Finding Top-k Sequences over Data Streams according to Temporal Conditional Preferences

Marcos Roberto Ribeiro[1,2], **Maria Camila N. Barioni**[2], **Sandra de Amo**[2], **Claudia Roncancio**[3], **Cyril Labbé**[3]

[1] Instituto Federal de Minas Gerais (IFMG), Bambuí, Brazil

[2]Universidade Federal de Uberlândia (UFU), Uberlândia, Brazil

[3]Univ. Grenoble Alpes, CNRS, Grenoble INP, LIG, F-38000, Grenoble, France

marcos.ribeiro@ifmg.edu.br, {camila.barioni, deamo}@ufu.br, {claudia.roncancio, cyril.labbe}@imag.fr

Abstract. *Recently, several research works have been conducted on processing of preference queries over data streams. Preference queries are useful for many application domains where users aim to find out the closest data items to their wishes. This paper presents a new operator for the StreamPref language that can be employed to obtain the top-k data stream sequences according to temporal conditional preferences. Temporal conditional preferences can allow a user to express how past instants of a data stream may influence his preferences at a present instant. In order to evaluate this new operator, two new algorithm strategies are also presented. The extensive set of experiments performed show that the incremental strategy presents a superior performance in all experimental settings. Moreover, the results achieved show that the proposed operator has a superior performance when compared to the equivalent operation in CQL.*

1. Introduction

Preference queries aim to select the closest data items to the user wishes [Ribeiro et al. 2016]. Considering posing this type of query in data stream scenarios, preference queries must be processed efficiently to meet the high-speed data transfer requirement. There are several related research works concerned with skyline queries where the user preferences are represented as wishes for maximum or minimum attribute values [Börzsönyi et al. 2001, Lin et al. 2005, Tao and Papadias 2006]. However, there are application domains that require the users to express conditional preferences. This kind of preference allows the user to say how some attribute value influences their preferences over another attribute. In order to illustrate, let us consider a soccer coach who wants to hire a player based on his nationality. He can express his desire considering a conditional preference as follows: "if the player is Brazilian then I prefer the attack position than the midfield position".

The data tuples in data stream scenarios have an implicit temporal information. If we consider this rich information, it is possible to use conditional temporal preferences to express how a user wishes at a given moment are impacted by past attribute values. As an example, consider a coach who wants to monitor a data stream of a soccer match. The coach may use preferences such as "if the player was in offensive intermediary then I prefer that he stays in the same place instead of going to midfield". We have been exploring this research topic in some preliminary works. We proposed a new formalism for reasoning with temporal conditional preferences [Ribeiro et al. 2017a] and used this formalism to define the first version of the StreamPref query language which allows querying data streams with temporal conditional preferences [Ribeiro et al. 2017b].

The current version of the StreamPref query language is able to select dominant sequences. A sequence s is dominant if there is no other sequence better than s. However, if a query returns few dominant sequences, this result could not be enough for the user. The user may want to rank the sequences according to their preferences to get the best k sequences in this rank (i.e., the top-k sequences). For instance, the same soccer coach mentioned previously can be also interested in answering the following query: "Give me the best four sequences of positioning according to my preferences". Others interesting practical applications for queries with temporal preferences are stock market, telecommunications, web applications, sensor networks, among others. This paper presents a new operator that incorporates the ability to select the top-k sequences according to temporal conditional preferences in the StreamPref query language.

The main contributions of this paper can be summarized as follows: **(1)** We propose an extension of the StreamPref query language with a new operator that allows to find top-k sequences according to temporal conditional preferences; **(2)** We present the demonstration of the equivalence for the proposed operator and the existing operators; **(3)** We propose an efficient algorithm to evaluate the new operator; **(4)** We describe the results of an extensive set of experiments comparing two strategies used by our algorithm.

The remainder of this paper is organized as follows. Section 2 introduces the logical formalism and the existing operators of the StreamPref language. Section 3 presents our new proposed operator to extend the StreamPref language. Section 4 describes the algorithm used to evaluate the proposed operator. Section 5 presents the experiments and discusses the results. Section 6 discusses the main related research works. Finally, the conclusion and the future work directions are presented in Section 7.

2. The StreamPref Language

This section presents the fundamental concepts regarding the preference model and the operators of the StreamPref query language [Ribeiro et al. 2017a, Ribeiro et al. 2017b]. Section 2.1 describes the preference model and Section 2.2 presents the existing operators.

2.1. Temporal Conditional Preferences

Let $R(A_1, ..., A_l)$ be a relational schema. A sequence $s = \langle t_1, ..., t_n \rangle$ over R is an ordered set of tuples, such that $t_i \in \mathbf{Tup}(R)$ for all $i \in \{1, ..., n\}$ where $\mathbf{Tup}(R) = \mathbf{Dom}(A_1) \times ... \times \mathbf{Dom}(A_l)$ is the set of all tuples over R. The length of a sequence s is denoted by $|s|$. A tuple in the position i of a sequence s is denoted by $s[i]$ while the notation $s[i].A$ represents the attribute A in the position i of s. The set of all possible sequences over R is denoted by $\mathbf{Seq}(R)$. The StreamPref formulas are based on propositions $(A\theta a)$, where $a \in \mathbf{Dom}(A)$ and $\theta \in \{<, \leq, =, \neq, \geq, >\}$ (see Definition 1). Let $Q(A)$ be a proposition, $S_{Q(A)} = \{a \in \mathbf{Dom}(A) \mid a \models Q(A)\}$ denotes the set of values satisfying $Q(A)$.

