Top-k Context-Aware Queries on Streams

Loïc Petit[1,3], Sandra de Amo[2], Claudia Roncancio[3], and Cyril Labbé[3]

[1] Orange Labs, France `name.surname@orange.com`
[2] Federal University of Uberlândia, Brazil `name.surname@ufu.br`
[3] Grenoble University, France `Name.Surname@imag.fr`

Abstract. Preference queries have been largely studied for relational systems but few proposals exist for stream data systems. Most of the existing proposals concern the skyline, top-k or top-k dominating queries, coupled with the sliding-window operator. However, user preferences queries on data streams may be more sophisticated than simple skyline or top-k and may involve more expressive operations on streams. This paper improves the existing work on data stream query-answering personalization by proposing a solution to express and handle *contextual* preferences together with a large variety of queries including one-shot and continuous queries. It adopts a more expressive preference model supporting context-based preferences, allowing to capture a wide range of situations. We propose algorithms to implement the new preference operators on stream data and validate their performance on a real-world dataset of stock market streams.

1 Introduction

Query-answering personalization has been attracting much attention in the database community in recent years [3, 7]. Such works have been motivated by the need to select the data items that better fit user preferences. This is useful in situations when the number of potential answers is either too high or too small. When it is too high, user preferences are used to restrict the answer set by identifying the subset of the most preferred data items. On the other hand, some queries may involve hard conditions which imply a very small (or even a disappointing empty) answer-set. In this case, user preferences could be used to enhance the set of retrieved data by including answers which could be of user interest even if they do not verify the hard constraints specified in the query.

Numerous application domains such as financial, monitoring and sensor-based applications require now data stream management. Supporting preference queries on evolving data is more challenging than their evaluation on persistent data. *Contextual* preference queries are particularly helpful to users dealing with data streams. For instance, in a stock market scenario, buyers may want to know the most interesting deals so far before making their trading decisions. Some statistical data such as the *volatility rate* of the stock options in the last three days or the *economic situation* of the stock options country can influence their decision. So, it is possible to support queries as “What are the most interesting deals you propose given that for stock options coming from countries in bad economic situation in the last year I prefer those presenting a lower volatility rate in the last three days" may be issued. The existing proposals on contextual preferences query processing [11, 4] are designed for conventional DBMS and are not tailored to handle stream data. Besides, few proposals in the literature support preference queries on data streams [12, 13, 9]. Most of them concern the skyline, top-k and top-k dominating queries, coupled with the sliding-window operator. To the best of our knowledge there are no proposal in the literature dealing with *contextual preference* query processing on data streams.

This work goes a step beyond by proposing contextual preference queries on both conventional and stream databases. We consider the stream algebra Astral introduced in a previous paper [16] as the core stream query language. We extend Astral by the introduction of two preference operators Best and KBest. These operators are adapted from the preference operators of the query language CPrefSQL originally designed for querying static data [4].

Main Contributions. The main contributions of this paper can be summarized as follows: (1) The introduction of the top-k contextual preference queries in a data stream context; (2) The introduction of two new operators in the Astral algebra designed to query relations and data streams; (3) The design and implementation of incremental algorithms for evaluating continuous and instantaneous queries over streams and relations; (4) The implementation of the preference operators in the original prototype of Astral and their performance evaluation.

This paper is organized as follows: In Section 2, we motivate our proposal by presenting a real-world scenario where user preferences are naturally influenced by the user context. Section 3 introduces the main theoretical concepts underlying the preference model and the stream algebra Astral. In Section 4, we present the Preference Astral algebra incorporating two new preference operators Best and KBest. In Section 5, we present the incremental algorithms for continuously evaluating the preference operators Best and KBest. In Section 6 we present and discuss some experimental results. Related work are resented in section 7. Finally, in Section 8 we conclude the article and present some research perspectives.

2 A Motivating Example

Tom is a very cautious investor who likes to get as much information as possible before making his decisions about buying and selling stocks shares. He has free access to a web site that provides information about real-time quotations and volatility rates as well as real-time transactions. These data involve the following data streams and static data stored in a relational DBMS:

• Relation *StockOption(<u>StOpName</u>, Category, Country)*: stores the stock name, its category (Commodities (c), Info-Tech (it)), and the country where the company headquarters are located.
• Stream *Transactions(<u>OrderID</u>, TTime, StOpName, Volume, Price)*: a data stream providing real-time information about stock options transactions. It in-

cludes the transaction time ($TTime$), the quantity of shares ($Volume$) and the price ($Price$) of the stock option share.
• Stream *Volatility(StOpName, ETime, Rate, Method)*: a data stream providing real-time information about the estimated volatility ($rate$) of stock options. It includes the time of the estimation ($ETime$) and the estimation method ($Method$).

Based on his past experience and the information he reads in the papers, Tom has some preferences he wants to be taken into account in order to facilitate and speed up his decisions. His preferences are described by the following statements:

[P1] Concerning commodities stocks, at each moment Tom prefers those with a volatility-rate less than 0.25. On the other hand, concerning IT stocks, Tom is more aggressive and prefers those with a volatility-rate greater than 0.35.
[P2] For stock options with volatility-rate greater than 0.35 at present (calculated according to some method) Tom prefers those from Brazil than those from Venezuela.
[P3] For stock options with volatility-rate greater than 0.35 at present, Tom is interested in transactions carried out during the last 3 days concerning these stock options, preferring those transactions with quantity exceeding 1000 shares than those with a lower amount of shares.

