Efficient Algorithms for Processing Preference Queries

Marcos Roberto Ribeiro, UFU/IFMG
Fabiola S. F. Pereira, UFU
Vinicius V. S. Dias, UFMG

ACM SAC, April 2016
Motivation

which destination do you prefer?
Motivation
Motivation

Between two travel options with the same duration, I prefer beach than urban itineraries.
Motivation

Between two travel options with the same duration, I prefer beach than urban itineraries.

Between two travel options with the same price, I prefer cruise than beach itineraries.
Motivation

Between two travel options with the same duration, I prefer beach than urban itineraries.

Between two travel options with the **same price**, I prefer **cruise** than **beach** itineraries.

For **cruise** itineraries, I prefer options costing less than $2500.
<table>
<thead>
<tr>
<th>Destination</th>
<th>Price</th>
<th>Duration</th>
<th>Itinerary</th>
</tr>
</thead>
<tbody>
<tr>
<td>Disney Cruise</td>
<td>2000</td>
<td>4</td>
<td>cruise</td>
</tr>
<tr>
<td>Ibiza</td>
<td>2000</td>
<td>5</td>
<td>beach</td>
</tr>
<tr>
<td>Caribbean</td>
<td>2600</td>
<td>6</td>
<td>cruise</td>
</tr>
<tr>
<td>Dubai</td>
<td>2700</td>
<td>5</td>
<td>urban</td>
</tr>
<tr>
<td>Rio de Janeiro</td>
<td>2600</td>
<td>7</td>
<td>beach</td>
</tr>
</tbody>
</table>

Between two travel options with the same price, I prefer cruise than beach itineraries.

For cruise itineraries, I prefer options costing less than $2500.
Motivation

<table>
<thead>
<tr>
<th>Destination</th>
<th>Price</th>
<th>Duration</th>
<th>Itinerary</th>
</tr>
</thead>
<tbody>
<tr>
<td>Disney Cruise</td>
<td>2000</td>
<td>4</td>
<td>cruise</td>
</tr>
<tr>
<td>Ibiza</td>
<td>2000</td>
<td>5</td>
<td>beach</td>
</tr>
<tr>
<td>Caribbean</td>
<td>2600</td>
<td>6</td>
<td>cruise</td>
</tr>
<tr>
<td>Dubai</td>
<td>2700</td>
<td>5</td>
<td>urban</td>
</tr>
<tr>
<td>Rio de Janeiro</td>
<td>2600</td>
<td>7</td>
<td>beach</td>
</tr>
</tbody>
</table>

CREATE PREFERENCES MyPrefs FROM Travels AS

(I = 'cruise ') > (I = 'beach ') [D, Du] AND
(I = 'beach ') > (I = 'urban ') [P, D] AND
IF (I = 'cruise') THEN (P < 2500) > (P >= 2500) [D,Du];
Motivation

CREATE PREFERENCES MyPrefs FROM Travels AS

(I = 'cruise ') > (I = 'beach ') [D, Du] AND
(I = 'beach ') > (I = 'urban ') [P, D] AND
IF (I = 'cruise') THEN (P < 2500) > (P >= 2500) [D,Du];
Motivation

CREATE PREFERENCES MyPrefs FROM Travels AS
(I = 'cruise ') > (I = 'beach ') [D, Du] AND
(I = 'beach ') > (I = 'urban ') [P, D] AND
IF (I = 'cruise') THEN (P < 2500) > (P >= 2500) [D,Du];

SELECT * FROM Travels
WHERE I <> ' ecological '
ACCORDING TO PREFERENCES 3, MyPrefs;
Motivation

CREATE PREFERENCES MyPrefs FROM Travels AS
(I = 'cruise') > (I = 'beach') [D, Du] AND
(I = 'beach') > (I = 'urban') [P, D] AND
IF (I = 'cruise') THEN (P < 2500) > (P >= 2500) [D,Du];

SELECT * FROM Travels
WHERE I <> 'ecological'
ACCORDING TO PREFERENCES 3, MyPrefs;

CPrefSQL language
(Conditional Preference SQL)
The CPrefSQL Language
SELECT * FROM Travels
WHERE I <> ' ecological '
ACCORDING TO PREFERENCES 3, MyPrefs;
CPrefSQL Language - Syntax

```
SELECT * FROM Travels
    WHERE I <> 'ecological'
ACCORDING TO PREFERENCES 3, MyPrefs;
```

Hard constraints
CPrefSQL Language - Syntax

SELECT * FROM Travels
WHERE I <> ' ecological '

ACCORDING TO PREFERENCES 3, MyPrefs;

Soft constraints
CPrefSQL Language - Query Execution Plan

R1 ⋈ \( \sigma \) \( \pi \) ...

Preference Operator

Rn
CPrefSQL Language - Semantics

✓ Ceteris paribus = “Everything else equal”

✓ Goal: establish an order between tuples

P1: (I = 'cruise ') > (I = 'beach ') [D, Du]
P2: (I = 'beach ') > (I = 'urban ') [P, D]
P3: IF (I = 'cruise') THEN (P < 2500) > (P >= 2500) [D,Du]
### CPrefSQL Language - Semantics

<table>
<thead>
<tr>
<th>Destination</th>
<th>Price</th>
<th>Duration</th>
<th>Itinerary</th>
</tr>
</thead>
<tbody>
<tr>
<td>t1 Disney Cruise</td>
<td>2000</td>
<td>4</td>
<td>cruise</td>
</tr>
<tr>
<td>t2 Ibiza</td>
<td>2000</td>
<td>5</td>
<td>beach</td>
</tr>
<tr>
<td>t3 Caribbean</td>
<td>2600</td>
<td>6</td>
<td>cruise</td>
</tr>
<tr>
<td>t4 Dubai</td>
<td>2700</td>
<td>5</td>
<td>urban</td>
</tr>
<tr>
<td>t5 Rio de Janeiro</td>
<td>2600</td>
<td>7</td>
<td>beach</td>
</tr>
</tbody>
</table>

\[P1: (I = 'cruise ') > (I = 'beach ') \ [D, Du]\]

\[P2: (I = 'beach ') > (I = 'urban ') \ [P, D]\]

\[P3: IF (I = 'cruise') \ THEN \ (P < 2500) > (P >= 2500) \ [D,Du]\]
## CPrefSQL Language - Semantics