Definition 1 (StreamPref Formulas) *The StreamPref formulas are defined as follows:* ***(1)*** $\mathbf{true}$ *and* $\mathbf{false}$ *are StreamPref formulas;* ***(2)*** *If F is a proposition then F is a Stream-*

*Pref formula; **(3)** If F and G are StreamPref formulas then $(F \wedge G)$, $(F \vee G)$, $(F$ **since** $G)$, $\neg F$ and $\neg G$ are StreamPref formulas.*

A StreamPref formula F is satisfied by a sequence $s = \langle t_1, ..., t_n \rangle$ at a position $i \in \{1, ..., n\}$, denoted by $(s, i) \models F$, according to the following conditions: ***(1)*** $(s, i) \models Q(A)$ if and only if $s[i].A \models Q(A)$; ***(2)*** $(s, i) \models F \wedge G$ if and only if $(s, i) \models F$ and $(s, i) \models G$; ***(3)*** $(s, i) \models F \vee G$ if and only if $(s, i) \models F$ or $(s, i) \models G$; ***(4)*** $(s, i) \models \neg F$ if and only if $(s, i) \not\models F$; ***(5)*** $(s, i) \models (F \textbf{ since } G)$ if and only if there exists j where $1 \le j < i$ and $(s, j) \models G$ and $(s, k) \models F$ for all $k \in \{j + 1, ..., i\}$. The $\textbf{true}$ formula is always satisfied and the $\textbf{false}$ formula is never satisfied. The StreamPref also has the following derived formulas:

$\textbf{Prev}\ Q(A)$**:** Equivalent to $(\textbf{false since } Q(A))$, $(s, i) \models \textbf{Prev}\ Q(A)$ if and only if $i > 1$ and $(s, i - 1) \models Q(A)$;

$\textbf{SomePrev}\ Q(A)$**:** Equivalent to $(\textbf{true since } Q(A))$, $(s, i) \models \textbf{SomePrev}\ Q(A)$ if and only if there exists j such that $1 \le j < i$ and $(s, j) \models Q(A)$;

$\textbf{AllPrev}\ Q(A)$**:** Equivalent to $\neg(\textbf{SomePrev}\neg Q(A))$, $(s, i) \models \textbf{AllPrev}\ Q(A)$ if and only if $(s, j) \models Q(A)$ for all $j \in \{1, ..., i - 1\}$;

$\textbf{First}$**:** Equivalent to $\neg(\textbf{Prev}(\textbf{true}))$, $(s, i) \models \textbf{First}$ if and only if $i = 1$.

Definition 2 formalizes the *temporal conditions* used by Definition 3 (*tcp-rules* and *tcp-theories*). Example 1 shows a practical application using these definitions.

Definition 2 (Temporal Conditions) *A temporal condition is a formula $F = F_1 \wedge ... \wedge F_n$, where $F_1, ..., F_n$ are propositions or derived formulas. The temporal components of F, denoted by $F^{\leftarrow}$, are the conjunction of all derived formulas in F. The non-temporal components of F, denoted by $F^{\bullet}$, is the conjunction of all propositions in F and not present in $F^{\leftarrow}$. The notation $\mathbf{Att}(F)$ represents the attributes appearing in F.*

Definition 3 (TCP-Rules and TCP-Theories) *Let R be a relational schema. A temporal conditional preference rule, or tcp-rule, is an expression in the format $\varphi : C_\varphi \rightarrow Q^+_\varphi \succ Q^-_\varphi[W_\varphi]$, where: **(1)** The propositions Q^+_φ and Q^-_φ, over the preference attribute A_φ, represent the preferred values and non-preferred values, respectively, such that $S_{Q^+_\varphi} \cap S_{Q^-_\varphi} = \{\}$; **(2)** $W_\varphi \subset R$ is the set of indifferent attributes such that $A_\varphi \notin W_\varphi$; **(3)** C_φ is a temporal condition such that $\mathbf{Att}(C^\bullet_\varphi) \cap \{A_\varphi\} \cap W_\varphi = \{\}$. A temporal conditional preference theory, or tcp-theory, is a finite set of tcp-rules.*