Notice that Tom's preferences are expressed by means of *rules* of form `IF` *some context is verified* `THEN` *Tom prefers something to something else. Contexts* are conditions involving the values of some data attributes. For instance, in statement [P1] the context is $StockOption.Category = `Commodities'$ and Tom's preference is $Volatility.Rate \leq 0.25$ *better than* $Volatility.Rate > 0.25$. Preference rules may involve streams or relational data on both the context side and the preference side of the rule.

As well as his preferences, Tom's queries may concern relational and stream data and be "one-shot" or continuous queries. Here are some of Tom's:

[Q1] Considering the last 100 transactions with a volume greater than 1000 shares, list my top 10 most preferred ones.
[Q2] Give me the list of quotations during the last 2 days, concerning the stock options which most fulfill my preferences.
[Q3] Give me the list of quotations during the last 2 days, concerning only IT stock options which most fulfill my preferences.
[Q4] Every 30 minutes, give me a complete description of my 10 preferred stock options: country, category, the last transaction concerning the stock option (volume and quotation), the volatility rate with its corresponding estimating method.

It is important to emphasize that, differently from the *hard constraint* expressed by statements like "IT stock options", preferences should be viewed as *soft constraints*: If no database entry fulfills the hard constraints (for instance, there is no IT stock options in the database), the result answer-set is empty. On the other hand, if there are not K tuples in the database which are considered *perfect* according to the preferences, a list of K tuples respecting the preference hierarchy is returned instead.

3 Preliminaires

To achieve our work we build-on two existing proposals (1) the Astral algebra proposing operators to query data stream and relational data together and (2) theoretical foundations on contextual preferences rules. This Section introduces such proposals. Section 4 and the following, present our proposal to integrate preferences in Astral queries.

3.1 The Astral Stream Algebra

We use the Astral algebra [16] which provides a formal definition of operators involved in streams querying. Such a formalism facilitates the expression and understanding of queries. Putting aside preferences and the top-k operator, queries presented in section 2 can be expressed in Astral. This section provides the very few definitions necessary to introduce Astral queries. Our examples refer to queries **Q1, Q2, Q3 and Q4** of section 2. We'll use **Q1', Q2', Q3' and Q4'** which are their counterpart without preferences.

In Astral, streams and relations (denoted by S and R respectively in the following) are two different concepts [1]. A *stream* S is a possibly infinite set of tuples s with a common schema containing two special attributes: a timestamp t and, the position in the stream, p [4]. A *temporal relation* R is a step function that maps a time identifier t to a set of tuples $R(t)$ having a common schema. Classical relational operators, as selection σ, projection π and join $\bowtie$, are extended to temporal relations. The extension of π and σ for streams is simple whereas joins are very complex. For example, $\sigma_{Volume>10000}(Transactions)$ is the stream of transactions having $Volume > 10000$, whereas $\sigma_{Category=it}(StockOption)$ is a *temporal relation* containing only tuples for which $Category$ is it .

A temporal relation can be extracted from a stream using windows operators. Astral provides an extended model for windows operators including positional, temporal and non standard cross domain windows (e.g. slide n tuples every δ seconds). The following expressions represent some very useful windows: (a) $S[L]$ contains the last tuple of a stream;
(b) $S[A/L]$ is the partitioned window containing the last tuple of each sub-stream identified with the attribute A. For example $Transaction[StOpName/L]$ contains the last known transaction for each stock option.
(c) $S[N \text{ slide } \Delta]$ is a sliding window of size N sliding Δ every Δ. N and Δ are either a time duration or a number of tuples. For instance, **Q1'** is written as $(\sigma_{Volume>10000}(Transactions))[100 \text{ slide } 1]$. The window is 100 tuples large and slides of 1 tuple whenever one new tuple arrives. **Q2'** is $\pi_{Price}(Transactions[2\ days \text{ slide } 1])$. The window is 2 *days* large and slides of 1 tuple whenever one new tuple appears.

A stream can be generated from a temporal relation using a *streamer* operator. Among them, $I_S(R)$ produces the stream of tuples inserted in R. Streamers and windows may be composed in order to join two streams or a stream and a relation. Given a window description W, a streamer Sc and a join condition c, the join operator (stream×relation)→stream can be defined as:

[4] These definitions can be extended using the notion of batch [16].

	StOpName	*Cat*	*Country*	*ETime*	*Rate*	*Method*
t_1	MS	it	USA	T1	0.30	M1
t_2	AP	it	India	T2	0.55	M1
t_3	USSteel	c	USA	T1	0.20	M2
t_4	Petr4	c	Brazil	T2	0.40	M2
t_5	Bel5	m	Venezuela	T3	0.55	M2

(a)

$$\begin{tikzcd} & t_3 \arrow[dl] \arrow[dd, dashed] & t_2 \arrow[dd] \\ t_4 \arrow[dr] & & \\ & t_5 & t_1 \end{tikzcd}$$

(b)

Fig. 1. Instance of *StockOption* $\bowtie$ *Volatility* and Better-than Graph associated to Γ

$S \bowtie_c R = Sc(S[W] \bowtie_c R)$. In the following we will use: $S \bowtie_c R = I_S(S[L] \bowtie_c R)$. The stream $S \bowtie_c R$ contains tuples generated by updates in R. Other types of join operators can be defined. For instance, tuples can be added to the output stream only for new tuples in S and not when R is updated: the semi-sensitive-join operator (stream×relation)→stream produces a stream resulting from a join between the last tuple of the stream and the relation at the time of the last tuple of the stream: $S \ltimes_c R = I_S(S[L] \bowtie_c R(\tau_S(S[L])))$. Here τ_S denotes the function that gives the timestamp of a tuple in the stream S. (see [16] for more details). For instance, query **Q3'** is written as

$$\pi_{Price}((Transactions \ltimes (\sigma_{Category=IT}(StockOption)))[2\ days \text{ slide } 1]).$$

The expression for query **Q4'** provides a stream built from the join between the $Transaction$ stream, the last known values of the $Volatility$ and the *StockOption* relation. The required windows expression is $[W] = [30\ min \text{ slide } 30\ min]$.