<table>
<thead>
<tr>
<th>Destination</th>
<th>Price</th>
<th>Duration</th>
<th>Itinerary</th>
</tr>
</thead>
<tbody>
<tr>
<td>Disney Cruise</td>
<td>2000</td>
<td>4</td>
<td>cruise</td>
</tr>
<tr>
<td>Ibiza</td>
<td>2000</td>
<td>5</td>
<td>beach</td>
</tr>
<tr>
<td>Caribbean</td>
<td>2600</td>
<td>6</td>
<td>cruise</td>
</tr>
<tr>
<td>Dubai</td>
<td>2700</td>
<td>5</td>
<td>urban</td>
</tr>
<tr>
<td>Rio de Janeiro</td>
<td>2600</td>
<td>7</td>
<td>beach</td>
</tr>
</tbody>
</table>

\[
P1: (I = 'cruise ') > (I = 'beach ') \ [D, Du]
\]
\[
P2: (I = 'beach ') > (I = 'urban ') \ [P, D]
\]
\[
P3: IF (I = 'cruise') THEN (P < 2500) > (P >= 2500) \ [D,Du]
\]
### CPrefSQL Language - Semantics

<table>
<thead>
<tr>
<th>Destination</th>
<th>Price</th>
<th>Duration</th>
<th>Itinerary</th>
</tr>
</thead>
<tbody>
<tr>
<td>Disney Cruise</td>
<td>2000</td>
<td>4</td>
<td>cruise</td>
</tr>
<tr>
<td>Ibiza</td>
<td>2000</td>
<td>5</td>
<td>beach</td>
</tr>
<tr>
<td>Caribbean</td>
<td>2600</td>
<td>6</td>
<td>cruise</td>
</tr>
<tr>
<td>Dubai</td>
<td>2700</td>
<td>5</td>
<td>urban</td>
</tr>
<tr>
<td>Rio de Janeiro</td>
<td>2600</td>
<td>7</td>
<td>beach</td>
</tr>
</tbody>
</table>

**Rules:**

- **P1:** \((I = 'cruise') \rightarrow (I = 'beach')\) \([D, Du]\)
- **P2:** \((I = 'beach') \rightarrow (I = 'urban')\) \([P, D]\)
- **P3:** IF \((I = 'cruise')\) THEN \((P < 2500) \rightarrow (P \geq 2500)\) \([D, Du]\)
### CPrefSQL Language - Semantics

<table>
<thead>
<tr>
<th>Destination</th>
<th>Price</th>
<th>Duration</th>
<th>Itinerary</th>
</tr>
</thead>
<tbody>
<tr>
<td>Disney Cruise</td>
<td>2000</td>
<td>4</td>
<td>cruise</td>
</tr>
<tr>
<td>Ibiza</td>
<td>2000</td>
<td>5</td>
<td>beach</td>
</tr>
<tr>
<td>Caribbean</td>
<td>2600</td>
<td>6</td>
<td>cruise</td>
</tr>
<tr>
<td>Dubai</td>
<td>2700</td>
<td>5</td>
<td>urban</td>
</tr>
<tr>
<td>Rio de Janeiro</td>
<td>2600</td>
<td>7</td>
<td>beach</td>
</tr>
</tbody>
</table>

\[ P1: (I = 'cruise ') > (I = 'beach ') \]  [D, Du]
\[ P2: (I = 'beach ') > (I = 'urban ') \]  [P, D]
\[ P3: IF (I = 'cruise') THEN (P < 2500) > (P >= 2500) \]  [D,Du]
### CPrefSQL Language - Semantics

<table>
<thead>
<tr>
<th>Destination</th>
<th>Price</th>
<th>Duration</th>
<th>Itinerary</th>
</tr>
</thead>
<tbody>
<tr>
<td>t1 Disney Cruise</td>
<td>2000</td>
<td>4</td>
<td>cruise</td>
</tr>
<tr>
<td>t2 Ibiza</td>
<td>2000</td>
<td>5</td>
<td>beach</td>
</tr>
<tr>
<td>t3 Caribbean</td>
<td>2600</td>
<td>6</td>
<td>cruise</td>
</tr>
<tr>
<td>t4 Dubai</td>
<td>2700</td>
<td>5</td>
<td>urban</td>
</tr>
<tr>
<td>t5 Rio de Janeiro</td>
<td>2600</td>
<td>7</td>
<td>beach</td>
</tr>
</tbody>
</table>

**P1:** $(I = 'cruise') > (I = 'beach') [D, Du]

**P2:** $(I = 'beach') > (I = 'urban') [P, D]

**P3:** IF $(I = 'cruise')$ THEN $(P < 2500) > (P >= 2500)$ [D, Du]
### CPrefSQL Language - Semantics

<table>
<thead>
<tr>
<th>T</th>
<th>Destination</th>
<th>Price</th>
<th>Duration</th>
<th>Itinerary</th>
</tr>
</thead>
<tbody>
<tr>
<td>t1</td>
<td>Disney Cruise</td>
<td>2000</td>
<td>4</td>
<td>cruise</td>
</tr>
<tr>
<td>t2</td>
<td>Ibiza</td>
<td>2000</td>
<td>5</td>
<td>beach</td>
</tr>
<tr>
<td>t3</td>
<td>Caribbean</td>
<td>2600</td>
<td>6</td>
<td>cruise</td>
</tr>
<tr>
<td>t4</td>
<td>Dubai</td>
<td>2700</td>
<td>5</td>
<td>urban</td>
</tr>
<tr>
<td>t5</td>
<td>Rio de Janeiro</td>
<td>2600</td>
<td>7</td>
<td>beach</td>
</tr>
</tbody>
</table>