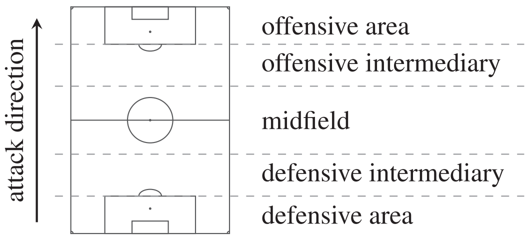

Figure 1. Soccer field places

Example 1 *Suppose a soccer coach who has access to an information system that provides real-time data concerning field positioning of the players. The data is in* *the stream* `positioning(pid, place, ball, direction)` *composed of the attributes* `pid` *(player identifier),* `place` *(field place),* `ball` *(ball possession) and* `direction` *(moving direction). The field places are depicted in Figure 1. The attribute* `ball` *has the value* 1 *when the team has the ball possession and* 0 *for otherwise. For the attribute* `direction`*, the possible values are forward (fw), lateral (la) and rewind (rw). The coach has the following preferences:* ***[P1]*** *Lateral moves are better than forward moves, independent of ball possession;* ***[P2]*** *Forward moves are better than rewind moves, independent of ball possession;* ***[P3]*** *If, in a given moment, the team does not have the ball possession and, immediately before this moment, the player was at offensive intermediary and, always before this moment, the team had the ball, then I prefer midfield place than offensive intermediary place;* ***[P4]*** *If, in a given moment, the team has the ball possession and, immediately before this moment, the player was at midfield then I prefer offensive intermediary place than midfield place; These preferences can be expressed using the tcp-theory* Φ *composed of the following tcp-rules:* $\varphi_1 : \rightarrow (\texttt{direction} = la) \succ (\texttt{direction} = fw)[\texttt{ball}]$*;* $\varphi_2 : \rightarrow (\texttt{direction} = fw) \succ (\texttt{direction} = rw)[\texttt{ball}]$*;* $\varphi_3 : (\texttt{ball} = 0) \wedge \mathbf{Prev}(\texttt{place} = oi) \wedge \mathbf{AllPrev}(\texttt{ball} = 1) \rightarrow (\texttt{place} = mf) \succ (\texttt{place} = oi)$*;* $\varphi_4 : (\texttt{ball} = 1) \wedge \mathbf{Prev}(\texttt{place} = mf) \rightarrow (\texttt{place} = oi) \succ (\texttt{place} = mf)$*.*

A sequence s is preferred to a sequence s' (or s dominates s') according to a φ, denoted by $s \succ_\varphi s'$, if and only if there exists a position i such that: ***(1)*** All positions before i must be identical in both sequences, $s[j] = s'[j]$ for all $j \in \{1, ..., i-1\}$; ***(2)*** The position i of s and s' must satisfy the rule condition C_φ, $(s, i) \models C_\varphi$ and $(s', i) \models C_\varphi$; ***(3)*** The position i of s has a preferred value and the position i of s' has a non-preferred value, $s[i].A_\varphi \models Q^+_\varphi$ and $s'[i].A_\varphi \models Q^-_\varphi$; ***(4)*** Excluding the preference attribute A_φ and the indifferent attributes of W_φ, all attributes of position i must have identical values in both sequences (*ceteris paribus* semantic), $s[i].A' = s'[i].A'$ for all $A' \notin (\{A_\varphi\} \cup W_\varphi)$. Example 2 presents a possible comparison of sequences.

Example 2 *Considering the sequences* $s = \langle (oi, 1, la), (oi, 1, fw),\ (oi, 0, fw) \rangle$ *and* $s' = \langle (oi, 1, la), (oi, 1, fw), (mf, 0, fw) \rangle$ *and the tcp-theory* Φ *of Example 1. It is possible to say that* $s \succ_{\varphi_3} s'$ *since:* ***(1)*** $s[1] = s'[1]$ *and* $s[2] = s'[2]$*;* ***(2)*** $(s, 2) \models (\texttt{ball} = 0) \wedge \mathbf{Prev}(\texttt{place} = oi) \wedge \mathbf{AllPrev}(\texttt{ball} = 1)$ *and* $(s', 2) \models (\texttt{ball} = 0) \wedge \mathbf{Prev}(\texttt{place} = oi) \wedge \mathbf{AllPrev}(\texttt{ball} = 1)$*;* ***(3)*** $s[3].\texttt{place} = mf$ *(preferred value),* $s'[3].\texttt{place} = oi$ *(non preferred value);* ***(4)*** $s[3].\texttt{ball} = s'[3].\texttt{ball}$ *and* $s[3].\texttt{direction} = s'[3].\texttt{direction}$ *(ceteris paribus semantic).*

The notation $\succ_\Phi$ represents the transitive closure of $\bigcup_{\varphi \in \Phi} \succ_\varphi$. Let Φ be a tcp-theory over a relational schema $\mathbf{Seq}(R)$. A sequence $s \in \mathbf{Seq}(R)$ is preferred to $s' \in \mathbf{Seq}(R)$ according to Φ, denoted by $s \succ_\Phi s'$, if there exists the sequences $s_1, ..., s_{m+1} \in \mathbf{Seq}(R)$ and the tcp-rules $\varphi_1, ..., \varphi_m \in \Phi$ such that $s_1 \succ_{\varphi_1} ... \succ_{\varphi_m} s_{m+1}$, where $s = s_1$ and $s' = s_{m+1}$. When two sequences cannot be compared, they are called incomparable. For instance, consider the sequences s and s' of Example 2 and the sequence $s'' = \langle (oi, 1, la), (oi, 1, fw), (mf, 1, la) \rangle$. We have the comparisons $s \succ_{\varphi_3} s'$ and $s' \succ_{\varphi_1} s''$. So, by transitivity, $s \succ_\Phi s''$. We must also consider consistency issues when dealing with order induced by rules to avoid inferences such as "a sequence is preferred to itself". Please, see [Ribeiro et al. 2017a] for more details about consistency issues.

2.2. StreamPref Operators

The first step in the evaluation of a continuous tcp-query is the extraction of sequences using $\mathbf{SEQ}$ operator. This task is performed by the operation $\mathbf{SEQ}_{X,n,d}(S)$, where X is the set of identifier attributes, n is the temporal range, d is the slide interval and S is the input data stream. The parameters n and d are used to delimit a portion of the data stream analogously to the selection performed by the *sliding window* operators [Arasu et al. 2016]. The parameter X is a key used to group the tuples with the same identifier in a sequence. Example 3 demonstrates the use of the $\mathbf{SEQ}$ operator.

