrons que si un point q_1 domine un point q_2, les sous-espaces dominants maximaux de q_1 par rapport à chaque point de pivot doivent être un sur-ensemble des sous-espaces dominants maximaux de q_2. Sur la base des propriétés présentées dans cette section, l'esquisse de l'application de notre méthode peut être décrite comme suit :

1. Trouver plusieurs points pivots skyline pour générer le sous-espace dominant maximal pour chaque point non élagué.
2. Exécuter un algorithme skyline avec les nouvelles actions suivantes :
   (a) Une fois qu'un point skyline est déterminé, placer-le dans la structure d'index de skyline proposée par rapport à son sous-espace dominant maximum.
   (b) Lors du test d'un point avec le skyline actuel, n'obtenir que l'ensemble des points comparables de la structure d'index de skyline par rapport au sous-espace dominant maximum.
3. Renvoyer le skyline.

Par conséquent, notre méthode proposée ne concerne pas les algorithmes concrets de calcul d'horizon car elle est conçue comme un composant comme un conteneur qui permet de stocker (en tant que fonction *put*) les points skyline et de récupérer (en tant que fonction *get*) un nombre minimum de points skyline à comparer avec un point de test. Le paradigme de notre

méthode correspond aux meilleurs algorithmes skyline basés sur le tri qui peuvent produire progressivement les points skyline, mais les algorithmes skyline basés sur le partitionnement ne peuvent pas beaucoup bénéficier de notre méthode car les données ont déjà été partitionnées et les tests de dominance sont limités dans les partitions, le double le partitionnement des données et de skyline entraîne des coûts supplémentaires. Par conséquent, l'avantage de notre méthode est de booster les algorithmes de skyline existants. Plusieurs algorithmes skyline basés sur le tri tels que `SFS`, `SaLSa` et `SDI` ne dépendent d'aucune structure de données particulière (par exemple, le treillis et SkyTree requis par `BSkyTree-S` et `BSkyTree-P` , l'arbre ZB requis par `ZINC`, l'arbre R requis par `BBS` et l'arbre B^+ requis par `Index`, etc.), qui peuvent être facilement implémentés dans n'importe quel des langages de programmation comme `Python` et `PHP`, donc booster de tels algorithmes a des intérêts réalistes pour les industries de données.

Notre résultat principal présenté est que l'indexation skyline peut être efficacement résolue par une **subset query**. Le *subset query problem* est défini comme : étant donné un ensemble $\mathcal{X}$ de sous-ensembles distincts d'un univers D, pour tout ensemble $Q \subseteq D$, retourne l'ensemble $\{Q' \in \mathcal{X} \mid Q \subset Q'\}$ qui contient tous les sur-ensembles de Q. L'ensemble Q est appelé *ensemble de requêtes* (*query set*). En effet, étant donné un point de test, son sous-espace dominant maximal peut être considéré comme un ensemble de requêtes, la tâche consiste à renvoyer tous les points skyline dont les sous-espaces dominants maximaux sont les sur-ensembles de l'ensemble de requêtes, par conséquent, les tests de dominance requis peuvent être limités dans les points d'horizon renvoyés. Afin de rendre notre méthode efficace, nous avons inversé le problème de la requête de sous-ensemble, c'est-à-dire pour retourner l'ensemble $\{Q' \in \mathcal{X} \mid Q' \subset Q\}$. La structure de données de requête de sous-ensemble et les algorithmes proposés sont basés sur une map de hachage (prise en charge par la plupart des langages de programmation), qui peuvent ajouter un point skyline en temps linéaire par rapport à la dimensionnalité d et peuvent récupérer un ensemble de points skyline par rapport à un sous-espace donné en $O((\frac{d}{2})^2)$ temps dans le cas moyen pour tout $d > 2$. En particulier, dans le cas où $d = 2$, la requête de sous-ensemble est un problème binaire donc l'utilité de notre méthode proposée est très limitée.