**Rule P1:**
\[(I = 'cruise ') > (I = 'beach ') \] [D, Du]

**Rule P2:**
\[(I = 'beach ') > (I = 'urban ') \] [P, D]

**Rule P3:**
\[\text{IF} \ (I = 'cruise') \ \text{THEN} \ (P < 2500) > (P >= 2500) \] [D,Du]

**Diagram:**

- t1 -> t2 -> t3
- t4 -> t5
### CPrefSQL Language - Semantics

<table>
<thead>
<tr>
<th>Destination</th>
<th>Price</th>
<th>Duration</th>
<th>Itinerary</th>
</tr>
</thead>
<tbody>
<tr>
<td>Disney Cruise</td>
<td>2000</td>
<td>4</td>
<td>cruise</td>
</tr>
<tr>
<td>Ibiza</td>
<td>2000</td>
<td>5</td>
<td>beach</td>
</tr>
<tr>
<td>Caribbean</td>
<td>2600</td>
<td>6</td>
<td>cruise</td>
</tr>
<tr>
<td>Dubai</td>
<td>2700</td>
<td>5</td>
<td>urban</td>
</tr>
<tr>
<td>Rio de Janeiro</td>
<td>2600</td>
<td>7</td>
<td>beach</td>
</tr>
</tbody>
</table>

**P1:** \( (I = 'cruise') \) > \( (I = 'beach') \) [D, Du]

**P2:** \( (I = 'beach') \) > \( (I = 'urban') \) [P, D]

**P3:** IF \( (I = 'cruise') \) THEN \( (P < 2500) \) > \( (P \geq 2500) \) [D, Du]
### CPrefSQL Language - Semantics

<table>
<thead>
<tr>
<th>Destination</th>
<th>Price</th>
<th>Duration</th>
<th>Itinerary</th>
</tr>
</thead>
<tbody>
<tr>
<td>t1 Disney Cruise</td>
<td>2000</td>
<td>4</td>
<td>cruise</td>
</tr>
<tr>
<td>t2 Ibiza</td>
<td>2000</td>
<td>5</td>
<td>beach</td>
</tr>
<tr>
<td>t3 Caribbean</td>
<td>2600</td>
<td>6</td>
<td>cruise</td>
</tr>
<tr>
<td>t4 Dubai</td>
<td>2700</td>
<td>5</td>
<td>urban</td>
</tr>
<tr>
<td>t5 Rio de Janeiro</td>
<td>2600</td>
<td>7</td>
<td>beach</td>
</tr>
</tbody>
</table>

P1: \((I = 'cruise ') > (I = 'beach ')\) [D, Du]

P2: \((I = 'beach ') > (I = 'urban ')\) [P, D]

P3: IF \((I = 'cruise')\) THEN \((P < 2500) > (P >= 2500)\) [D,Du]
### CPrefSQL Language - Semantics

**Table:**

<table>
<thead>
<tr>
<th>Destination</th>
<th>Price</th>
<th>Duration</th>
<th>Itinerary</th>
</tr>
</thead>
<tbody>
<tr>
<td>Disney Cruise</td>
<td>2000</td>
<td>4</td>
<td>cruise</td>
</tr>
<tr>
<td>Ibiza</td>
<td>2000</td>
<td>5</td>
<td>beach</td>
</tr>
<tr>
<td>Caribbean</td>
<td>2600</td>
<td>6</td>
<td>cruise</td>
</tr>
<tr>
<td>Dubai</td>
<td>2700</td>
<td>5</td>
<td>urban</td>
</tr>
<tr>
<td>Rio de Janeiro</td>
<td>2600</td>
<td>7</td>
<td>beach</td>
</tr>
</tbody>
</table>

**Relations:**

- **P1:** $(I = 'cruise ') > (I = 'beach ') [D, Du]
- **P2:** $(I = 'beach ') > (I = 'urban ') [P, D]
- **P3:** IF $(I = 'cruise') THEN $(P < 2500) > (P >= 2500) [D,Du]
### CPrefSQL Language - Semantics

<table>
<thead>
<tr>
<th>Destination</th>
<th>Price</th>
<th>Duration</th>
<th>Itinerary</th>
</tr>
</thead>
<tbody>
<tr>
<td>t1 Disney Cruise</td>
<td>2000</td>
<td>4</td>
<td>cruise</td>
</tr>
<tr>
<td>t2 Ibiza</td>
<td>2000</td>
<td>5</td>
<td>beach</td>
</tr>
<tr>
<td>t3 Caribbean</td>
<td>2600</td>
<td>6</td>
<td>cruise</td>
</tr>
<tr>
<td>t4 Dubai</td>
<td>2700</td>
<td>5</td>
<td>urban</td>
</tr>
<tr>
<td>t5 Rio de Janeiro</td>
<td>2600</td>
<td>7</td>
<td>beach</td>
</tr>
</tbody>
</table>

P1: \((I = 'cruise') > (I = 'beach')\) [D, Du]

P2: \((I = 'beach') > (I = 'urban')\) [P, D]

P3: IF \((I = 'cruise')\) THEN \((P < 2500) > (P >= 2500)\) [D, Du]
### CPrefSQL Language - Semantics

<table>
<thead>
<tr>
<th>Destination</th>
<th>Price</th>
<th>Duration</th>
<th>Itinerary</th>
</tr>
</thead>
<tbody>
<tr>
<td>t1 Disney Cruise</td>
<td>2000</td>
<td>4</td>
<td>cruise</td>
</tr>
<tr>
<td>t2 Ibiza</td>
<td>2000</td>
<td>5</td>
<td>beach</td>
</tr>
<tr>
<td>t3 Caribbean</td>
<td>2600</td>
<td>6</td>
<td>cruise</td>
</tr>
<tr>
<td>t4 Dubai</td>
<td>2700</td>
<td>5</td>
<td>urban</td>
</tr>
<tr>
<td>t5 Rio de Janeiro</td>
<td>2600</td>
<td>7</td>
<td>beach</td>
</tr>
</tbody>
</table>

P1: (I = 'cruise') \(>\) (I = 'beach') \([D, Du]\)

P2: (I = 'beach') \(>\) (I = 'urban') \([P, D]\)

P3: IF (I = 'cruise') THEN (P < 2500) \(>\) (P \(\geq\) 2500) \([D, Du]\)
## CPrefSQL Language - Semantics

