Foundations of Preference Queries

Jan Chomicki
University at Buffalo

Plan of the course

1. Preference relations
2. Preference queries
3. Preference management
4. Advanced topics

Part I

Preference relations

Outline of Part I

1. Preference relations
   - Preference
   - Equivalence
   - Preference specification
   - Combining preferences
   - Skylines

Preference relations

Universe of objects

- constants: uninterpreted, numbers,...
- individuals (entities)
- tuples
- sets

Preference relations

Universe of objects

- constants: uninterpreted, numbers,...
- individuals (entities)
- tuples
- sets

Preference relation $\succ$

- binary relation between objects
- $x \succ y \equiv x$ *is_better_than* $y \equiv x$ dominates y
- an abstract, uniform way of talking about desirability, worth, cost, timeliness,..., and their combinations
- preference relations used in queries

Buying a car

Buying a car

Salesman: What kind of car do you prefer?

Buying a car

Salesman: What kind of car do you prefer?
Customer: The newer the better, if it is the same make. And cheap, too.

Buying a car

Salesman: What kind of car do you prefer?
Customer: The newer the better, if it is the same make. And cheap, too.
Salesman: Which is more important for you: the age or the price?

Buying a car

Salesman: What kind of car do you prefer?
Customer: The newer the better, if it is the same make. And cheap, too.
Salesman: Which is more important for you: the age or the price?
Customer: The age, definitely.

Buying a car

Salesman: What kind of car do you prefer?
Customer: The newer the better, if it is the same make. And cheap, too.
Salesman: Which is more important for you: the age or the price?
Customer: The age, definitely.
Salesman: Those are the best cars, according to your preferences, that we have in stock.

Buying a car

Salesman: What kind of car do you prefer?
Customer: The newer the better, if it is the same make. And cheap, too.
Salesman: Which is more important for you: the age or the price?
Customer: The age, definitely.
Salesman: Those are the best cars, according to your preferences, that we have in stock.
Customer: Wait...it better be a BMW.

Properties of preference relations

Properties of preference relations

Properties of $\succ$

- irreflexivity: $\forall x.\ x \not\succ x$
- asymmetry: $\forall x, y.\ x \succ y \Rightarrow y \not\succ x$
- transitivity: $\forall x, y, z.\ (x \succ y \wedge y \succ z) \Rightarrow x \succ z$
- negative transitivity: $\forall x, y, z.\ (x \not\succ y \wedge y \not\succ z) \Rightarrow x \not\succ z$
- connectivity: $\forall x, y.\ x \succ y \vee y \succ x \vee x = y$

Properties of preference relations

Properties of $\succ$

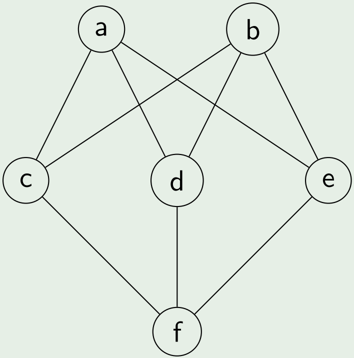
- irreflexivity: $\forall x.\ x \not\succ x$
- asymmetry: $\forall x, y.\ x \succ y \Rightarrow y \not\succ x$
- transitivity: $\forall x, y, z.\ (x \succ y \wedge y \succ z) \Rightarrow x \succ z$
- negative transitivity: $\forall x, y, z.\ (x \not\succ y \wedge y \not\succ z) \Rightarrow x \not\succ z$
- connectivity: $\forall x, y.\ x \succ y \vee y \succ x \vee x = y$

Orders

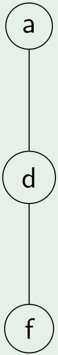
- strict partial order (SPO): irreflexive and transitive
- weak order (WO): negatively transitive SPO
- total order: connected SPO

Weak and total orders

Weak order



Total order



Order properties of preference relations

Order properties of preference relations

Irreflexivity, asymmetry: uncontroversial.

Order properties of preference relations

Irreflexivity, asymmetry: uncontroversial.

Transitivity:

- captures rationality of preference
- not always guaranteed: voting paradoxes
- helps with preference querying

Order properties of preference relations

Irreflexivity, asymmetry: uncontroversial.

Transitivity:

- captures rationality of preference
- not always guaranteed: voting paradoxes
- helps with preference querying

Negative transitivity:

- scoring functions represent weak orders

Order properties of preference relations

Irreflexivity, asymmetry: uncontroversial.

Transitivity:

- captures rationality of preference
- not always guaranteed: voting paradoxes
- helps with preference querying

Negative transitivity:

- scoring functions represent weak orders

We assume that preference relations are SPOs.

When are two objects equivalent?

When are two objects equivalent?

Relation $\sim$

- binary relation between objects
- $x \sim y \equiv x$ *"is equivalent to"* y

When are two objects equivalent?