Example 3 *Consider the stream `positioning (pid, place, ball, direction)` of Figure 2(a) and the preferences of Example 1. Now, suppose that a coach submits the following query to the information system: "**[Q1]** At every instant, give me the sequences of positioning that best fit my preferences over the last 3 seconds". The extraction of sequences is performed by the operation $\mathbf{SEQ}_{\{pid\},3,1}(positioning)$. Figure 2(b) shows, instant by instant, the result of this operation. As the user wants to consider just the last three seconds, from instant 3, the old tuples are removed from the sequences.*

Instant	`pid`	`place`	`ball`	`direction`
0	1	mf	1	la
0	2	oi	1	la
0	3	mf	1	fw
0	4	oi	1	la
0	5	oi	1	la
1	1	oi	0	la
1	2	oi	0	la
1	3	mf	0	la
1	4	mf	0	la
1	5	oi	0	fw
2	1	oi	1	la
2	2	oi	1	rw
2	3	di	1	la
2	4	di	1	rw
2	5	oi	1	rw
3	1	oi	0	rw
3	2	oi	0	rw
3	3	mf	0	la
3	4	oi	0	rw
3	5	mf	0	rw

(a)

Instant 0
$s_1 = \langle (mf, 1, la) \rangle$
$s_2 = \langle (oi, 1, la) \rangle$
$s_3 = \langle (mf, 1, fw) \rangle$
$s_4 = \langle (oi, 1, la) \rangle$
$s_5 = \langle (oi, 1, la) \rangle$

Instant 1
$s_1 = \langle (mf, 1, la), (oi, 0, la) \rangle$
$s_2 = \langle (oi, 1, la), (oi, 0, la) \rangle$
$s_3 = \langle (mf, 1, fw), (mf, 0, la) \rangle$
$s_4 = \langle (oi, 1, la), (mf, 0, la) \rangle$
$s_5 = \langle (oi, 1, la), (oi, 0, fw) \rangle$

Instant 2
$s_1 = \langle (mf, 1, la), (oi, 0, la), (oi, 1, la) \rangle$
$s_2 = \langle (oi, 1, la), (oi, 0, la), (oi, 1, rw) \rangle$
$s_3 = \langle (mf, 1, fw), (mf, 0, la), (di, 1, la) \rangle$
$s_4 = \langle (oi, 1, la), (mf, 0, la), (di, 1, rw) \rangle$
$s_5 = \langle (oi, 1, la), (oi, 0, fw), (oi, 1, rw) \rangle$

Instant 3
$s_1 = \langle \cancel{(mf, 1, la)}, (oi, 0, la), (oi, 1, la), (oi, 0, rw) \rangle$
$s_2 = \langle \cancel{(oi, 1, la)}, (oi, 0, la), (oi, 1, rw), (oi, 0, rw) \rangle$
$s_3 = \langle \cancel{(mf, 1, fw)}, (mf, 0, la), (di, 1, la), (mf, 0, la) \rangle$
$s_4 = \langle \cancel{(oi, 1, la)}, (mf, 0, la), (di, 1, rw), (oi, 0, rw) \rangle$
$s_5 = \langle \cancel{(oi, 1, la)}, (oi, 0, fw), (oi, 1, rw), (mf, 0, rw) \rangle$

(b)

Figure 2. (a) Stream `positioning` (b) Sequences extracted by SEQ operator.

Let Z be a set of sequences and Φ be a tcp-theory. The operation $\mathbf{BESTSEQ}_{\Phi}(Z)$ returns the *dominant* sequences in Z according to Φ. A sequence $s \in Z$ is dominant according to Φ, if $\nexists s' \in Z$ such that $s' \succ_{\Phi} s$. Example 4 shows how the $\mathbf{BESTSEQ}$ operator can be used to evaluate a query.

Example 4 *Let Z be the extracted sequences from Example 3 at instant 3. We can select the best sequences according to the tcp-theory in Example 1 by using the operation $\mathbf{BESTSEQ}_{\Phi}(\mathbf{SEQ}_{\{pid\},3,1}(positioning))$. At instant 3, $s_1 \succ_{\Phi} s_2$, $s_1 \succ_{\Phi} s_5$, $s_2 \succ_{\Phi} s_5$ and $s_3 \succ_{\Phi} s_4$. Thus, the result of query **Q1** at instant 3 is $\{s_1, s_3\}$.*

3. The New Operator TOPKSEQ

It is possible to use the $\mathbf{BESTSEQ}$ operator to obtain the top-k sequences. However, as we will see at the end of this section, the $\mathbf{BESTSEQ}$ operator is not suitable for this task. Thus, in this paper, we propose the $\mathbf{TOPKSEQ}$ operator in order to select the top-k sequences. The $\mathbf{TOPKSEQ}$ operator returns the top-k sequences of an input set of sequences Z according to a tcp-theory Φ. The top-k sequences are the sequences of Z with the lowest preference level (Definition 4)

Definition 4 (Preference level) *Let Φ be a tcp-theory. Let Z be a set of sequences. The preference level of a sequences s, denoted by $level(s)$, is:* ***(1)*** *If $\nexists s' \in Z$ such that $s' \succ_\Phi s$, then $level(s) = 0$;* ***(2)*** *Otherwise, $level(s) = \max\{level(s') \mid s' \in Z$* ***and*** *$s' \succ_\Phi s\} + 1$.*

Notice that the sequences with level zero are exactly those returned by the $\mathbf{BESTSEQ}$ operator. The $\mathbf{TOPKSEQ}$ operator is especially useful when the result of the $\mathbf{BESTSEQ}$ operator has few sequences. In this case, the $\mathbf{TOPKSEQ}$ operator can complement the answer using sequences with greater levels (see Example 5).