<table>
<thead>
<tr>
<th></th>
<th>Destination</th>
<th>Price</th>
<th>Duration</th>
<th>Itinerary</th>
</tr>
</thead>
<tbody>
<tr>
<td>t1</td>
<td>Disney Cruise</td>
<td>2000</td>
<td>4</td>
<td>cruise</td>
</tr>
<tr>
<td>t2</td>
<td>Ibiza</td>
<td>2000</td>
<td>5</td>
<td>beach</td>
</tr>
<tr>
<td>t3</td>
<td>Caribbean</td>
<td>2600</td>
<td>6</td>
<td>cruise</td>
</tr>
<tr>
<td>t4</td>
<td>Dubai</td>
<td>2700</td>
<td>5</td>
<td>urban</td>
</tr>
<tr>
<td>t5</td>
<td>Rio de Janeiro</td>
<td>2600</td>
<td>7</td>
<td>beach</td>
</tr>
</tbody>
</table>

\[
P1: (I = 'cruise') > (I = 'beach') [D, Du] \\
P2: (I = 'beach') > (I = 'urban') [P, D] \\
P3: IF (I = 'cruise') THEN (P < 2500) > (P >= 2500) [D, Du]
\]
How to compute the preferred tuples?
Algorithms for Processing Preference Queries
Algorithms for CP-Queries

<table>
<thead>
<tr>
<th></th>
<th>Destination</th>
<th>Price</th>
<th>Duration</th>
<th>Itinerary</th>
</tr>
</thead>
<tbody>
<tr>
<td>t1</td>
<td>Disney Cruise</td>
<td>2000</td>
<td>4</td>
<td>cruise</td>
</tr>
<tr>
<td>t2</td>
<td>Ibiza</td>
<td>2000</td>
<td>5</td>
<td>beach</td>
</tr>
<tr>
<td>t3</td>
<td>Caribbean</td>
<td>2600</td>
<td>6</td>
<td>cruise</td>
</tr>
<tr>
<td>t4</td>
<td>Dubai</td>
<td>2700</td>
<td>5</td>
<td>urban</td>
</tr>
<tr>
<td>t5</td>
<td>Rio de Janeiro</td>
<td>2600</td>
<td>7</td>
<td>beach</td>
</tr>
</tbody>
</table>

\[ P1: (I = 'cruise') > (I = 'beach') [D, Du] \]

\[ P2: (I = 'beach') > (I = 'urban') [P, D] \]

\[ P3: IF (I = 'cruise') THEN (P < 2500) > (P >= 2500) [D, Du] \]
<table>
<thead>
<tr>
<th></th>
<th>Destination</th>
<th>Price</th>
<th>Duration</th>
<th>Itinerary</th>
</tr>
</thead>
<tbody>
<tr>
<td>t1</td>
<td>Disney Cruise</td>
<td>2000</td>
<td>4</td>
<td>cruise</td>
</tr>
<tr>
<td>t2</td>
<td>Ibiza</td>
<td>2000</td>
<td>5</td>
<td>beach</td>
</tr>
<tr>
<td>t3</td>
<td>Caribbean</td>
<td>2600</td>
<td>6</td>
<td>cruise</td>
</tr>
<tr>
<td>t4</td>
<td>Dubai</td>
<td>2700</td>
<td>5</td>
<td>urban</td>
</tr>
<tr>
<td>t5</td>
<td>Rio de Janeiro</td>
<td>2600</td>
<td>7</td>
<td>beach</td>
</tr>
</tbody>
</table>
Scan the Set of preference rules

P1: (I = 'cruise ') > (I = 'beach ') [D, Du]
P2: (I = 'beach ') > (I = 'urban ') [P, D]
P3: IF (I = 'cruise') THEN (P < 2500) > (P >= 2500) [D,Du]
The dominance test (given 2 tuples, which one is preferred?)
Algorithms for CP-Queries

Recursive SQL (Baseline)

Block nested loops [Pereira et al 2010]

SQL

BNL Algorithms

Partition Algorithms

Partitions
CREATE OR REPLACE VIEW Rules (D, P, Du, I, des, pri, dur, iti) AS
    SELECT * FROM Travels T, Travels T1
UNION
    SELECT * FROM Travels T, Travels T1
    WHERE T.Du < 4 AND T1.Du < 4 AND T.I = 'urban'
    AND T1.I = 'cruise' AND T.P = T1.P
UNION
    SELECT * FROM Travels T, Travels T1
    WHERE T.I = 'cruise' AND T1.I = 'beach' AND T.P = T1.P
    AND T.Du = T1.Du;

WITH RECURSIVE Recursion (des, pri, dur, iti, D, P, Du, I) AS (
    SELECT * FROM Rules
    UNION
    FROM Rules T, Recursion R
    WHERE T.des = R.des AND T.pri = R.pri AND T.dur = R.dur
    AND T.iti = R.iti)

SELECT * FROM Travels WHERE I <> 'ecological'
EXCEPT
FROM Recursion R;
Recursive SQL - Baseline [Pereira et al 2010]

```sql
CREATE OR REPLACE VIEW Rules (D, P, Du, I, des, pri, dur, iti) AS
SELECT *
FROM Travels T, Travels T1
UNION
SELECT *
FROM Travels T, Travels T1
WHERE T.Du < 4 AND T1.Du < 4 AND T.I = 'urban'
    AND T1.I = 'cruise' AND T.P = T1.P
UNION
SELECT *
FROM Travels T, Travels T1
WHERE T.I = 'cruise' AND T1.I = 'beach' AND T.P = T1.P
    AND T.Du = T1.Du;

WITH RECURSIVE Recursion (des, pri, dur, iti, D, P, Du, I) AS
( SELECT * FROM Rules
UNION
FROM Rules T, Recursion R
WHERE T.des = R.des AND T.pri = R.pri AND T.dur = R.dur
    AND T.iti = R.iti)

SELECT *
FROM Travels
WHERE I <> 'ecological'
EXCEPT
FROM Recursion R;
```

6. EXPERIMENTAL RESULTS

All the experiments have been performed on a PC with the following setup: 12x Intel(R) Core(TM) i7 CPU @ 3.20GHz, 32GB of main memory, HD of 3TB 7200rpm and Ubuntu 14.04 64bits OS. We have used `psql` as front-end to PostgreSQL 9.2, connecting to the server on a local host to handle the queries. The implementations under experiments showed here were developed as an on-top approach, which means that preference operators implementations are treated as functions and do not modify the RDBMS source code. All algorithms were implemented in Python.