3.2 The Preference Model

In this section we present the main concepts concerning the logical formalism for *specifying and reasoning* with preferences. Details can be found in [4, 5].

Let R be a relational schema with attributes $\text{Attr}(R) = \{A_1, A_2, ..., A_n\}$. R can be a non-temporal or temporal relational schema. If R is temporal then one of its attributes is T (time). For each attribute $A \in Attr(R)$, let $\mathbf{dom}(A)$ be the set of values of A (the domain of A). The set $\text{Tup}(Attr(R)) = \mathbf{dom}(A_1) \times \mathbf{dom}(A_2) \times ... \times \mathbf{dom}(A_n)$ is the set of all possible tuples over $Attr(R)$.

Definition 1 (Contextual Preference Rules) A *conditional preference rule* (or *cp-rule* for short) over the relational schema R is a statement φ of the form: φ: $u \rightarrow Q_1(X) \succ Q_2(X) \quad [W]$ where:
- X is a non-temporal attribute of R, $W \subseteq Attr(R)$, $X \notin W$,
- $Q_i(X)$ (for $i = 1, 2$) is a statement of the form $X\theta a$ where $\theta \in \{=, \neq, \leq, \geq, <, <\}$ and $a \in \mathbf{dom}(X)$.
- There is no $x \in \mathbf{dom}(X)$ satisfying both $Q_1(X)$ and $Q_2(X)$ simultaneously. For instance, $X > 1$ and $X \leq 3$ cannot be considered as statements $Q_1(X)$ and $Q_2(X)$ in the right side of a cp-rule, since $X > 1 \cap X \leq 3 = (1, 3] \neq \emptyset$.

• u is a conjunction of simple statements of the form: $A_1\theta_1 a_1 \wedge ... \wedge A_k\theta_k a_k$, where $\theta_i \in \{=, \leq, \geq, <, <\}$ for $i = 1, ..., k$. We assume X and the attributes in W do not appear among the attributes of u.

The formula u in the left side is called the *context* of the rule φ. The statement $Q_1(X) \succ Q_2(X)$ in the right side is called the *preference statement* and the attributes in W are called the *ceteris paribus* attributes. This will be clearer in the sequel. A tuple $t \in Tup(Attr(R))$ is said to be *compatible* with a cp-rule φ if t satisfies its context. For instance, the tuples $t_1 = (1, 2, 4, 1)$ and $t_2 = (0, 2, 5, 6)$ over the relation schema $R(A, B, C, D)$ are compatible with the cp-rule $(A < 2 \wedge B = 2 \wedge C > 3) \rightarrow (D \leq 2 \succ D > 4)$. The context of this cp-rule is the formula $(A < 2 \wedge B = 2 \wedge C > 3)$ and its preference statement is $(D \leq 2 \succ D > 4)$. Intuitively, this cp-rule means that between two tuples compatible with the context $(A < 2 \wedge B = 2 \wedge C > 3)$ I prefer the one with $D \leq 2$ than the one with $D > 4$. So, between the tuples t_1 and t_2, I prefer t_1.
A *contextual preference theory* (*cp-theory* for short) over R is a finite set of cp-rules Γ over R. We denote by $\text{Attr}(\Gamma)$ the set of attributes appearing in the cp-rules of Γ. Notice that $\text{Attr}(\Gamma) \subseteq \text{Attr}(R)$.

Example 1: Let us consider the two preference statements P1 and P2 of our motivating example. They can be expressed by the following cp-theory over the schema $T(StOpName, Cat, Country, ETime, Rate, Method)$:
• φ_1: $Cat = c \rightarrow (Rate < 0.25 \succ Rate \geq 0.25)$, $[M]$
• φ_2: $Cat = it \rightarrow (Rate \geq 0.35 \succ Rate < 0.35)$, $[M]$
• φ_3: $Rate > 0.35 \rightarrow (Country = Brazil \succ Country = Venezuela)$

The attributes between brackets mean that in order to compare two tuples by means of a cp-rule, these tuples must coincide on these attributes. For the other attributes there is no restriction. For instance, in the scenario of Example 1, let t_1 and t_2 as described in Figure 1(a). Then t_1 and t_2 can be compared by using the rule φ_2, since they have the same context ($Cat = it$), the *Rate* (volatility rate) of t_2 is greater than 0.35 and the *Rate* of t_1 is lower than 0.35, and the method used to measure *Rate* is the same for both tuples.

It is clear by now that a cp-rule φ over R induces a *binary relation* (denoted by $\succ_\varphi$ on the set $Tup(R)$: the set of pairs (t, t') such that t is better than t' according to φ. Of course, this binary relation is not necessarily an *order relation*, since it is not always transitive. In the following we define the notion of *Preference Relation* induced by a cp-theory Γ.