Example 5 *Consider again the Example 4. The query result has just two sequences. Now, suppose that the coach has the following query: "**[Q2]** At every instant, give me the best four sequences of positioning according to my preferences over the last 3 seconds". This query is evaluated by the operation $\mathbf{TOPKSEQ}_{\Phi,4}(\mathbf{SEQ}_{\{pid\},3,1}(\mathtt{positioning}))$. At instant 3, $s_1 \succ_\Phi s_2$, $s_1 \succ_\Phi s_5$, $s_2 \succ_\Phi s_5$ and $s_3 \succ_\Phi s_4$. So, the preference levels are $level(s_1) = 0$, $level(s_3) = 0$, $level(s_2) = \max\{level(s_1)\} + 1 = 1$, $level(s_4) = \max\{level(s_3)\} + 1 = 1$ and $level(s_5) = \max\{level(s_1), level(s_2)\} + 1 = 2$. Thus the result of query **Q2** at instant 3 is $\{s_1, s_3, s_2, s_4\}$.*

The StreamPref language is an extension of the Continuous Query Language (CQL). The StreamPref operators do not increase the expression power of the CQL [Ribeiro et al. 2017b]. However, the equivalences are not trivial since the StremPref operators is equivalent to complex operations using several CQL operators and intermediary relations. So, these equivalences are not simple to be written by the user. Moreover, the new operator has algorithms that are specially tailored to process queries more efficiently than their CQL counterparts. The CQL equivalences for the SEQ and $\mathbf{BESTSEQ}$ operators were already demonstrated in our previous work [Ribeiro et al. 2017b]. Equations (1a)-(1d) show how we can use the $\mathbf{BESTSEQ}$ operator to evaluate the $\mathbf{TOPKSEQ}$ operator. As the $\mathbf{BESTSEQ}$ operator has a CQL equivalence, we can conclude that the $\mathbf{TOPKSEQ}$ operator has also a CQL counterpart.

$$L_0 \leftarrow \mathbf{BESTSEQ}_\Phi(Z) \tag{1a}$$
$$L_1 \leftarrow \mathbf{BESTSEQ}_\Phi(Z - L_0) \tag{1b}$$
$$L_2 \leftarrow \mathbf{BESTSEQ}_\Phi(Z - L_1 - L_0) \tag{1c}$$
$$\vdots$$
$$L_m \leftarrow \mathbf{BESTSEQ}_\Phi(Z - L_{m-1} - \ldots - L_0) \tag{1d}$$

The term m of Equation (1d) represents the maximum preference level imposed by Φ. This number is equal to the number of tcp-rules of Φ in the worst case. Each set L_i contains the sequences with preference level i. The top-k sequences can be obtained by taking the sequences of these sets (following the preference level order) until k sequences are reached. Despite the TOPKSEQ operator can be evaluated using the BESTSEQ operator, it is necessary to process all sequences at every instant to reach all preference levels and sort the sequences by level. On the other hand, Section 4 presents algorithms which stop the processing after the top-k sequences are obtained.

4. The Algorithm

As discussed in the previous section, the **TOPKSEQ** operator can be processed by using the **BESTSEQ** operator a certain number of times. The algorithm *GetTopkSeq* (see Algorithm 1) employs this idea to evaluates the **TOPKSEQ** operator. First, the algorithm creates a list to keep the sequences ordered by their preference level. The first iteration of the loop uses the *GetBestSeq* routine to select the sequences with level zero. This routine basically removes the dominant sequences from Z. Every iteration of the loop selects the sequences of the next level. This process stops when the list has at least k sequences or Z is empty.

Algorithm 1: *GetTopkSeq*(Φ, k, Z)

```
1 $L \leftarrow$ NewList();
2 while $(|L| < k)$ and $Z \neq \{\}$ do
3     $Z' \leftarrow$ GetBestSeq$(\Phi, Z)$;
4     $L$.append$(Z')$;
5 return $L$.getFirst$(k)$;
```

Algorithm 2: *NaiveBestSeq*(Φ, Z)

```
1 $Z' \leftarrow Z$;
2 foreach $s, s' \in Z'$ do
3     if $s \succ_\Phi s'$ then $Z' \leftarrow Z' - \{s'\}$ ;
4     else if $s' \succ_\Phi s$ then $Z' \leftarrow Z' - \{s\}$ ;
5 return $Z'$;
```

The *GetBestSeq* routine is basically an algorithm to evaluate the **BESTSEQ** operator. This algorithm can use a naive approach [Ribeiro et al. 2017a] or an incremental approach [Ribeiro et al. 2017b]. The naive approach must compare all sequences at every instant as addressed by the algorithm *NaiveBestSeq* (see Algorithm 2). On the other hand, the incremental approach keeps an index structure updated using just the sequence changes. This index structure is a sequence tree created using the sequences tuples.

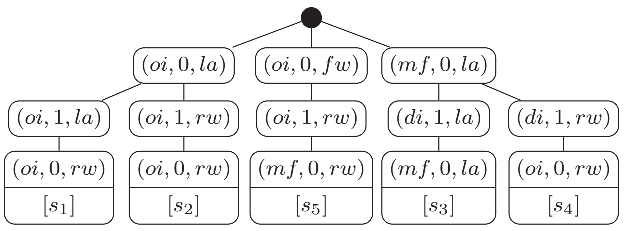

Figure 3. Sequence tree

Figure 3 shows the sequence tree built with the sequences shown in Example 3 at instant 3. Only the changed sequences are reallocated in the tree. The tree structure is useful to find the position where two sequences must be compared (the fork nodes). In addition, a node keeps a preference hierarchy representing the children comparison. So, for each tree node, we can obtain the dominant nodes (and, by consequence, the dominant sequences) by using this preference hierarchy. Please see [Ribeiro et al. 2017b] for more details about the index structure. The algorithm *IncBestSeq* (see Algorithm 3) obtains the dominant sequences by using the sequence tree. The algorithm starts at the tree root and uses recursive calls over the dominant children to reach all dominant sequences.