Relation $\sim$

- binary relation between objects
- $x \sim y \equiv x$ *"is equivalent to"* y

Several notions of equivalence

- equality: $x \sim^{eq} y \;\equiv\; x = y$
- indifference: $x \sim^{i} y \;\equiv\; x \not\succ y \wedge y \not\succ x$
- restricted indifference:
  $x \sim^{r} y \equiv \forall z.\, (x \prec z \Leftrightarrow y \prec z) \wedge (z \prec y \Leftrightarrow z \prec x)$

When are two objects equivalent?

Relation $\sim$

- binary relation between objects
- $x \sim y \equiv x$ *"is equivalent to"* y

Several notions of equivalence

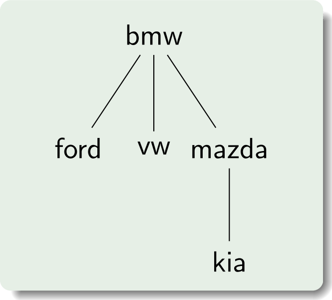
- equality: $x \sim^{eq} y \;\equiv\; x = y$
- indifference: $x \sim^{i} y \;\equiv\; x \not\succ y \wedge y \not\succ x$
- restricted indifference:
  $x \sim^{r} y \equiv \forall z.\ (x \prec z \Leftrightarrow y \prec z) \wedge (z \prec y \Leftrightarrow z \prec x)$

Properties of equivalence

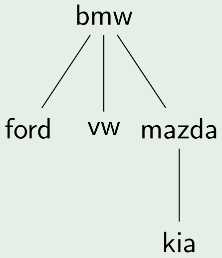
- equivalence relation: reflexive, symmetric, transitive
- equality and restricted indifference (if $\succ$ is an SPO) are equivalence relations
- indifference is reflexive and symmetric; transitive for WO

Example

Example



Example



Preference:

$bmw \succ ford$, $bmw \succ vw$
$bmw \succ mazda$, $bmw \succ kia$
$mazda \succ kia$

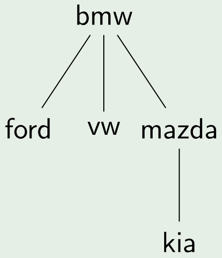
Indifference:

$ford \sim^i vw$, $vw \sim^i ford$,
$ford \sim^i mazda$, $mazda \sim^i ford$,
$vw \sim^i mazda$, $mazda \sim^i vw$,
$ford \sim^i kia$, $kia \sim^i ford$,
$vw \sim^i kia$, $kia \sim^i vw$

Restricted indifference:

$ford \sim^r vw$, $vw \sim^r ford$

Example



This is a strict partial order which is not a weak order.

Preference:

$bmw \succ ford$, $bmw \succ vw$
$bmw \succ mazda$, $bmw \succ kia$
$mazda \succ kia$

Indifference:

$ford \sim^i vw$, $vw \sim^i ford$,
$ford \sim^i mazda$, $mazda \sim^i ford$,
$vw \sim^i mazda$, $mazda \sim^i vw$,
$ford \sim^i kia$, $kia \sim^i ford$,
$vw \sim^i kia$, $kia \sim^i vw$

Restricted indifference:

$ford \sim^r vw$, $vw \sim^r ford$

Not every SPO is a WO

Canonical example

$mazda \succ kia,\ mazda \sim^i vw,\ kia \sim^i vw$

Violation of negative transitivity

$mazda \not\succ vw,\ vw \not\succ kia,\ mazda \succ kia$

Preference specification

Preference specification

Explicit preference relations

Finite sets of pairs: bmw $\succ$ mazda, mazda $\succ$ kia,...

Preference specification

Explicit preference relations

Finite sets of pairs: bmw $\succ$ mazda, mazda $\succ$ kia,...

Implicit preference relations

- can be infinite but finitely representable
- defined using logic formulas in some constraint theory:

Preference specification

Explicit preference relations

Finite sets of pairs: bmw $\succ$ mazda, mazda $\succ$ kia,...

Implicit preference relations

- can be infinite but finitely representable
- defined using logic formulas in some constraint theory:

$$(m_1, y_1, p_1) \succ_1 (m_2, y_2, p_2) \equiv y_1 > y_2 \vee (y_1 = y_2 \wedge p_1 < p_2)$$

for relation $Car(Make, Year, Price)$.

Preference specification

Explicit preference relations

Finite sets of pairs: bmw $\succ$ mazda, mazda $\succ$ kia,...

Implicit preference relations

- can be infinite but finitely representable
- defined using logic formulas in some constraint theory:

$$(m_1, y_1, p_1) \succ_1 (m_2, y_2, p_2) \equiv y_1 > y_2 \vee (y_1 = y_2 \wedge p_1 < p_2)$$

  for relation $Car(Make, Year, Price)$.
- defined using preference constructors (Preference SQL)

Preference specification

Explicit preference relations

Finite sets of pairs: bmw $\succ$ mazda, mazda $\succ$ kia,...