6.1. Experimental Environment

Algorithms. In our experiments we compare the performance and scalability of the following algorithms: (1) X-BNL and XRank-BNL with DatalogDomTest, (2) X-BNL and XRank-BNL with KBDomTest (X-BNL KB and XRank-BNL KB), (3) X-Partition and XRank-Partition and (4) queries written in SQL – our baseline.
As remarked in Section 4, the new operators Select-Best and SelectK-Best do not increase the expressive power of SQL, since they can be translated into a SQL query. As a matter of fact, any operator designed to produce the non-dominated tuples can be expressed in SQL ([Chomicki 2003]). However, unlike the skyline queries and the pareto queries which can be expressed in the standard SQL-92, the cp-queries go beyound the expressive power of classical relational algebra, necessarily involving recursion in their specification. That is due to the transitive closure operation involved in the definition of the preference ordering associated to a cp-theory (this operation is not needed in the skyline and pareto queries as explained in Section 2). The SQL:99 command corresponding to the top-k cp-query of Example 1.1 is described in Figure 5.

```
CREATE OR REPLACE VIEW Rules (D, P, Du, I, des, pri, dur, iti) AS
  SELECT * FROM Travels T, Travels T1
UNION
  SELECT * FROM Travels T, Travels T1
  WHERE T.Du < 4 AND T1.Du < 4 AND T.I = 'urban'
    AND T1.I = 'cruise' AND T.P = T1.P
UNION
  SELECT * FROM Travels T, Travels T1
  WHERE T.I = 'cruise' AND T1.I = 'beach' AND T.P = T1.P
    AND T.Du = T1.Du;
WITH RECURSIVE Recursion (des, pri, dur, iti, D, P, Du, I) AS
  (SELECT * FROM Rules
UNION
  FROM Rules T, Recursion R
  WHERE T.des = R.des AND T.pri = R.pri AND T.dur = R.dur
    AND T.iti = R.iti)
SELECT * FROM Travels WHERE I <> 'ecological'
EXCEPT
FROM Recursion R;
```

As remarked in Section 4, the new operators \texttt{Select-Best} and \texttt{SelectK-Best} do not increase the expressive power of SQL, since they can be translated into a SQL query. As a matter of fact, any operator designed to produce the non-dominated tuples can be expressed in SQL ([Chomicki 2003]). However, unlike the skyline queries and the pareto queries which can be expressed in the standard SQL-92, the cp-queries go beyond the expressive power of classical relational algebra, necessarily involving recursion in their specification. That is due to the transitive closure operation involved in the definition of the preference ordering associated to a cp-theory (this operation is not needed in the skyline and pareto queries as explained in Section 2). The SQL:99 command corresponding to the top-k cp-query of Example 1.1 is described in Figure 5.

We have applied this classical approach as the baseline to compare with our proposed algorithms.

\begin{verbatim}
CREATE OR REPLACE VIEW Rules (D, P, Du, I, des, pri, dur, iti) AS 
SELECT * FROM Travels T, Travels T1 
UNION 
SELECT * FROM Travels T, Travels T1 
WHERE T.Du < 4 AND T1.Du < 4 AND T.I = 'urban' 
AND T1.I = 'cruise' AND T.P = T1.P 
UNION 
SELECT * FROM Travels T, Travels T1 
WHERE T.I = 'cruise' AND T1.I = 'beach' AND T.P = T1.P 
AND T.Du = T1.Du;

WITH RECURSIVE Recursion (des, pri, dur, iti, D, P, Du, I) AS ( 
SELECT * FROM Rules 
UNION 
FROM Rules T, Recursion R 
WHERE T.des = R.des AND T.pri = R.pri AND T.dur = R.dur 
AND T.iti = R.iti) 
SELECT * FROM Travels WHERE I <> 'ecological' 
EXCEPT 
FROM Recursion R;
\end{verbatim}
Recursive SQL - Baseline [Pereira et al 2010]

✓ Verbose

✓ Without specific preference operators

✓ High computational cost
Algorithms for CP-Queries

Recursive SQL (Baseline)

SQL

BNL Algorithms

Block nested loops [Pereira et al 2010]

Partition Algorithms

Partitions
Block Nested Loop [Pereira et al 2010]

Input table:
- t1 t5
- t2 t6
- t3 t7
- t4 t8

Buffer

Temporary table

Output table
Block Nested Loop [Pereira et al 2010]

- **Input table**: t4, t5, t6, t7, t8
- **Buffer**: t1, t2, t3
- **Temporary table**
- **Output table**
Block Nested Loop [Pereira et al 2010]

Datalog Program

Input table

Buffer

Output table

Temporary table

\[
\begin{align*}
t_1 & > t_2? \\
t_1 & > t_3? \\
t_2 & > t_3? \\
\end{align*}
\]
Block Nested Loop [Pereira et al 2010]

- Input table: t4, t5, t6, t7, t8
- Buffer: t1, t2
- Temporary table
- Discarding dominated tuples: t3, t3 (marked with an 'X')
- Output table
Block Nested Loop [Pereira et al 2010]

- Input table: t5, t6, t7, t8
- Buffer: t1, t2, t4
- Temporary table
- Output table
Block NeSted Loop [Pereira et al 2010]

Tuples indifferent of buffer tuples that do not fit in buffer

Input table

Buffer
  - t1
  - t2
  - t4

Temporary table

Output table
Block Nested Loop [Pereira et al 2010]

- **Input table**
- **Buffer**
- **Temporary table**
  - t5
  - t6
  - t7
  - t8
- **Dominant tuples**
  - t1
  - t2
  - t4
- **Output table**
Block Nested Loop [Pereira et al 2010]
Block Nested Loop [Pereira et al 2010]