Algorithm 3: *IncBestSeq*(nd)

```
1 $Z \leftarrow nd.Z$;
2 foreach dominant child of $nd$ do
3     $Z \leftarrow Z \cup$ IncBestSeq$(child)$;
4 return $Z$;
```

The naive algorithm must find the position to be compared. In the incremental version, the sequence tree already points to the position to be compared. In addition, the tree nodes use preference hierarchies to store many comparisons of previous instants. Only changed sequences cause updates in the tree and preference hierarchy.

The complexity analysis of the algorithms takes into account the number of input sequences (z), the length of the largest sequence (n) and the number of tcp-rules in Φ (m). We assume a constant factor for the number of attributes. The algorithm *NaiveBestSeq* must compare every pair of sequences. The comparison start by looking for the first different position in the sequences. In the worst case, this position is the last one (n). Next, for the dominance test, the algorithm uses a deep first search strategy to find a chain of sequences and rules. The search tree of this strategy has height and node degree equal to m in the worst case. Thus, the complexity of the algorithm *NaiveBestSeq* is $O(z^2 \times (n + m^m))$ where the factor m^m is the cost of the dominance test.

The incremental strategy to obtain the dominant sequences must update the sequence tree. In the worst case scenario, the degree of nodes is $O(z)$ and the tree depth is $O(n)$. We also have to consider the cost to deal with the preference hierarchy. Our preference hierarchy uses the partition strategy described in [Ribeiro et al. 2016]. The update cost of this hierarchy is m^4. Thus, the complexity of *IncBestSeq* is $O(zn \times m^4)$ since every sequence can cause the update of n nodes.

The cost of the algorithm *GetTopkSeq* is related to the complexity and number of calls to the routine *GetBestSeq*. This routine is called $O(m)$ times in the worst case. Thus, the complexity of the algorithm *GetTopkSeq* is the cost of this routine multiplied by m. In data stream scenarios, the incremental algorithms usually are faster than naive algorithms. The experimental results of the next section show this tendency in the algorithms when processing the **TOPKSEQ** operator.

5. Experimental Results

We conducted an extensive set of experiments to analyze the performance (runtime) and the memory usage of the algorithms used to evaluate the **TOPKSEQ** operator. All experiments were carried out on a machine with a 3.2 GHz twelve-core processor and 32 GB of main memory, running Linux. The algorithms were implemented in a Data Stream Management System (DSMS) prototype using Python language[1].

The same tool used in [Ribeiro et al. 2017b] was employed to generate the synthetic datasets for our experiments[2]. This tool generates streams composed of integer attributes. In addition, it allows evaluating several parameter settings. For each experi-

[1]`http://streampref.github.io/`
[2]`http://streampref.github.io/streamprefgen/`

ment, we varied one parameter and fixed a default value for the others. We do not include the CQL equivalence in our experiments because this equivalence was already discussed in [Ribeiro et al. 2017b]. The experimental results described in [Ribeiro et al. 2017b] showed higher runtime and greater memory usage for the CQL equivalence due to its various intermediary operation and temporary relations.

Table 1. The parameters of the experiments: (a) Data generation; (b) Sequence extraction; (c) Preferences.

(a)

Param.	Variation
ATT	8, 10, **12**, 14, 16
NSQ	8, 16, **24**, 32, 40

(b)

Param.	Variation
RAN	20, 40, **60**, 80, 100
SLI	10, 20, **30**, 40, 50

(c)

Param.	Variation
RUL	8, 16, **24**, 32, 40
LEV	1, 2, **3**, 4, 5

Table 1 shows the variation of the parameters (with default values in bold). The number of attributes (ATT) allows for the evaluation of the algorithm behavior according to different data dimensionality. The number of sequences (NSQ) controls how the number of tuples per instant (equal to NSQ $\times$ 0.75) affects the algorithms. The temporal range (RAN) delimits the maximum length of the sequences and the slide interval (SLI) is related to the number of deletions when the sliding window moves.

The number of rules (RUL) and the maximum preference level (LEV) are employed in the generation of the preferences. These parameters allow us to evaluate how different preferences affect the algorithms. We used rules in the form φ_i : $\mathbf{First} \wedge Q(A_3) \rightarrow Q^+(A_2) \succ Q^-(A_2)[A_4, A_5]$ and φ_{i+1} : $\mathbf{Prev}Q(A_3) \wedge \mathbf{SomePrev}Q(A_4) \wedge \mathbf{AllPrev}Q(A_5) \wedge Q(A_3) \rightarrow Q^+(A_2) \succ Q^-(A_2)[A_4, A_5]$ having variations on propositions $Q^+(A_2)$, $Q^-(A_2)$, $Q(A_3)$, $Q(A_4)$, $Q(A_5)$. The number of iterations is RAN plus the maximum slide interval and the sequence identifier is the attribute A_1. Moreover, we executed experiments varying the number of top-k sequences (TOP). For this parameter, we used the values 4, 8, 12, 16 and 20 (8 is the default value). Greater values for TOP parameter causes more iterations in the loop of the *GetTopkSeq* algorithm.