Nos résultats expérimentaux montrent que notre méthode peut significativement booster `SFS`, `SaLSa` et `SDI`. En résumé, nous présentons une approche de sous-ensemble pour un calcul efficace de skyline, qui est conçue pour stimuler les algorithmes basés sur le tri. Nous avons proposé une méthode d'union de sous-espaces pour attribuer à tous les points un sous-espace dominant maximum, avec lequel les tests de dominance entre les points skyline et les points de test ne sont nécessaires qu'entre des sous-espaces comparables. Afin de récupérer efficacement les points skyline pour les tests de dominance par rapport aux sous-espaces dominants maximaux, nous

avons proposé une méthode de requête de sous-ensemble pour indexer le skyline. Notre analyse théorique et nos résultats expérimentaux montrent que le calcul de skyline peut être amélioré par notre méthode d'union de sous-espace et de requête de sous-ensemble. La perspective particulière du travail présenté comprend : (1) l'extension de la méthode proposée aux données AC et CO ; (2) développer un modèle de coût pour améliorer le seuil de stabilité afin de trouver le meilleur nombre de points pivots ; (3) adapter le procédé proposé à la mise à jour de données telles que des flux de données.

Perspectives Les problèmes liés au traitement des requêtes de skyline sont très similaires aux problèmes d'extraction de motifs, nous sommes confrontés à : (1) comment calculer le skyline plus rapidement, pour lequel de nombreux algorithmes ont été développés ; (2) comment retourner moins de résultats, pour lesquels la requête skycube et sous-espace est bien étudiée ; (3) comment renvoyer des résultats plus utiles, pour lesquels la mesure la plus objective est les requêtes top-k, basées sur la dominance ou sur le sous-espace. Notez que la plupart des algorithmes skyline sont basés sur le calcul en mémoire, par conséquent, dans le contexte du Big Data, il est difficile de calculer les différents skyline ci-dessus dans de grands ensembles de données, même quelques méthodes basées sur MapReduce existent [88, 61, 101, 133, 135].

J'ai de nombreuses perspectives dans le traitement des requêtes skyline, qui sont basées sur le framework `SDI` (Section 2.2) et mon approche `Subset` (Section 2.4):

- Rendre le calcul général de skyline plus rapide. L'idée principale est d'adapter mon approche `Subset` pour booster également les algorithmes skyline basés sur les partitions car en général, les algorithmes basés sur les partitions fonctionnent beaucoup plus rapidement que les algorithmes basés sur le tri, même nous avons montré que `SaLSa+Subset` et `SDI+Subset` surpassent les algorithmes de l'état de l'art sur les données de type UI.

- Rendre le calcul de skyline du sous-espace plus évolutif. Comme introduit dans l'introduction, la notion de Skycube a été proposée pour pré-calculer les skylines dans des sous-espaces basés sur un treillis, dont il y a $2^d - d - 1$ sous-espaces si nous ne considérons pas les skylines unidimensionnel (il suffit de trier les valeurs). Cependant, Skycube ne répond pas aux exigences du Big Data d'aujourd'hui : par exemple, étant donné un énorme ensemble de données de 32 dimensions, un cube de 2 294 967 263 nœuds devrait être créé pour stocker les skylines des sous-espaces. Je suis intéressé par la conception d'algorithmes skyline rapides de sous-espace sur-demande basés sur `SDI+Subset`, avec la fusion (bottom-up) ou la division (top-down) des résultats mis en cache.

- Rendre le calcul de skyline du sous-espace top-k plus évolutif et plus efficace. Comme mentionné précédemment, il existe actuellement deux définitions des skyline top-k : basé sur la dominance et basé sur le sous-espace. Nous avons montré qu'avec l'augmentation de la dimensionnalité des données, la taille de skyline augmente. Par exemple, alors que 90% des points sont des points skyline dans un jeu de données, le skyline de top-k basé sur la dominance n'a pas de sens car presque tous les points sont dans le skyline et ils ne dominent qu'une petite partie des points. Par conséquent, pour ma part, je pense que le calcul top-k skyline basé sur le sous-espace est un sujet avec des défis passionnants en plus de `SDI+Subset`.

- Concevoir des algorithmes skyline efficaces basés sur MapReduce pour d'énormes volumes de données de grande dimension stockés dans HDFS/HBase[16]/Hive[17] clusters), où le calcul en mémoire ne peut pas s'adapter au volume de données. Ce sujet est une recherche en cours avec Mostafa Bamha dans le laboratoire LIFO de l'Université d'Orléans, dont le premier résultat (à soumettre) montre que `SDI` peut être bien adapté pour calculer le skyline directement à partir de HBase. Les requêtes sous-espace et top-k skyline sont également intéressantes avec le modèle de programmation MapReduce.