- Input table
- Buffer
- Temporary table
- Output table
Algorithms for CP-Queries

Recursive SQL (Baseline)

SQL

BNL Algorithms

Block nested loops [Pereira et al 2010]

Partition Algorithms

Partitions
Preference Rules

\[ P_1: (I = 'cruise') > (I = 'beach') \]
\[ [D, Du] \]
\[ P_2: (I = 'beach') > (I = 'urban') \]
\[ [P, D] \]
\[ P_3: \text{IF} \ (I = 'cruise') \ \text{THEN} \ (P < 2500) > (P >= 2500) \]
\[ [D, Du] \]
Partition Algorithms

Preference Rules

P1: (I = 'cruise ') > (I = 'beach ')
[D, Du]
P2: (I = 'beach ') > (I = 'urban ')
[P, D]
P3: IF (I = 'cruise') THEN (P < 2500) > (P >= 2500) [D,Du]


#### Preference Rules

- **P1:** \( I = 'cruise' \) > \( I = 'beach' \) \[D, Du\]
- **P2:** \( I = 'beach' \) > \( I = 'urban' \) \[P, D\]
- **P3:** IF \( I = 'cruise' \) THEN \( P < 2500 \) > \( P >= 2500 \) \[D, Du\]
Partition Algorithms

Preference Rules

P1: (I = 'cruise ') > (I = 'beach ')
[D, Du]
P2: (I = 'beach ') > (I = 'urban ')
[P, D]
P3: IF (I = 'cruise') THEN (P < 2500) > (P >= 2500) [D,Du]

Knowledge Base

Partitions (Hash Index)

Database

t1
t2
t3
t4
t5
Partition Algorithms

\[ b_1: (P < 2300)[Du,D] > (P \geq 2300)[Du,D] \]
\[ S = \{t_1, t_2, t_3, t_4, t_5, t_6\} \]

Partitions attributes:
\[ \text{Attr}(r) - \text{Attr}(b) = \{ 1 \} \]
Partition Algorithms

\[ b_1: (P < 2300)[Du,D] > (P \geq 2300)[Du,D] \]
\[ S = \{t_1, t_2, t_3, t_4, t_5, t_6\} \]

Partitions attributes:
Attr(r) - Attr(b) = \{1\}

PARTITION P1
(I = beach)
\[ t_1 \]
\[ t_3 \]
\[ t_1 \sim_{b_1} t_3 \]

PARTITION P2
(I = cruise)
\[ t_2 \]
\[ t_4 \]

PARTITION P3
(I = urban)
\[ t_5 \]
\[ t_6 \]
\[ t_5 \sim_{b_1} t_6 \]
Partition Algorithms

Partition Algorithms

**Algorithm 4**

**MostPref** \((r, c)\): 

**Input:** a set \(r\) of tuples and a cp-theory \(c\)  

**Output:** a set \(S\) containing the most preferred tuples of \(r\)

1. Clear the in-memory page \(W\) and the temporary table \(F\).
2. Make \(r\) the input.
3. While the input is not empty, do:
   4. For all tuple \(t\) in the input, do:
      5. \(\text{dominate} = -1\)
      6. For all tuple \(t_0\) in \(W\), do:
         7. \(\text{dominanceTest}(t, t_0)\)
         8. If \(t\) is dominated by \(t_0\), then:
            9. \(\text{dominate} = 0\)
            10. Break
         11. If \(t\) dominates \(t_0\), then:
            12. Eliminate \(t_0\) from \(W\)
            13. \(\text{dominate} = 1\)
            14. If \(\text{dominate} = 1\), then:
                15. Insert \(t\) into \(W\)
            16. Else if \(\text{dominate} = -1\), then:
                17. Insert \(t\) into \(W\) if there is room, otherwise add to \(F\)
18. Insert in \(S\) the tuples of \(W\) which were added there when \(F\) was empty.
19. Make \(F\) the input, clear the temporary table.
20. Return \(S\).

**Fig. 4:** Snapshots from runs of preference queries algorithms (a) X-BNL and (b) X-Partition

The complexity of X-BNL where the dominance test is a datalog program is \(O(kn^2m^5)\) (in the worst case), where \(n\) is the size of the input instance \(r\) and \(m\) is the number of rules in the cp-theory. If the knowledge base is considered the complexity of X-BNL is \(O(k^2n^2m^5)\) (in the worst case), where \(k\) is the number of attributes appearing in the rules of the cp-theory.
Partition Algorithms

B1: (I=cruse) > (I=urban) [D,Du,I,P]
Partition Algorithms

B1: (I=cruise) > (I=urban) [D, Du, I, P]

B1: t1, t2, t3, t4, t5
Partition Algorithms

B1: \((I=\text{cruise}) > (I=\text{urban})\) \([D,Du,I,P]\)

Discarding dominated tuples
**Partition Algorithms**

B1: \( (I=\text{cruise}) \, \rightarrow \, (I=\text{urban}) \) \( [D,Du,I,P] \)

\[ t_1, t_2, t_3, t_5 \]
Partition Algorithms

B1:
Partition Algorithms

B2: (I=cruise) ∧ (P<2500) > (I=beach) ∧ (P≥2500) [D,Du,I,P]
Partition Algorithms

B2: (I=cruise) ∧ (P<2500) > (I=beach) ∧ (P≥2500) [D,Du,I,P]
Partition Algorithms

B2: (I=cruise) ∧ (P<2500) > (I=beach) ∧ (P≥2500) [D,Du,I,P]

Discarding dominated tuples
Partition Algorithms

B2: (I=cruise) ∧ (P<2500) > (I=beach) ∧ (P≥2500) [D,Du,I,P]
Partition Algorithms

B2:

t1
t2
t3
Partition Algorithms

B3: (I=beach) > (I=urban) [D,I,P]

B3: Du=4  Du=5  Du=6
Partition Algorithms

B3: (I=beach) > (I=urban) [D, I, P]
Partition Algorithms

B3: (I=beach) > (I=urban) [D, I, P]

B3: t1 t2 t3

Nothing to do
Partition Algorithms

B4: (I=cruise) > (I=beach) [D,I,Du]