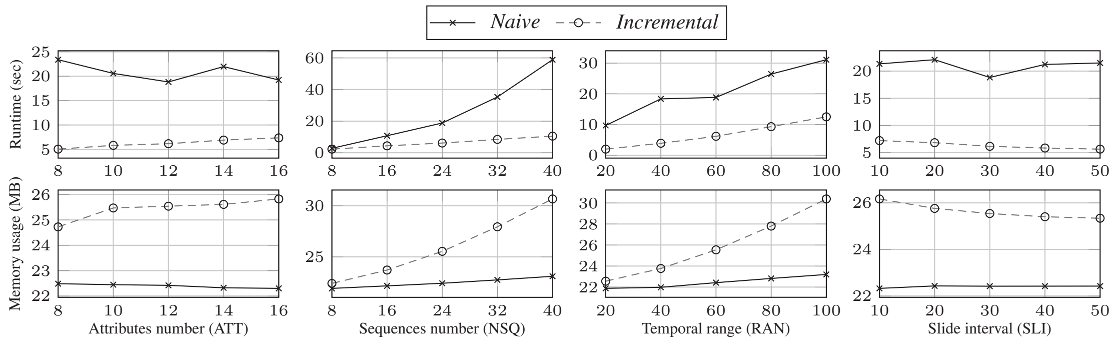

Figure 4. Experimental results for the parameters ATT, NSQ, RAN and SLI

Figure 4 shows the results obtained for the experiments with the parameters ATT, NSQ, RAN and SLI. Analyzing these results we observe that the incremental algorithm presented a better performance and a greater memory usage. This behavior is due to the maintenance of the index structure which speeds up the processing but consumes more memory. Considering the results of the experiments with the parameters NSQ and RAN, it is possible to see that for a greater number of sequences, the algorithm has to perform more comparisons consuming more process time. Moreover, the memory usage of the incremental algorithm increases to keep an index for more sequences. It is also important to notice that the executions with bigger temporal range imply in longer sequences resulting in higher runtime and memory usage.

Figure 5 presents the results of the experiments with the parameters RUL, LEV and TOP. The results obtained are similar to the ones obtained for the other parameters. Analyzing these results it is possible to see that the incremental algorithm showed a better performance and a higher memory usage. Among these results, it is important to highlight the results obtained with the variation in the number of rules (RUL). The naive algorithm presented a poor efficiency when dealing with more rules as more comparisons are required. The incremental algorithm, however, is few affected due to its index structure as addressed in Section 4.

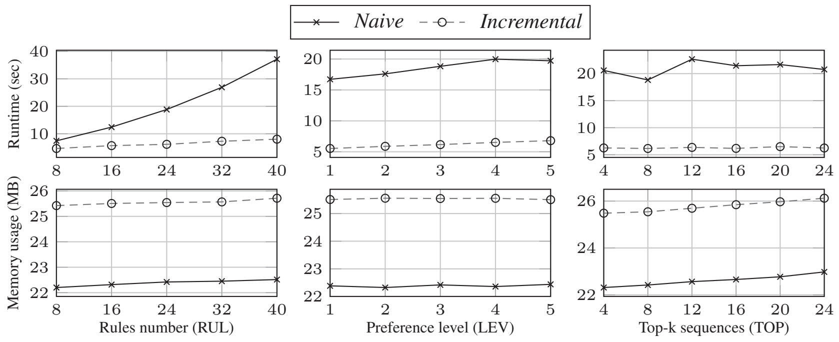

Figure 5. Experimental results for the parameters RUL, LEV and TOP

6. Related Work

The pioneering work on preference queries was the proposal of the *skyline* queries [Börzsönyi et al. 2001]. This research work provided support for many related studies. In [Chan et al. 2006], the authors introduced the concept of k-dominance. A tuple t k-dominates a tuple t' if t is better than t' in at least k attributes. The research work described in [Yiu and Mamoulis 2007] specified how to rank tuples using a *dominance degree*. The dominance degree of tuple t is the number of tuples dominated by t. The CPrefSQL query language proposed a new preference operator to compare tuples according to conditional preferences [de Amo and Ribeiro 2009].

The first research works about continuous preference queries were proposed by [Lin et al. 2005] and [Tao and Papadias 2006]. In [Lin et al. 2005] the authors explored the n-of-N problem for skyline queries, where a query is evaluated over the n most recent tuples, with $n \leq N$. The work of [Tao and Papadias 2006] designed algorithms to incrementally compute the preferred tuples over a sliding window with the most recent tuples. The work of [Kontaki et al. 2012] proposed algorithms for the evaluation of continuous preference queries over the most recent data, where each tuple has a timestamp and a validity interval. The evaluation of continuous preference queries using a graph-based index was introduced by [Santoso and Chiu 2014]. This work also designed an algorithm that outperforms the algorithms proposed in [Kontaki et al. 2012].

The first research work concerning the evaluation of continuous queries with conditional preferences (continuous cp-queries) were proposed by [de Amo and Bueno 2011, Petit et al. 2012]. In [de Amo and Bueno 2011], the authors presented an incremental algorithm based on ancestor lists for evaluating continuous cp-queries. The study described in [Petit et al. 2012] uses a graph structure to perform the same task.