Definition 2 (Preference Relation) *Let Γ be a contextual preference theory over a relational schema R (temporal or non-temporal). The Preference Relation associated to Γ (denoted by $\succ_\Gamma$) is defined as: $\succ_\Gamma = (\bigcup_{\varphi\in\Gamma} \succ_\varphi)^*$, where $*$ denotes transitive closure.*

Example 2: Let us consider the cp-theory Γ of Example 1. Let us consider instances I and J of relation schemas *StockOption* and $Volatility$ respectively, such that the result of *StockOption* $\bowtie$ *Volatility*(I, J) is given in Figure 1(a). It is clear that $t_3 \succ_{\varphi_1} t_4$ and $t_4 \succ_{\varphi_3} t_5$. Then, by transitivity, we conclude that

$t_3 \succ_\Gamma t_5$. Notice that t_3 and t_5 cannot be compared using only one rule in Γ. However, they can be compared by transitivity using different rules in Γ.

Discussion. We say that a cp-theory Γ is *consistent* if and only if the induced order $>_\Gamma$ is irreflexive and consequently, a *strict partial order* over $Tup(R)$. In [17], a sufficient condition for ensuring consistency of a cp-theory is given. This condition involves testing the acyclicity of the *dependency graph* associated to the cp-theory and its *local consistency.* For lack of space we omit the details. In this paper, we will suppose our cp-theories are consistent, that is the associated Preference Relation $\succ_\Gamma$ is a strict partial order. For more details on the theoretical foundations and consistency test see [5].

4 Introducing Preference Operators into Astral

Let us focus on the integration of contextual preferences in the Astral algebra. This Section presents the syntax and semantics of the preference operators integrated to Astral whereas Section 5 presents the algorithms for implementing them.

4.1 Global approach

The objective of our proposal is to provide an integrated solution where the full expressivity of both, queries and preferences are available. We propose to capture the semantics of the preference evaluation as algebraic operators that extend the Astral algebra. Such preference operators can be part of instantaneous and continuous queries performing on streams and relations, and using any of the existing operators. Particularly, queries involving data streams can use the wide variety of temporal, positional and hybrid windows [15]. The preference operators calculate user preferred answers according to the available cp-theory. Each user provides the system with his/her preferences (a cp-theory Γ) which become some kind of *user profile.* During querying, these preferences are used for answer customization if the user asks for. Concretely, we will allow powerful contextual *most preferred* and *top-k* queries by the introduction of two operators:
(1) The $\textbf{Best}_\Gamma$ operator selects a subset of *optimal* tuples according to user preferences Γ.
(2) The $\textbf{KBest}_\Gamma$ operator selects the K most preferred tuples respecting the preference *hierarchy* specified by Γ. For the sake of simplifying the presentation we omit the subscript Γ whenever it is implied by the context.

4.2 Best and KBest operators

The **Best** operator selects from a given temporal or non-temporal relation those tuples which are not dominated by other tuples according to the preference order inferred from Γ (see Definition 2).

Definition 3 (Best) *Let R be a relational schema and Γ be a cp-theory over R. Let $r(t)$ be an instance over R at time t* $\textbf{Best}(r(t)) = \{u \in r(t) \mid \nexists v \in r(t)$ *such that* $v \succ_\Gamma u$ $\}$

The operator **KBest** selects the *top-k tuples* according to the preference *hierarchy* dictated by Γ. Intuitively, $\mathbf{KBest}(I, k)$ returns the set of k tuples of I having the minimum number of tuples dominating them in the preference hierarchy. In order to define its semantics, we need first to introduce the notion of *level* of a tuple u (denoted by $l(u)$) according to a cp-theory Γ. The level of a tuple reflects "how far" is the tuple from the most preferred ones (those which best fit the user preferences).

Definition 4 (Level) *Let R be a relational schema and Γ be a cp-theory over R. Let $r(t)$ be a tuple-set or instance of R at time t, and let tuple $u \in r(t)$. The level of u, $l(u)$, according to Γ is inductively defined as follows:*

– *If there is no $u' \in r(t)$ such that $u' \succ_\Gamma u$, then $l(u) = 0$.*
– *Otherwise $l(u) = 1 + max\{l(u') \mid u' \succ_\Gamma u\}$*

It is easy to show that if $u \succ u'$ then $l(u) < l(u')$. The reverse implication does not hold. The semantics of the **KBest** operator is defined as follows.

Definition 5 (KBest) *Let R be a temporal (or non-temporal) relation and Γ be a cp-theory over R. Let $r(t)$ be a tuple-set or instance of R at time t. $\mathbf{KBest}(r(t), k)$ is the set of the k tuples $\in r(t)$ with the lowest levels. The positional order is used to sort tuples at the same level.*

Example 3: Let us consider the cp-theory $\Gamma = \{\varphi_1, \varphi_2, \varphi_3\}$ of Example 1 and the instantaneous relation I of Figure 1(a). Figure 1(b) shows the *Better-Than Graph G* (BTG) associated to cp-theory Γ over I. The nodes are the tuples of I. An edge (t_i, t_j) expresses that t_i is preferred to t_j according to a rule of Γ. A dotted edge (t_i, t_j) means that t_i is preferred to t_j by transitivity. We have that $\mathbf{Best}(I) = \{t_1, t_3\}$, since these are the tuples which are not dominated by others. Note that level(t_1) = level(t_3) = 0, level(t_2) = level(t_4) = 1 and level(t_5) = 2. So, $\mathbf{KBest}(I, 3) = \{t_1, t_3, t_2\}$. As t_2 and t_4 have the same level in the preference hierarchy the positional order is used to decide between them.