Implicit preference relations

- can be infinite but finitely representable
- defined using logic formulas in some constraint theory:

$$(m_1, y_1, p_1) \succ_1 (m_2, y_2, p_2) \equiv y_1 > y_2 \vee (y_1 = y_2 \wedge p_1 < p_2)$$

  for relation $Car(Make, Year, Price)$.
- defined using preference constructors (Preference SQL)
- defined using real-valued scoring functions:

Preference specification

Explicit preference relations

Finite sets of pairs: bmw $\succ$ mazda, mazda $\succ$ kia,...

Implicit preference relations

- can be infinite but finitely representable
- defined using logic formulas in some constraint theory:

$$(m_1, y_1, p_1) \succ_1 (m_2, y_2, p_2) \equiv y_1 > y_2 \vee (y_1 = y_2 \wedge p_1 < p_2)$$

  for relation $Car(Make, Year, Price)$.
- defined using preference constructors (Preference SQL)
- defined using real-valued scoring functions: $[F(m, y, p) = \alpha \cdot y + \beta \cdot p$

Preference specification

Explicit preference relations

Finite sets of pairs: bmw $\succ$ mazda, mazda $\succ$ kia,...

Implicit preference relations

- can be infinite but finitely representable
- defined using logic formulas in some constraint theory:

$$(m_1, y_1, p_1) \succ_1 (m_2, y_2, p_2) \equiv y_1 > y_2 \vee (y_1 = y_2 \wedge p_1 < p_2)$$

  for relation $Car(Make, Year, Price)$.
- defined using preference constructors (Preference SQL)
- defined using real-valued scoring functions: $[F(m, y, p) = \alpha \cdot y + \beta \cdot p$
  $(m_1, y_1, p_1) \succ_2 (m_2, y_2, p_2) \equiv F(m_1, y_1, p_1) > F(m_2, y_2, p_2)$

Logic formulas

Logic formulas

The language of logic formulas

- constants
- object (tuple) attributes
- comparison operators: $=, \neq, <, >, \ldots$
- arithmetic operators: $+, \cdot, \ldots$
- Boolean connectives: $\neg, \wedge, \vee$
- quantifiers:
  - $\forall, \exists$
  - usually can be eliminated (quantifier elimination)

Representability

Representability

Definition

A scoring function f represents a preference relation $\succ$ if for all x, y

$$x \succ y \equiv f(x) > f(y).$$

Representability

Definition

A scoring function f represents a preference relation $\succ$ if for all x, y

$$x \succ y \equiv f(x) > f(y).$$

Necessary condition for representability

The preference relation $\succ$ is a weak order.

Representability

Definition

A scoring function f represents a preference relation $\succ$ if for all x, y

$$x \succ y \equiv f(x) > f(y).$$

Necessary condition for representability

The preference relation $\succ$ is a weak order.

Sufficient condition for representability

- $\succ$ is a weak order
- the domain is countable or some continuity conditions are satisfied (studied in decision theory)

Not every WO can be represented using a scoring function

Not every WO can be represented using a scoring function

Lexicographic order in $R \times R$

$$(x_1, y_1) \succ^{lo} (x_2, y_2) \equiv x_1 > x_2 \vee (x_1 = x_2 \wedge y_1 > y_2)$$

Not every WO can be represented using a scoring function

Lexicographic order in $R \times R$

$$(x_1, y_1) \succ^{lo} (x_2, y_2) \equiv x_1 > x_2 \vee (x_1 = x_2 \wedge y_1 > y_2)$$

Proof

Not every WO can be represented using a scoring function

Lexicographic order in $R \times R$

$$(x_1, y_1) \succ^{lo} (x_2, y_2) \equiv x_1 > x_2 \vee (x_1 = x_2 \wedge y_1 > y_2)$$

Proof

1. Assume there is a real-valued function f such that $x \succ^{lo} y \equiv f(x) > f(y)$.

Not every WO can be represented using a scoring function

Lexicographic order in $R \times R$

$$(x_1, y_1) \succ^{lo} (x_2, y_2) \equiv x_1 > x_2 \vee (x_1 = x_2 \wedge y_1 > y_2)$$

Proof

1. Assume there is a real-valued function f such that $x \succ^{lo} y \equiv f(x) > f(y)$.
2. For every x_0, $(x_0, 1) \succ^{lo} (x_0, 0)$.

Not every WO can be represented using a scoring function

Lexicographic order in $R \times R$

$$(x_1, y_1) \succ^{lo} (x_2, y_2) \equiv x_1 > x_2 \vee (x_1 = x_2 \wedge y_1 > y_2)$$

Proof

1. Assume there is a real-valued function f such that $x \succ^{lo} y \equiv f(x) > f(y)$.
2. For every x_0, $(x_0, 1) \succ^{lo} (x_0, 0)$.
3. Thus $f(x_0, 1) > f(x_0, 0)$.