In the work of [de Amo and Giacometti 2007], the authors proposed the TPref formalism to express temporal conditional preferences. The StreamPref formalism, proposed in [Ribeiro et al. 2017a], is a refinement of the TPref formalism. The StreamPref is more suitable for reasoning over data streams. In [Ribeiro et al. 2017b], the StreamPref formalism was used to define the query language *StreamPref*. The StreamPref language was originally composed of the operators $\mathbf{SEQ}$ and $\mathbf{BESTSEQ}$. The queries using the $\mathbf{BESTSEQ}$ operator are similar to skyline queries since both return the dominant elements according to the preferences. On the other hand, the queries with $\mathbf{TOPKSEQ}$ operator are a kind of top-k dominant query. In this case, the sequences are ranked using the preference level. We also should mention the importance of the CQL language [Arasu et al. 2006]. The CQL was not designed to work with preference queries, but it is a solid and expressive SQL-based declarative language for general purpose queries over data streams. In addition, the StreamPref query language is an extension of the CQL.

7. Conclusion

This paper presented the new operator $\mathbf{TOPKSEQ}$ for the StreamPref query language. The $\mathbf{TOPKSEQ}$ uses the preference level imposed by temporal conditional preferences to find the top-k sequences. First, we revisited the existing operators $\mathbf{SEQ}$ and $\mathbf{BESTSEQ}$ of the StreamPref. The $\mathbf{SEQ}$ operator extracts sequences from a data stream and the $\mathbf{BESTSEQ}$ is used to select the dominant sequences according to temporal conditional preferences. Considering that the $\mathbf{BESTSEQ}$ operator is not enough to obtain a good result to the user in all situations, the $\mathbf{TOPKSEQ}$ can be used to complement the results obtained using sequences with higher preference level.

We demonstrated the equivalence between the operators $\mathbf{TOPKSEQ}$ and $\mathbf{BESTSEQ}$. Moreover, we proposed an algorithm to evaluate the $\mathbf{TOPKSEQ}$ operator. This algorithm can use both the naive and the incremental strategies already proposed for the $\mathbf{BESTSEQ}$ operator. The extensive set of experiments performed showed a slightly greater memory usage for the incremental strategy recompensed by its superior performance.

Our future research directions include the possibility to use new approaches to rank the sequences beyond the preference level. We are also interested in exploring a new preference formalism to compare sequences considering not only the first different position but using a kind of distance based on preferences. Another future work is the development of algorithms for preference mining. The discovered preferences can be used in queries to monitor data streams.

Acknowledgments. The authors would like to thank the Research Agencies CNPq, CAPES and FAPEMIG for supporting this work.

References

Arasu, A., Babcock, B., Babu, S., Cieslewicz, J., Datar, M., Ito, K., Motwani, R., Srivastava, U., and Widom, J. (2016). *STREAM: The Stanford Data Stream Management System*, pages 317–336. Springer, Berlin, Heidelberg.

Arasu, A., Babu, S., and Widom, J. (2006). The CQL continuous query language: semantic foundations and query execution. *The VLDB Journal*, 15(2):121–142.

Börzsönyi, S., Kossmann, D., and Stocker, K. (2001). The skyline operator. In *ICDE*, pages 421–430, Heidelberg, Germany.

Chan, C.-Y., Jagadish, H. V., Tan, K.-L., Tung, A. K. H., and Zhang, Z. (2006). Finding k-dominant skylines in high dimensional space. In *ACM SIGMOD International Conference on Management of Data*, pages 503–514, Chicago, USA.

de Amo, S. and Bueno, M. L. P. (2011). Continuous processing of conditional preference queries. In *SBBD*, Florianópolis, Brasil.

de Amo, S. and Giacometti, A. (2007). Temporal conditional preferences over sequences of objects. In *ICTAI*, pages 246–253, Patras, Greece.

de Amo, S. and Ribeiro, M. R. (2009). CPref-SQL: A query language supporting conditional preferences. In *ACM SAC*, pages 1573–1577, Honolulu, Hawaii, USA.

Kontaki, M., Papadopoulos, A. N., and Manolopoulos, Y. (2012). Continuous top-k dominating queries. *IEEE Trans. on Knowledge and Data Eng. (TKDE)*, 24(5):840–853.

Lin, X., Yuan, Y., Wang, W., and Lu, H. (2005). Stabbing the sky: Efficient skyline computation over sliding windows. In *ICDE*, pages 502–513, Tokyo, Japan.

Petit, L., de Amo, S., Roncancio, C., and Labbé, C. (2012). Top-k context-aware queries on streams. In *DEXA*, pages 397–411, Vienna, Austria.

Ribeiro, M. R., Barioni, M. C. N., de Amo, S., Roncancio, C., and Labbé, C. (2017a). Reasoning with temporal preferences over data streams. In *FLAIRS*, Marco Island, USA.

Ribeiro, M. R., Barioni, M. C. N., de Amo, S., Roncancio, C., and Labbé, C. (2017b). Temporal conditional preference queries on streams. In *International Conference on Database and Expert Systems Applications (DEXA)*, Lyon, France.

Ribeiro, M. R., Pereira, F. S. F., and Dias, V. V. S. (2016). Efficient algorithms for processing preference queries. In *ACM SAC*, pages 972–979, Pisa, Italy.

Santoso, B. J. and Chiu, G.-M. (2014). Close dominance graph: An efficient framework for answering continuous top-dominating queries. *IEEE Transactions on Knowledge and Data Engineering (TKDE)*, 26(8):1853–1865.

Tao, Y. and Papadias, D. (2006). Maintaining sliding window skylines on data streams. *IEEE TKDE*, 18(3):377–391.

Yiu, M. L. and Mamoulis, N. (2007). Efficient processing of top-k dominating queries on multi-dimensional data. In *International Conference on Very Large Data Bases (VLDB)*, pages 483–494, Vienna, Austria.