B4: \( P < 2500 \) \( P \geq 2500 \)

t1
t2
t3
Partition Algorithms

\[ B5: (I=\text{cruise}) \land (P<2500) \succ (I=\text{cruise}) \land (P \geq 2500) [D, Du, P] \]
Partition Algorithms

The preferred tuple!

t1
Partition Algorithms

Knowledge Base

Partitions

t1  t5
 t2  t6
t3  t7
t4  t8
Complexity Analysis
Complexity Analysis

BNL Algorithms

$O(n^2 \times m^m)$

- Number of tuples
- Number of preference rules
Complexity Analysis

BNL Algorithms

\[ O(n^2 \times m^m) \]

- Number of tuples
- Number of preference rules

Partition Algorithms

\[ O(n) + O(\text{KB}) \]

- Number of tuples
- Cost of building the knowledge base
Experimental Results
4. EXPERIMENTAL RESULTS

In practice, the number of input tuples causes the biggest impact on the algorithms complexity because all algorithms have a cost of nested loops over the input tuples. In the case of BNL**, BNL-KB and Partition, respectively, multiplied by the building cost and size of the knowledge base. The procedure to build the knowledge base is responsible for the number of dominance test calls. Summarizing, all parameters and their variations are shown in Table 1.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Default</th>
<th>Variation</th>
</tr>
</thead>
<tbody>
<tr>
<td>Dataset (MB)</td>
<td>32</td>
<td>16, 32, 64, 128, 256, 512 and 1024</td>
</tr>
<tr>
<td>Depth level</td>
<td>2</td>
<td>1, 2, 3, 4, 5 and 6</td>
</tr>
<tr>
<td>Number of rules</td>
<td>6</td>
<td>6, 10, 20, 30 and 40</td>
</tr>
<tr>
<td>Query</td>
<td>$Q_5$</td>
<td>$Q_3$, $Q_5$, $Q_{10}$ and $Q_{18}$</td>
</tr>
<tr>
<td>k</td>
<td>-1</td>
<td>-1, 10, 100, 1000 and 5000</td>
</tr>
</tbody>
</table>

Preferences.

Dataset

TPC-H Benchmark

TPC-H Benchmark

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Default</th>
<th>Variation</th>
</tr>
</thead>
<tbody>
<tr>
<td>Dataset (MB)</td>
<td>32</td>
<td>16, 32, 64, 128, 256, 512 and 1024</td>
</tr>
<tr>
<td>Depth level</td>
<td>2</td>
<td>1, 2, 3, 4, 5 and 6</td>
</tr>
<tr>
<td>Number of rules</td>
<td>6</td>
<td>6, 10, 20, 30 and 40</td>
</tr>
<tr>
<td>Query</td>
<td>$Q_5$</td>
<td>$Q_3$, $Q_5$, $Q_{10}$ and $Q_{18}$</td>
</tr>
<tr>
<td>k</td>
<td>-1</td>
<td>-1, 10, 100, 1000 and 5000</td>
</tr>
</tbody>
</table>
We intend to follow three main directions of research in the general preference rules. The preliminary CPrefSQL version [15] by considering more algorithms. We have also enhanced the syntax and semantics of some built-in base preference constructors and accumulated some built-in base preference constructors and accumulated in the cp-theory have a considerable negative impact on BNL** algorithms and BNL-KB performances. On the other hand, the performance of Partition algorithm has not been strongly affected by transitivity between BNL algorithm and Partition algorithm. The bigger databases have more tuples and this causes more complexity between BNL algorithm and Partition algorithm.

Scalability results.

To be processed. The number of tuples directly submitted to the preference operator. Again, the performance of Partition algorithm is far better than BNL algorithms because the number of tuples to be processed di

Results – Varying Number of Rules

![Graph showing Execution Time (sec) vs Number of rules for different algorithms: Recursive SQL, BNL, and Partition.](image)
The topic of preference query evaluation has been extensively studied. The new algorithms have been implemented following the logical framework of preference specification introduced in [4, 2]. In this paper, we follow the preliminary CPrefSQL version [15] by considering more general conditional preference semantics [8].

The research literature on preference models and extension constructors. The optimizer uses a rewriting procedure which transforms preference queries into standard SQL queries. Regarding extensions of SQL, the research of [13] introduces CP-Net and TCP-Net approaches. We have also tested the performance of cp-queries by varying the number of rules, that is, the number of preferred tuples (the desired amount of preferred tuples) as for different levels.

Figure 5(c) shows how algorithms vary the reduction factor of the parameter $k$ for $k > 1$ (in this case the algorithms return the most similar tuples). Since for $k = 1$ the algorithms return the most different tuples.

Figure 5(d) shows how algorithms compete for the same object simultaneously. The authors proposed the algorithm BNL for pareto query evaluation. The algorithm BNL has been proposed which outperforms the BNL algorithm for skyline queries. The topic of preference query evaluation has been addressed, where different algorithms need extra time to handle different levels of depth.

Figure 5(e) shows the behavior of algorithms when varying the reduction factor of the reduction factor. The bigger databases have more tuples and this causes more complexity between BNL algorithm and Partition algorithm. The performance of Partition algorithm has not been strongly affected by the presence of non-deterministic reductions. On the other hand, the performance of both BNL algorithms and BNL-KB performances. In Figure 5(a) we can see the behavior of algorithms when the size of the database increases from 16MB to 1024MB. This experiment evidences the different behavior of algorithms when the size of the database increases from 16MB to 1024MB.
Results - Varying Database Size

![Graph showing execution time vs database size for different algorithms](image)

- Recursive SQL
- BNL
- Partition

The graph illustrates the execution time (in seconds) for different database sizes (in MB). The y-axis represents the execution time, ranging from $10^{-1}$ to $10^9$, while the x-axis represents the database size from 200 to 1000 MB.

The graph shows the performance comparison of Recursive SQL, BNL, and Partition algorithms. Recursion SQL has a higher execution time compared to BNL and Partition, especially as the database size increases. The Partition algorithm shows a more linear growth in execution time with respect to the database size, indicating better scalability.