Not every WO can be represented using a scoring function

Lexicographic order in $R \times R$

$$(x_1, y_1) \succ^{lo} (x_2, y_2) \equiv x_1 > x_2 \vee (x_1 = x_2 \wedge y_1 > y_2)$$

Proof

1. Assume there is a real-valued function f such that $x \succ^{lo} y \equiv f(x) > f(y)$.
2. For every x_0, $(x_0, 1) \succ^{lo} (x_0, 0)$.
3. Thus $f(x_0, 1) > f(x_0, 0)$.
4. Consider now $x_1 > x_0$.

Not every WO can be represented using a scoring function

Lexicographic order in $R \times R$

$$(x_1, y_1) \succ^{lo} (x_2, y_2) \equiv x_1 > x_2 \vee (x_1 = x_2 \wedge y_1 > y_2)$$

Proof

1. Assume there is a real-valued function f such that $x \succ^{lo} y \equiv f(x) > f(y)$.
2. For every x_0, $(x_0, 1) \succ^{lo} (x_0, 0)$.
3. Thus $f(x_0, 1) > f(x_0, 0)$.
4. Consider now $x_1 > x_0$.
5. Clearly $f(x_1, 1) > f(x_1, 0) > f(x_0, 1) > f(x_0, 0)$.

Not every WO can be represented using a scoring function

Lexicographic order in $R \times R$

$$(x_1, y_1) \succ^{lo} (x_2, y_2) \equiv x_1 > x_2 \vee (x_1 = x_2 \wedge y_1 > y_2)$$

Proof

1. Assume there is a real-valued function f such that $x \succ^{lo} y \equiv f(x) > f(y)$.
2. For every x_0, $(x_0, 1) \succ^{lo} (x_0, 0)$.
3. Thus $f(x_0, 1) > f(x_0, 0)$.
4. Consider now $x_1 > x_0$.
5. Clearly $f(x_1, 1) > f(x_1, 0) > f(x_0, 1) > f(x_0, 0)$.
6. So there are uncountably many nonempty disjoint intervals in R.

Not every WO can be represented using a scoring function

Lexicographic order in $R \times R$

$$(x_1, y_1) \succ^{lo} (x_2, y_2) \equiv x_1 > x_2 \vee (x_1 = x_2 \wedge y_1 > y_2)$$

Proof

1. Assume there is a real-valued function f such that $x \succ^{lo} y \equiv f(x) > f(y)$.
2. For every x_0, $(x_0, 1) \succ^{lo} (x_0, 0)$.
3. Thus $f(x_0, 1) > f(x_0, 0)$.
4. Consider now $x_1 > x_0$.
5. Clearly $f(x_1, 1) > f(x_1, 0) > f(x_0, 1) > f(x_0, 0)$.
6. So there are uncountably many nonempty disjoint intervals in R.
7. Each such interval contains a rational number: contradiction with the countability of the set of rational numbers.

Preference constructors [Kie02, KK02]

Preference constructors [Kie02, KK02]

Good values

Prefer $v \in S_1$ over $v \notin S_1$.

Preference constructors [Kie02, KK02]

Good values

Prefer $v \in S_1$ over $v \notin S_1$.

`POS(Make,{mazda,vw})`

Preference constructors [Kie02, KK02]

Good values

Prefer $v \in S_1$ over $v \notin S_1$.

Bad values

Prefer $v \notin S_1$ over $v \in S_1$.

```
POS(Make,{mazda,vw})
```

Preference constructors [Kie02, KK02]

Good values

Prefer $v \in S_1$ over $v \notin S_1$.

Bad values

Prefer $v \notin S_1$ over $v \in S_1$.

`POS(Make,{mazda,vw})`

`NEG(Make,{yugo})`

Preference constructors [Kie02, KK02]

Good values

Prefer $v \in S_1$ over $v \notin S_1$.

`POS(Make,{mazda,vw})`

Bad values

Prefer $v \notin S_1$ over $v \in S_1$.

`NEG(Make,{yugo})`

Explicit preference

Preference encoded by a finite directed graph.

Preference constructors [Kie02, KK02]

Good values

Prefer $v \in S_1$ over $v \notin S_1$.

`POS(Make,{mazda,vw})`

Bad values

Prefer $v \notin S_1$ over $v \in S_1$.

`NEG(Make,{yugo})`

Explicit preference

Preference encoded by a finite directed graph.

```
EXP(Make,{(bmw,ford),...,
              (mazda,kia)})
```

Preference constructors [Kie02, KK02]

Good values

Prefer $v \in S_1$ over $v \notin S_1$.

`POS(Make,{mazda,vw})`

Bad values

Prefer $v \notin S_1$ over $v \in S_1$.

`NEG(Make,{yugo})`

Explicit preference

Preference encoded by a finite directed graph.

```
EXP(Make,{(bmw,ford),...,
          (mazda,kia)})
```

Value comparison

Prefer larger/smaller values.

Preference constructors [Kie02, KK02]

Good values

Prefer $v \in S_1$ over $v \notin S_1$.

`POS(Make,{mazda,vw})`

Bad values

Prefer $v \notin S_1$ over $v \in S_1$.