Let us consider the query **Q2** of section 2. It is expressed in the extended algebra by:

$$\pi_{Price}\ \mathbf{Best}(Transactions[2\ days\ \text{slide } 1])$$

5 Best and KBest Algorithms

This section presents the algorithms we propose to evaluate the **Best** and **KBest** operators (see section 5.2). As such operators require the preference hierarchy of tuples which is represented by a BTG, the algorithms to create the BTG are first introduced in section 5.1. Section 5.3 presents an incremental alternative to manage the BTG.

Algorithm 1: Compare(t_1, t_2, φ)

```
Data: $\varphi : u \rightarrow Q_1(X) > Q_2(X)$ $[W]$
Result: $\{1, -1, \emptyset\}$,
  resp. $\{t_1 >_\varphi t_2$, $t_1 <_\varphi t_2$, inc.$\}$
if $t_1 \not\models u$ || $t_2 \not\models u$ then return $\emptyset$
foreach $V \in W$ do
    if $t_1(V) \neq t_2(V)$ then return $\emptyset$
if $t_1 \models Q_1(X)$ & $t_2 \models Q_2(X)$ then
    return 1
if $t_2 \models Q_1(X)$ & $t_1 \models Q_2(X)$ then
    return $-1$
return $\emptyset$
```

Algorithm 2: CompT(t_1, t_2, Γ) (without transitive closure)

```
Data: $\Gamma = \{\varphi_1, ..., \varphi_k\}$ a
      cp-theory
Result: $\{1, -1, \emptyset\}$,
  resp. $\{t_1 >_\Gamma t_2$, $t_1 <_\Gamma t_2$, inc.$\}$
foreach $\varphi_k \in \Gamma$ do
    $r \leftarrow Compare(t_1, t_2, \varphi_k)$
    if $r \neq \emptyset$ then return $r$
return $\emptyset$
```

5.1 The Preference Hierarchy and the Better-Than Graph

First and foremost, we introduce the algorithm to establish the preference order between two tuples t_1 and t_2 according to a rule φ. The straight forward Algorithm 1 returns $\emptyset$ if t_1 and t_2 are incomparable, 1 if t_1 is more preferred than t_2 and -1 otherwise. Algorithm 2 extends the comparison to a cp-theory Γ. It relies on the set of rules (φ_n) to identify the preference order. However, it does not compute the transitive closure as stated in the cp-theory definition. The transitivity will be computed in the preference algorithms.

Better-Than graph: The Best and KBest preference operators are applied on a tuple-set TS and require the *BTG* of TS. For its implementation we adopt the $Graph(Next, Prec, Src)$ defined as follows: $Next$ associates to each tuple the list of its direct dominated tuples. $Prec$ associates to each tuple the list of tuples that directly dominate it. Src, the no-dominated tuples, that is the sources of the graph. From a formal point of view:

$$Next = s \in TS \mapsto \{s' \in TS,\ \exists \varphi_n \in \Gamma,\ s >_{\varphi_n} s'\}$$
$$Prec = s \in TS \mapsto \{s' \in TS,\ \exists \varphi_n \in \Gamma,\ s <_{\varphi_n} s'\}$$
$$Src = \{s \in TS,\ Prec(s) = \emptyset\}$$

To provide good performance, the implementation of the graph uses hash-sets and hash-maps. The $Next$ and $Prec$ functions also have a $keys()$ method defined by: $s \in F.keys() \Leftrightarrow F(s) \neq \emptyset$.

The construction and maintenance of the graph require *Insert* and *Delete* methods. To insert a tuple, Algorithm 3 iterates over the graph to update *Next*, *Prec* and the *Src* set. As the cost of insertion and deletion in a hash structure can be considered as $\mathcal{O}(1)$, the global cost of the insertion of a tuple in the graph is $\mathcal{O}(|G|)$. The deletion of a tuple s, presented in Algorithm 4, is an iteration over the nodes connected to the one we delete[5]. The cost is $\mathcal{O}(\text{degree}(s))$.

Given a known cp-theory Γ and a tuple-set, the construction of the entire BTG relies on the insert method (see Algorithm 6).

[5] As a recall, in graph theory, the number of edges connected to a node is called the *degree* of a node.

Algorithm 3: G.Insert ; Insert a tuple in the BTG

```
Input: A tuple $s, \Gamma$ and the BTG structure
$L \leftarrow Prec$.keys()$\cup Src$
$Src$.add($s$)
foreach $s' \in L$ do
    $r \leftarrow$ CompT($s, s', \Gamma$)
    if $r > 0$ then
        $Src$.remove($s'$)
        $Prec$.put($s', s$)
        $Next$.put($s, s'$)
    else if $r < 0$ then
        $Src$.remove($s$)
        $Prec$.put($s, s'$)
        $Next$.put($s', s$)
```

Algorithm 4: Graph.Delete ; Removes a tuple from the BTG

```
Input: A tuple $s$ and the BTG structure
$Src$.remove($s$) ; $P \leftarrow Prec$.remove($s$) ;
$Dom \leftarrow Next$.remove($s$)
foreach $s' \in P$ do
    $Anc \leftarrow Next$.get($s'$)
    if $Anc.size() = 1$ then  $Next$.remove($s'$)
    else if $r < 0$ then  $Anc$.remove($s$)
foreach $s' \in Dom$ do
    $Anc \leftarrow Prec$.get($s'$)
    if $Anc.size() = 1$ then
        $Src$.add($s'$)
        $Prec$.remove($s'$)
    else if $r < 0$ then  $Anc$.remove($s$)
```