**Legend**
- Recursive SQL
- BNL
- Partition
Results - Varying Query

![Graph showing results with queries Q3, Q5, Q10, Q18]
Conclusions
Conclusions

Conditional Preference Queries
Conclusions

Conditional Preference Queries

SQL → BNL Algorithms
Conclusions

Conditional Preference Queries

SQL → BNL Algorithms → Partition Algorithms
Conclusions

Conditional Preference Queries

SQL → BNL Algorithms → Partition Algorithms → Continuous Queries
Acknowledgements
Efficient Algorithms for Processing Preference Queries

Marcos Roberto Ribeiro, UFU/IFMG
Fabiola S. F. Pereira, UFU
Vinicius V. S. Dias, UFMG
### Datalog Program

\[
\text{pref}(x_1, \ldots, x_n, y_1, \ldots, y_n) \leftarrow x_{p_1} \models P_1(A_1), y_{p_1} = x_{p_1}, \ldots, x_{p_k} \models P_k(A_k), y_{p_k} = x_{p_k},
\]
\[
x_j \models Q_1(X), y_j \models Q_2(X),
\]
\[
x_{l_1} = y_{l_1}, \ldots, x_{l_m} = y_{l_m}.
\]

\[
\text{dominate}(x_1, \ldots, x_n, y_1, \ldots, y_n) \leftarrow \text{pref}(x_1, \ldots, x_n, y_1, \ldots, y_n).
\]
\[
\text{dominate}(x_1, \ldots, x_n, y_1, \ldots, y_n) \leftarrow \text{pref}(x_1, \ldots, x_n, z_1, \ldots, z_n),
\]
\[
\text{dominate}(z_1, \ldots, z_n, y_1, \ldots, y_n).
\]
Partition Algorithms - Building the KB

P1: (I = 'cruise ') > (I = 'beach ') [D, Du]
P2: (I = 'beach ') > (I = 'urban ') [P, D]
P3: IF (I = 'cruise') THEN (P < 2500) > (P >= 2500) [D,Du]

Generating essential formulas
Partition Algorithms - Building the KB

P1: \( I = 'cruise' \) > \( I = 'beach' \) [D, Du]

P2: \( I = 'beach' \) > \( I = 'urban' \) [P, D]

P3: IF \( I = 'cruise' \) THEN \( P < 2500 \) > \( P \geq 2500 \) [D, Du]

Generating essential formulas

\[
(I = cruise), (I = beach), (I = urban), (P < 2500), \\
(P \geq 2500), (I = cruise) \land (P < 2500), \\
(I = beach) \land (P < 2500), (I = urban) \land (P < 2500), \\
(I = cruise) \land (P \geq 2500), (I = beach) \land (P \geq 2500), \\
(I = urban) \land (P \geq 2500)
\]
Partition Algorithms - Building the KB

Preferences

P1: (I = 'cruise') > (I = 'beach') [D, Du]
P2: (I = 'beach') > (I = 'urban') [P, D]
P3: IF (I = 'cruise') THEN (P < 2500) > (P ≥ 2500) [D, Du]

Essential Formulas

(I = cruise), (I = beach), (I = urban), (P < 2500), (P ≥ 2500),
(I = cruise) ∧ (P < 2500), (I = beach) ∧ (P < 2500),
(I = urban) ∧ (P ≥ 2500), (I = beach) ∧ (P ≥ 2500),
(I = urban) ∧ (P ≥ 2500)
Partition Algorithms - Building the KB

Preferences

P1: (I = 'cruise') > (I = 'beach') [D, Du]
P2: (I = 'beach') > (I = 'urban') [P, D]
P3: IF (I = 'cruise') THEN (P < 2500) > (P >= 2500) [D,Du]

Essential Formulas

(I = cruise), (I = beach), (I = urban), (P < 2500), (P ≥ 2500), (I = cruise) ∧ (P < 2500), (I = beach) ∧ (P < 2500), (I = urban) ∧ (P < 2500), (I = cruise) ∧ (P ≥ 2500), (I = beach) ∧ (P ≥ 2500), (I = urban) ∧ (P ≥ 2500)

Generating the Knowledge Base

B1: (I=cruise) > (I=urban)[D,Du,I,P]
B2: (I=cruise) ∧ (P<2500) > (I=beach) ∧ (P≥2500)[D,Du,I,P]
B3: (I=beach) > (I=urban)[D,I,P]
B4: (I=cruise) > (I=beach)[D,I,Du]
B5: (I=cruise) ∧ (P<2500) > (I=cruise) ∧ (P≥2500)[D,Du,P]
Partition Algorithms

B1: (I=cruise) > (I=urban) [D,Du,I,P]
B2: (I=cruise) ∧ (P<2500) > (I=beach) ∧ (P≥2500) [D,Du,I,P]
B3: (I=beach) > (I=urban) [D,I,P]
B4: (I=cruise) > (I=beach) [D,I,Du]
B5: (I=cruise) ∧ (P<2500) > (I=cruise) ∧ (P≥2500) [D,Du,P]

B3: Du=4  Du=5  Du=6
Partition Algorithms

B1: \((I=\text{cruise}) > (I=\text{urban}) [D,Du,I,P]\)

B2: \((I=\text{cruise}) \land (P<2500) > (I=\text{beach}) \land (P\geq 2500) [D,Du,I,P]\)

B3: \((I=\text{beach}) > (I=\text{urban}) [D,I,P]\)

B4: \((I=\text{cruise}) > (I=\text{beach}) [D,I,Du]\)

B5: \((I=\text{cruise}) \land (P<2500) > (I=\text{cruise}) \land (P\geq 2500) [D,Du,P]\)
Partition Algorithms

B1: (I=cruise) > (I=urban) [D,Du,I,P]
B2: (I=cruise) ∧ (P<2500) > (I=beach) ∧ (P≥2500) [D,Du,I,P]
B3: (I=beach) > (I=urban) [D,I,P]
B4: (I=cruise) > (I=beach) [D,I,Du]
B5: (I=cruise) ∧ (P<2500) > (I=cruise) ∧ (P≥2500) [D,Du,P]