`NEG(Make,{yugo})`

Explicit preference

Preference encoded by a finite directed graph.

```
EXP(Make,{(bmw,ford),...,
          (mazda,kia)})
```

Value comparison

Prefer larger/smaller values.

```
HIGHEST(Year)
LOWEST(Price)
```

Preference constructors [Kie02, KK02]

Good values

Prefer $v \in S_1$ over $v \notin S_1$.

`POS(Make,{mazda,vw})`

Bad values

Prefer $v \notin S_1$ over $v \in S_1$.

`NEG(Make,{yugo})`

Explicit preference

Preference encoded by a finite directed graph.

```
EXP(Make,{(bmw,ford),...,
          (mazda,kia)})
```

Value comparison

Prefer larger/smaller values.

```
HIGHEST(Year)
LOWEST(Price)
```

Distance

Prefer values closer to v_0.

Preference constructors [Kie02, KK02]

Good values

Prefer $v \in S_1$ over $v \notin S_1$.

`POS(Make,{mazda,vw})`

Bad values

Prefer $v \notin S_1$ over $v \in S_1$.

`NEG(Make,{yugo})`

Explicit preference

Preference encoded by a finite directed graph.

```
EXP(Make,{(bmw,ford),...,
          (mazda,kia)})
```

Value comparison

Prefer larger/smaller values.

```
HIGHEST(Year)
LOWEST(Price)
```

Distance

Prefer values closer to v_0.

`AROUND(Price,12K)`

Combining preferences

Combining preferences

Preference composition

- combining preferences about objects of the same kind
- dimensionality is not increased
- representing preference aggregation, revision, ...

Combining preferences

Preference composition

- combining preferences about objects of the same kind
- dimensionality is not increased
- representing preference aggregation, revision, ...

Preference accumulation

- defining preferences over objects in terms of preferences over simpler objects
- dimensionality is increased (preferences over Cartesian product).

Combining preferences: composition

Combining preferences: composition

Boolean composition

$$x \succ^{\cup} y \equiv x \succ_1 y \ \vee \ x \succ_2 y$$

and similarly for $\cap$.

Combining preferences: composition

Boolean composition

$$x \succ^{\cup} y \equiv x \succ_1 y \;\vee\; x \succ_2 y$$

and similarly for $\cap$.

Prioritized composition

$$x \succ^{lex} y \equiv x \succ_1 y \vee (y \not\succ_1 x \wedge x \succ_2 y).$$

Combining preferences: composition

Boolean composition

$$x \succ^{\cup} y \equiv x \succ_1 y \ \vee \ x \succ_2 y$$

and similarly for $\cap$.

Prioritized composition

$$x \succ^{lex} y \equiv x \succ_1 y \vee (y \not\succ_1 x \wedge x \succ_2 y).$$

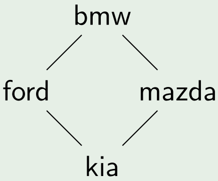
Pareto composition

$$x \succ^{Par} y \equiv (x \succ_1 y \wedge y \not\succ_2 x) \vee (x \succ_2 y \wedge y \not\succ_1 x).$$

Preference composition

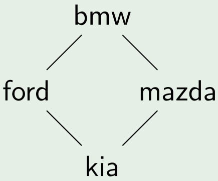
Preference composition

Preference relation $\succ_1$

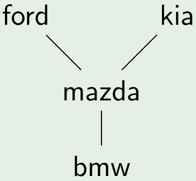


Preference composition

Preference relation $\succ_1$

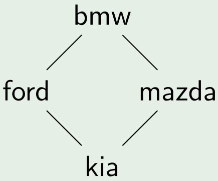


Preference relation $\succ_2$

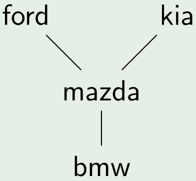


Preference composition

Preference relation $\succ_1$



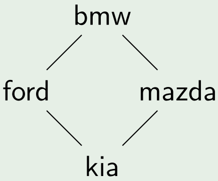
Preference relation $\succ_2$



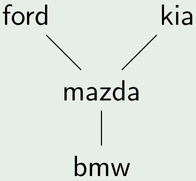
Prioritized composition

Preference composition

Preference relation $\succ_1$

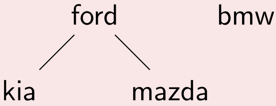


Preference relation $\succ_2$



Prioritized composition

Pareto composition



Combining preferences: accumulation [Kie02]

Combining preferences: accumulation [Kie02]

Prioritized accumulation: $\succ^{pr} = (\succ_1 \,\&\, \succ_2)$

$$(x_1, x_2) \succ^{pr} (y_1, y_2) \equiv x_1 \succ_1 y_1 \vee (x_1 = y_1 \wedge x_2 \succ_2 y_2).$$

Combining preferences: accumulation [Kie02]

Prioritized accumulation: $\succ^{pr} = (\succ_1 \,\&\, \succ_2)$

$$(x_1, x_2) \succ^{pr} (y_1, y_2) \equiv x_1 \succ_1 y_1 \vee (x_1 = y_1 \wedge x_2 \succ_2 y_2).$$