- De plus, dans les grandes bases de données relationnelles SQL, le calcul en mémoire de skyline est irréaliste. De plus, avec la croissance de l'externalisation de l'analyse des données, la protection des données privées dans le calcul de skyline avec trois parties (la source de données, la puissance de calcul et l'analytique) est un sujet critique. Basée sur le framework `SDI`, une nouvelle méthode de calcul de skyline sans tests de dominance est en cours de développement (à soumettre), avec l'équipe du Professeur Yuqi Huang de l'Université de Zhejiang (Chine) et l'équipe du Professeur associé Min Bao de l'Université Sci-Tech du Zhejiang (Chine). Sur ce sujet, nous rechercherons un financement international entre la France et la Chine pour produire plus de résultats.

Les perspectives ci-dessus devraient donner de nouvelles solutions à d'importants problèmes à skyline, y compris les Big Data dans le calcul de la mémoire externe ou par des solutions tierces.

Conclusion et travaux futurs

Le travail présenté dans ce mémoire HDR présente l'évolution de mes intérêts de recherche. Depuis mon stage de Master, ma thèse, et mon post-doc, j'ai apporté des contributions sur le

[16] `https://hbase.apache.org`
[17] `https://hive.apache.org`

domaine traditionnel de la fouille de motifs, qui consiste à proposer de nouveaux algorithmes pour miner des motifs existants, en particulier des motifs séquentiels, des motifs séquentiels inattendus, et leurs applications aux problèmes d'apprentissage automatique. Depuis 2016, j'ai commencé à étudier la fouille de motifs avec MapReduce dans le Big Data, et depuis 2018 je me suis intéressé aux applications de motifs à différents problèmes dans différents domaines liés à l'apprentissage vidéo (Chapitre 1). Depuis 2019, mes recherches se sont concentrées sur les problèmes de traitement des requêtes skyline (Chapitre 2). Dans ce mémoire, j'ai donné des perspectives dans chacun de la fouille de motifs (chapitre 1, section 5) et du traitement des requêtes skyline (chapitre 2, section 5), la future direction de recherche que j'ai l'intention de suivre.

Sans aucun doute, je continuerai à appliquer la fouille de motifs aux problèmes d'apprentissage et à utiliser l'apprentissage pour améliorer la qualité de motifs, dans ou sans le contexte du Big Data. Ainsi, mes priorités dans la fouille de motifs sont les applications de modèles dans les domaines industriels et la fouille de motifs séquentiels dans des grands volumes de données avec les frameworks MapReduce et Spark.

Actuellement, ma plus haute priorité se concentre sur différents problèmes de traitement des requêtes skyline dans des données à grande échelle avec une cardinalité et une dimensionnalité élevée. `SDI` est un cadre évolutif pour le calcul efficace de skyline dans des domaines à haute dimensionnalité, avec lequel des requêtes de sous-espace, top-k et dynamiques peuvent être construites et peuvent être améliorées par l'approche de sous-ensemble. Une autre direction de recherche sur le calcul de skyline consiste à utiliser skyline pour servir l'apprentissage automatique, y compris la sélection de fonctionnalités et la construction de modèles. En effet, le skyline, ou le front de Pareto, a été étudiée dans le problème de sélection de caractéristiques [41, 56], mais les méthodes existantes reposent sur des domaines de faible dimensionnalité, principalement pour trouver des combinaisons de caractéristiques non dominées par rapport à l'efficacité de l'apprentissage . Avec le cadre de calcul skyline de grande dimension proposé, il est intéressant de trouver le skyline dans un grand nombre de descripteurs avec un mécanisme de classement tel que LIME, afin de générer directement l'ensemble de descripteurs optimié. De plus, je suis très intéressé par l'utilisation de skyline pour combiner de petits modèles à un modèle hybride pour un problème d'apprentissage concret, au lieu d'utiliser un grand modèle pour résoudre tous les problèmes d'apprentissage. Par exemple, dans un problème d'apprentissage concret, toutes les instances d'apprentissage (cardinalité) peuvent être évaluées par différents modèles (dimensionnalité), le calcul du sous-espace et du top-k skyline permet de filtrer la meilleure combinaison de modèles.