5.2 Evaluation of Best and KBest

By definition, the *Graph* includes *Src* which corresponds to the most preferred tuples. *Src* is the answer of the Best operator. However, Algorithm 3 can be optimized by avoiding the entire construction of the *BTG*. The *Prec.put* and *Next.put* sequences can be suppressed from it. This variant will be named *Src* in section 6. It's complexity is so reduced to $\mathcal{O}(|Src|)$. The complexity of $Best(R)(t)$ becomes $\mathcal{O}(NS)$ where $N = |R(t)|$ and $S = |Best(R)(t)|$.

The main algorithm used to compute *KBest* from the *BTG* is a Kahn-topological sort limited to k results. See Algorithm 5.

Its complexity is majored by the complexity of the Kahn algorithm which is $\mathcal{O}(N + |Next|)$. The limitation to k introduces a global factor $\frac{k}{N}$. Leading to $\mathcal{O}(k + kD))$, where D is the average degree of each node (majored by $\mathcal{O}(N)$). The complexity of KBest is therefore $\mathcal{O}(N^2)$. Moreover, as k is usually small compared to N, and D is more likely to be very small compared to N (many tuples are not comparable) then the major complexity factor comes from the construction of the BTG.

5.3 Incremental Evaluation of BTG

This section introduces the obtention of BTG in an incremental way. This is motivated by queries over data streams where preferences are evaluated on sequences of windows. The BTG is required for the tuple-set contained in the *current* window. As two successive windows may overlap, then the new BTG can be constructed by incremental updates of the *current* one. The implementation of Astral's window sequences makes available two delta sets wrt the current window and the next one: δ_R^- are the tuples that "exit" from the window and, δ_R^+ are the ones arriving for the new window. There is no intersection between these sets. These delta sets are used to obtain the BTG of the new window based on the preceding one as shown in Algorithm 7.

Algorithm 5: Calculate KBest$(R)(t)$

```
Data: The BTG structure, $k$ the number of required tuples
Res $\leftarrow$ new TreeSet() ;                       /* Ordered set */
if $k < |Src|$ then                       /* Src contains more than $k$ best items */
    $N \leftarrow |Src| - k$ ;            /* The positional order in Src is used */
    foreach $s \in Src$ do                /* to keep the k more "recent" items */
        if $N = 0$ then Res.add($s$)
        else $N \leftarrow N - 1$
    return $Res$
NextLvl $\leftarrow Src$; $id \leftarrow 0$; PrecCount $\leftarrow$ new HashMap()
while $id < k$ and $id < |Src| + |Prec.keys()|$ do
    if $Buffer = \emptyset$ then /* Buffer contains tuples with the same level */
        foreach $t \in NextLvl$ do
            Buffer.push($t$)
        NextLvl.clear()
    $t \leftarrow$ Buffer.pop()
    foreach $s \in Next.get(t)$ do        /* For each node dominated by $t$ */
        $n \leftarrow$PrecCount.get($s$)
        if $n =$null then $n =$Prec.get($s$).size()
        if $n = 1$ then NextLvl.add($s$);  /* $s$ is part of the next level */
        else PrecCount.put($s, n - 1$); /* There are more nodes to browse */
    Res.add($t.copy(id{+}{+})$);          /* Update the positional order */
return $Res$
```

The complexity of updating the BTG is $\mathcal{O}(\left|\delta_R^-\right|.D + \left|\delta_R^+\right|.N)$, where D is the average degree of a node in the graph. If we consider that the size of the δ_R are similar and that D is ruled by $\mathcal{O}(N)$ then the complexity becomes $\mathcal{O}(|\delta_R|.N)$.

Table. 1 hereafter summarizes the complexity of the preference operators for the two BTG construction approaches. It is worth noting that the incremental approach is really interesting if the delta sets are small compared to the total number of nodes. A large portion of the current BTG can be reused for the new one. If it is not the case the BTG creation "from scratch" performs better.

Learning inferred preferences

The proposed implementation applies the mathematical definition of the preference order and does not keep trace of inferred preferences. For instance, for tuples s_1, s_2, s_3, if $s_1 <_\Gamma s_2$ and $s_2 <_\Gamma s_3$ then by transitivity $s_1 <_\Gamma s_3$. Now, if s_2 is no more in the current scope, we would say $s_1 \nless_\Gamma s_3$. However, at some point in time it was known that $s_1 <_\Gamma s_3$ and this knowledge could be reused. A small change in the Graph.Remove function allows us to provide that semantics if desired. When removing a node, the preceding nodes will be linked to the following nodes. The global complexity doesn't change.