Pareto accumulation: $\succ^{pa} = (\succ_1 \otimes \succ_2)$

$$(x_1, x_2) \succ^{pa} (y_1, y_2) \equiv (x_1 \succ_1 y_1 \wedge x_2 \succeq_2 y_2) \vee (x_1 \succeq_1 y_1 \wedge x_2 \succ_2 y_2).$$

Combining preferences: accumulation [Kie02]

Prioritized accumulation: $\succ^{pr} = (\succ_1 \& \succ_2)$

$$(x_1, x_2) \succ^{pr} (y_1, y_2) \equiv x_1 \succ_1 y_1 \vee (x_1 = y_1 \wedge x_2 \succ_2 y_2).$$

Pareto accumulation: $\succ^{pa} = (\succ_1 \otimes \succ_2)$

$$(x_1, x_2) \succ^{pa} (y_1, y_2) \equiv (x_1 \succ_1 y_1 \wedge x_2 \succeq_2 y_2) \vee (x_1 \succeq_1 y_1 \wedge x_2 \succ_2 y_2).$$

Properties

- closure
- associativity
- commutativity of Pareto accumulation

Skylines

Skylines

Skyline

Given single-attribute total preference relations $\succ_{A_1}, \ldots, \succ_{A_n}$ for a relational schema $R(A_1, \ldots, A_n)$, the skyline preference relation $\succ^{sky}$ is defined as

$$\succ^{sky} = \succ_{A_1} \otimes \succ_{A_2} \otimes \cdots \otimes \succ_{A_n} .$$

Unfolding the definition

$$(x_1, \ldots, x_n) \succ^{sky} (y_1, \ldots, y_n) \equiv \bigwedge_i x_i \succeq_{A_i} y_i \wedge \bigvee_i x_i \succ_{A_i} y_i .$$

Skylines

Skyline

Given single-attribute total preference relations $\succ_{A_1}, \ldots, \succ_{A_n}$ for a relational schema $R(A_1, \ldots, A_n)$, the skyline preference relation $\succ^{sky}$ is defined as

$$\succ^{sky} = \succ_{A_1} \otimes \succ_{A_2} \otimes \cdots \otimes \succ_{A_n} .$$

Unfolding the definition

$$(x_1, \ldots, x_n) \succ^{sky} (y_1, \ldots, y_n) \equiv \bigwedge_i x_i \succeq_{A_i} y_i \wedge \bigvee_i x_i \succ_{A_i} y_i .$$

Two-dimensional Euclidean space

$$(x_1, x_2) \succ^{sky} (y_1, y_2) \equiv x_1 \geq y_1 \wedge x_2 > y_2 \vee x_1 > y_1 \wedge x_2 \geq y_2$$

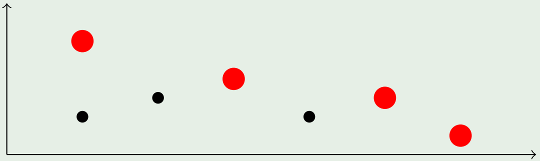
Skyline in Euclidean space

Skyline in Euclidean space

Maximal skyline vectors

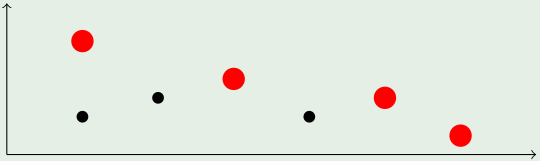
Skyline in Euclidean space

Maximal skyline vectors



Skyline in Euclidean space

Maximal skyline vectors

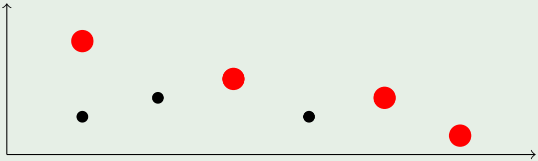


Maxima

A skyline consists of all maxima of monotonic scoring functions.

Skyline in Euclidean space

Maximal skyline vectors



Maxima

A skyline consists of all maxima of monotonic scoring functions.

Skyline is not a WO

$$(2,0) \not\succ_{sky} (0,2), (0,2) \not\succ_{sky} (1,0), (2,0) \succ_{sky} (1,0)$$

Skyline variants

Skyline variants

Groupwise skyline

- compare only tuples in the same group

Skyline variants

Groupwise skyline

- compare only tuples in the same group

Order properties

Attribute orders are general SPOs.

Skyline variants

Groupwise skyline

- compare only tuples in the same group

Order properties

Attribute orders are general SPOs.

Non-Euclidean spaces

Metric spaces:

- distance vectors in road networks

Skyline variants

Groupwise skyline

- compare only tuples in the same group

Order properties

Attribute orders are general SPOs.

Non-Euclidean spaces

Metric spaces:

- distance vectors in road networks

Dynamic attributes

Attribute values can change dynamically:

- distance from query point in road networks

Combining scoring functions

Combining scoring functions

Scoring functions can be combined using numerical operators.

Combining scoring functions

Scoring functions can be combined using numerical operators.

Common scenario

- scoring functions $f_1, \ldots, f_n$
- aggregate scoring function: $F(t) = E(f_1(t), \ldots, f_n(t))$
- linear scoring function: $\Sigma_{i=1}^{n} \alpha_i f_i$

Combining scoring functions

Scoring functions can be combined using numerical operators.

Common scenario

- scoring functions $f_1, \ldots, f_n$
- aggregate scoring function: $F(t) = E(f_1(t), \ldots, f_n(t))$
- linear scoring function: $\Sigma_{i=1}^{n} \alpha_i f_i$

Numerical vs. logical combination

- logical combination cannot be defined numerically
- numerical combination cannot be defined logically (unless arithmetic operators are available)

Part II

Preference Queries

Outline of Part II

2 Preference queries

- Retrieving non-dominated elements
- Rewriting queries with winnow
- Retrieving Top-K elements
- Optimizing Top-K queries

Winnow[Cho03]

Winnow[Cho03]

Winnow

- new relational algebra operator ω (other names: Best, BMO [Kie02])
- retrieves the non-dominated (best) elements in a database relation
- can be expressed in terms of other operators

Winnow[Cho03]

Winnow

- new relational algebra operator ω (other names: Best, BMO [Kie02])
- retrieves the non-dominated (best) elements in a database relation
- can be expressed in terms of other operators

Definition

Given a preference relation $\succ$ and a database relation r:

$$\omega_{\succ}(r) = \{t \in r \mid \neg\exists t' \in r.\ t' \succ t\}.$$

Winnow[Cho03]

Winnow

- new relational algebra operator ω (other names: Best, BMO [Kie02])
- retrieves the non-dominated (best) elements in a database relation
- can be expressed in terms of other operators

Definition

Given a preference relation $\succ$ and a database relation r:

$$\omega_{\succ}(r) = \{t \in r \mid \neg\exists t' \in r.\ t' \succ t\}.$$

Notation: If a preference relation $\succ_C$ is defined using a formula C, then we write $\omega_C(r)$, instead of $\omega_{\succ_C}(r)$.

Winnow[Cho03]

Winnow

- new relational algebra operator ω (other names: Best, BMO [Kie02])
- retrieves the non-dominated (best) elements in a database relation
- can be expressed in terms of other operators

Definition

Given a preference relation $\succ$ and a database relation r:

$$\omega_{\succ}(r) = \{t \in r \mid \neg\exists t' \in r.\ t' \succ t\}.$$

Notation: If a preference relation $\succ_C$ is defined using a formula C, then we write $\omega_C(r)$, instead of $\omega_{\succ_C}(r)$.

Skyline query

$\omega_{\succ_{sky}}(r)$ computes the set of maximal vectors in r (the skyline set).

Example of winnow

Example of winnow

Relation *Car*(*Make*, *Year*, *Price*)

Preference relation:

$$(m, y, p) \succ_1 (m', y', p') \equiv y > y' \vee (y = y' \wedge p < p').$$

Example of winnow

Relation $Car(Make, Year, Price)$

Preference relation:

$$(m, y, p) \succ_1 (m', y', p') \equiv y > y' \vee (y = y' \wedge p < p').$$

Make	Year	Price
mazda	2009	20K
ford	2009	15K
ford	2007	12K

Example of winnow

Relation $Car(Make, Year, Price)$

Preference relation:

$$(m, y, p) \succ_1 (m', y', p') \equiv y > y' \vee (y = y' \wedge p < p').$$

Make	Year	Price
mazda	2009	20K
ford	2009	15K
ford	2007	12K

Computing winnow using BNL [BKS01]

Require: SPO $\succ$, database relation r

```
initialize window W and temporary file F to empty
repeat
   for every tuple t in the input do
      if t is dominated by a tuple in W then
         ignore t
      else if t dominates some tuples in W then
         eliminate them and insert t into W
      else if there is room in W then
         insert t into W
      else
         add t to F
      end if
   end for
   output tuples from W that were added when F was empty
   make F the input, clear F
until empty input
```

BNL in action

Preference relation: a $\succ$ c, a $\succ$ d, b $\succ$ e.

BNL in action

Preference relation: $a \succ c$, $a \succ d$, $b \succ e$.

Temporary file

Window

Input

c,e,d,a,b

BNL in action

Preference relation: $a \succ c$, $a \succ d$, $b \succ e$.

Temporary file

Window

c

Input

e,d,a,b

BNL in action

Preference relation: a $\succ$ c, a $\succ$ d, b $\succ$ e.

Temporary file

Window

c
e

Input

d,a,b

BNL in action

Preference relation: $a \succ c$, $a \succ d$, $b \succ e$.

Temporary file

d

Window

c
e

Input

a,b

BNL in action

Preference relation: $a \succ c$, $a \succ d$, $b \succ e$.

Temporary file

d

Window

a
e

Input

b

BNL in action

Preference relation: $a \succ c$, $a \succ d$, $b \succ e$.