Algorithm 6: Create BTG

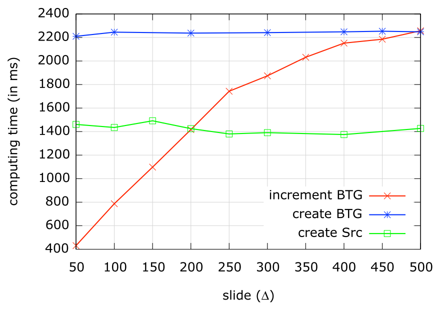
```
Input: A tuple-set $TS$
Data: The BTG structure
foreach $s \in TS$ do $Graph$.Insert($s$)
```

Algorithm 7: Incremental BTG

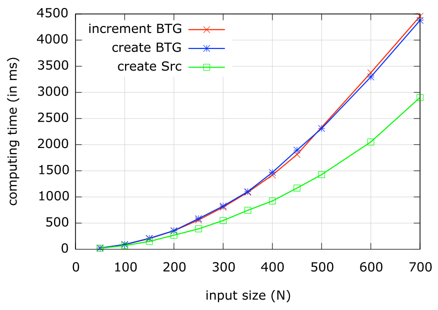
```
Input: $\delta_R^-$ and $\delta_R^+$
Data: The BTG structure
foreach $s \in \delta_R^-$ do
    $Graph$.Remove($s$)
foreach $s \in \delta_R^+$ do
    $Graph$.Insert($s$)
```

	BTG	Incremental BTG
Best	$\mathcal{O}(N.S)$	$\mathcal{O}(\Delta.N)$
KBest	$\mathcal{O}(N^2)$	$\mathcal{O}((\Delta + k)N)$

Table 1. Best/KBest complexity



(a) Varying Δ on KBest (for $N = 500$)



(b) Varying N on KBest (for $\Delta = N$)

Fig. 2. Computing time of Incremental BTG, Create BTG and Src maintenance.

6 Experimental Results

The algorithms presented in this paper have been implemented as extensions of the Astral DSMS Prototype[6]. These extensions have been facilitated by its SOA architecture. We performed experiences to study the behaviour of both the **Best** and the **KBest** operators.

Experimentation Setup: A quad-core Intel Xeon 2.6GHz computer with 6GB of RAM is used along with the Sun/Oracle 1.6 JVM, an Apache Felix OSGi platform with Astral. 30,000 tuples have been gathered from real-world quotes[7]. We used the preferences presented in Section 2 and the queries of the running example. Let us focus here on the experiments with queries in the style of **Q3** and **Q4** of Section 2. These are top-k queries over stream.

$$S = Transaction \ltimes (Volatility[StOpName/L] \bowtie StockOption)$$

The query with the KBest operator uses sliding windows as follows: **KBest**($S[N$ slide $\Delta], k$)

[6] Available at `http://astral.googlecode.com` under Apache 2 Licence

[7] Dump provided by Dukascopy's Data Export service Available at `http://www.dukascopy.com/swiss/english/data_feed/csv_data_export`

Results: Experiments show that the evaluation time of **KBest** is dominated by the construction/update of the *BTG*. We also observed the evolution of the structure of the BTG from one window to the next one: the maximum level varies from 2 to 6 and the number of non-dominated tuples varies from 1 to N. Big changes in the structure of the graph are *bad cases* for the incremental *BTG* algorithm (Algorithm 7) of the **KBest** operator.

Figure 2(a) shows the computing time of the two algorithms of *BTG*: create (Algorithm 6) and incremental (Algorithm 7). It also shows the time for the algorithm reduced to the *Src* maintenance. This can be used for the Best operator. In the experiments we used several window sizes (N) and rates (Δ). We noticed that changes in the rate do not impact the time for create *BTG*, whereas the incremental algorithm performs 6 times better for a N/Δ ratio $= 10$. Surprisingly, the two *BTG* algorithms behave similarly when $\Delta \sim N$ which correspond to few or no intersection between successive windows. This means that in the incremental version, the deletions in the *BTG* do not take long time compared to insertions. This may be not true when the *BTG* is a strongly connected graph with nodes with high level (though very unlikely in practice).

The variation of the size of the window (figure 2(b) with $\Delta = N$) shows that the behavior is not impacted by the number of tuples involved. As expected, the evolution is N quadratic and the incremental algorithm strictly follows the performance of the create algorithm.

7 Related work

The problem of enhancing well-known query languages with preference features has been tackled in several recent and important work in the area. For a comprehensive survey on preference modeling, languages and algorithms see [10]. In this section we present some related work concerning contextual preference support in traditional databases and preference support in stream data.

Contextual Preference Support. In the database field, several proposals for incorporating context in query languages exist in the literature. In [11] preferences are expressed in a *quantitative* format, that is, by means of scores associated to attribute-value clauses. A *contextual query* is a standard query enhanced with a user context. The main problem tackled in these papers is identifying the preferences that are most relevant to a contextual query and presenting an algorithm to locate them. The approach we adopt in this paper follows a *qualitative* model to express preferences: preferences are expressed by a (small) set of rules from which is inferred a strict *partial* order on tuples. Moreover, we assume that the contextual preferences are given and incorporated into the query language syntax. Qualitative approaches has many advantages when compared to quantitative ones due to their conciseness and deduction capability.

Top-k Preference Queries. In [6] the *top-k queries* have been introduced in a *quantitative* preference model setting, that is, where preference between tuples is expressed by a score function defined over the dataset. The *top-k dominating queries* have been introduced in [14] as an extension of the skyline queries of [3] which were originally designed to return the most preferred tuples, without any user control on the size of the result. A *top-k dominating query* returns the k tuples which dominated the maximum amount of tuples in the database. This concept is orthogonal to the skyline and pareto queries, as well as to the approach CPrefSQL we adopt in this paper.

Preference Support on Stream Data. Most work on preference queries in data streams [12, 13, 9] concern methods for the continuous evaluation of skyline queries, top-k queries and top-k dominating queries under the sliding window model. In these proposals, a preference operator is coupled with two forms of the sliding window operator over data streams: the *count-based* and the *time-based* ones. In the count-based sliding window the last N tuples of a stream are returned and for each arriving tuple, the oldest one expires. In the time-based sliding window, the active tuples are those arrived during the last T time instants. The preference operators are applied to the set of tuples returned after a sliding window execution over the stream. To the best of our knowledge no previous work exists that proposes a stream algebra incorporating both stream and preference operators. A comprehensive survey on continuous processing of skyline, top-k and top-k dominating queries can be found in [8].

Contextual Preferences on Data Streams. A recent work treating contextual preferences in data streams (coming from sensors) is [2]. The authors propose a preliminary and informal methodology described through a real-world example that tries to combine the research topics of context-awareness, data mining and preferences. The paper does not tackles the problem of incorporating the discovered preferences into a query language on sensor data.

8 Conclusion and Future Work

This paper proposes an integrated solution to support user personalized queries in rich data environments involving real time data streams and persistent data and providing powerful querying capabilities. Instantaneous and continuous preference queries are supported. They can benefit of the whole expressivity of Astral and particularly of the large variety of window support (positional, temporal and cross-domain windows) to manage data streams. The contributions of this work include the definition and implementation of preference operators as an extension of Astral. Our experiments allowed to identify patterns of queries (based on the window characteristics) that can be used to decide the best strategy to optimize the query evaluation. Our future research will focus on new optimization approaches and on the distributed evaluation of the preference queries.

Acknowledgment: This work is partially supported by CAPES, the STIC-AmSud ALAP project, CNPq and FAPEMIG and the project BQR Arteco of the Grenoble Institute of Technology. We also would like to thank the Sigma team and M. Echenim of the LIG laboratory for their support.

References

1. A. Arasu, B. Babcock, S. Babu, J. Cieslewicz, and M. Datar. STREAM: The Stanford Data Stream Management System. *Data Stream Management: Processing High-Speed Data Streams*, Jan. 2004.

2. D. Beretta, E. Quintarelli, and E. Rabosio. Mining context-aware preferences on relational and sensor data. In *6th International Workshop on Flexible Database and Information System Technology (FlexDBIST 2011)in conjonction with the 22nd International Conference on Database and Expert Systems Applications (DEXA)*, pages 116–120, 2011.
3. S. Borzsonyi, D. Kossmann, and K. Stocker. The skyline operator. In *Proc. 17th International Conference on Data Engineering (ICDE 2001), Germany*, pages 412–430, 2001.
4. S. de Amo and F. Pereira. Evaluation of conditional preference queries. *Proceedings of the 25th Brazilian Symposium on Databases, October 2010, Belo Horizonte, Brazil. Journal of Information and Data Management (JIDM).*, 1(3):521–536, 2010.
5. S. de Amo and F. Pereira. A context-aware preference query language: Theory and implementation. Technical report, Universidade Federal de Uberlândia, School of Computing, 2011.
6. V. Hristidis, N. Koudas, and Y. Papakonstantinou. Prefer: A system for the efficient execution of multi-parametric ranked queries. In *Proceedings of ACM SIGMOD International Conference on Management of Data, Santa Barbara, CA, USA*, pages 259–270, 2001.
7. W. Kießling and G. Köstler. Preference sql - design, implementation, experiences. In *Proceedings of the Int. Conf. on Very Large Databases*, pages 990–1001, 2002.
8. M. Kontaki, A. Papadopoulos, and Y. Manolopoulos. Continuous processing of preference queries on data streams. In *36th International Conference on Current Trends in Theory and Practice of Computer Science (SOFSEM 2010)*, 2010.
9. M. Kontaki, A. N. Papadopoulos, and Y. Manopoulos. Continuous top k-dominating queries. Technical report, Aristotle University of Thessaloniki, 2009.
10. G. Koutrika, E. Pitoura, and K. Stefanidis. Representation, composition and application of preferences in databases. In *International Conference on Data Engineering (ICDE)*, pages 1214–1215, 2010.
11. K.Stefanidis and E.Pitoura. Fast contextual preference scoring of database tuples. In *Proceedings of the International Conference on Extending Database Technology (EDBT)*, pages 344–355, 2008.
12. M. Morse, J. M. Patel, and W. Grosky. Efficient continuous skyline computation. *Information Sciences*, 177:3411–3437, 2007.
13. K. Mouratidis, S. Bakiras, and D. Papadias. Continuous monitoring of top-k queries over sliding windows. In *Proceedings of SIGMOD*, pages 635–646, 2006.
14. D. Papadias, Y. Tao, G. Fu, and B. Seeger. Progressive skyline computation in database systems. *ACM Transactions on Database Systems*, 30:41–82, 2005.
15. L. Petit, C. Labbé, and C. L. Roncancio. An Algebric Window Model for Data Stream Management. In *Proceedings of the 9th International ACM Workshop on Data Engineering for Wireless and Mobile Access*, pages 17–24. ACM, 2010.
16. L. Petit, C. Labbé, and C. L. Roncancio. Revisiting Formal Ordering in Data Stream Querying. In *Proceedings of the 2012 ACM Symposium on Applied Computing*, New York, NY, USA, 2012. ACM.
17. N. Wilson. Extending cp-nets with stronger conditional preference statements. In *AAAI*, pages 735–741, 2004.