Temporary file

d

Window

a
b

Input

BNL in action

Preference relation: $a \succ c$, $a \succ d$, $b \succ e$.

Temporary file

Window

a

b

Input

d

BNL in action

Preference relation: a $\succ$ c, a $\succ$ d, b $\succ$ e.

Temporary file

Window

a

b

Input

Computing winnow with presorting

Computing winnow with presorting

SFS: adding presorting step to BNL [CGGL03]

- topologically sort the input:
  - if x dominates y, then x precedes y in the sorted input
  - window contains only winnow points and can be output after every pass
- for skylines: sort the input using a monotonic scoring function, for example $\prod_{i=1}^{k} x_i$.

Computing winnow with presorting

SFS: adding presorting step to BNL [CGGL03]

- topologically sort the input:
  - if x dominates y, then x precedes y in the sorted input
  - window contains only winnow points and can be output after every pass
- for skylines: sort the input using a monotonic scoring function, for example $\prod_{i=1}^{k} x_i$.

LESS: integrating different techniques [GSG07]

- adding an elimination filter to the first external sort pass
- combining the last external sort pass with the first SFS pass
- average running time: $O(kn)$

SFS in action

Preference relation: a $\succ$ c, a $\succ$ d, b $\succ$ e.

SFS in action

Preference relation: a $\succ$ c, a $\succ$ d, b $\succ$ e.

Temporary file

Window

Input

a,b,c,d,e

SFS in action

Preference relation: $a \succ c$, $a \succ d$, $b \succ e$.

Temporary file

Window

a

Input

b,c,d,e

SFS in action

Preference relation: $a \succ c$, $a \succ d$, $b \succ e$.

Temporary file

Window

a
b

Input

c,d,e

SFS in action

Preference relation: $a \succ c$, $a \succ d$, $b \succ e$.

Temporary file

Window

a
b

Input

d,e

SFS in action

Preference relation: $a \succ c$, $a \succ d$, $b \succ e$.

Temporary file

Window

a
b

Input

e

SFS in action

Preference relation: a $\succ$ c, a $\succ$ d, b $\succ$ e.

Temporary file

Window

a
b

Input

Generalizations of winnow

Generalizations of winnow

Iterating winnow

$$\omega^0_\succ(r) = \omega_\succ(r)$$
$$\omega^{n+1}_\succ(r) = \omega_\succ(r - \bigcup_{1\leq i\leq n} \omega^i_\succ(r))$$

Generalizations of winnow

Iterating winnow

$$\omega^0_\succ(r) = \omega_\succ(r)$$
$$\omega^{n+1}_\succ(r) = \omega_\succ(r - \bigcup_{1\leq i\leq n} \omega^i_\succ(r))$$

Ranking

Rank tuples by their minimum distance from a winnow tuple:

$$\eta_\succ(r) = \{(t,i) \mid t \in \omega^i_C(r)\}.$$

Generalizations of winnow

Iterating winnow

$$\omega^0_\succ(r) = \omega_\succ(r)$$
$$\omega^{n+1}_\succ(r) = \omega_\succ(r - \bigcup_{1\leq i\leq n} \omega^i_\succ(r))$$

Ranking

Rank tuples by their minimum distance from a winnow tuple:

$$\eta_\succ(r) = \{(t, i) \mid t \in \omega^i_C(r)\}.$$

k-band

Return the tuples dominated by at most k tuples:

$$\omega_\succ(r) = \{t \in r \mid \#\{t' \in r \mid t' \succ t\} \leq k\}.$$

Preference SQL

Preference SQL

The language

- basic preference constructors
- Pareto/prioritized accumulation
- new SQL clause `PREFERRING`
- groupwise preferences
- implementation: translation to SQL

Preference SQL

The language

- basic preference constructors
- Pareto/prioritized accumulation
- new SQL clause `PREFERRING`
- groupwise preferences
- implementation: translation to SQL

Winnow in Preference SQL

```
SELECT * FROM Car
PREFERRING HIGHEST(Year)
    CASCADE LOWEST(Price)
```

Algebraic laws [Cho03]

Algebraic laws [Cho03]

Commutativity of winnow with selection

If the formula

$$\forall t_1, t_2.[\alpha(t_2) \wedge \gamma(t_1, t_2)] \Rightarrow \alpha(t_1)$$

is valid, then for every r

$$\sigma_\alpha(\omega_\gamma(r)) = \omega_\gamma(\sigma_\alpha(r)).$$

Algebraic laws [Cho03]

Commutativity of winnow with selection

If the formula

$$\forall t_1, t_2.[\alpha(t_2) \wedge \gamma(t_1, t_2)] \Rightarrow \alpha(t_1)$$

is valid, then for every r

$$\sigma_\alpha(\omega_\gamma(r)) = \omega_\gamma(\sigma_\alpha(r)).$$

Under the preference relation

$$(m, y, p) \succ_{C_1} (m', y', p') \equiv y > y' \wedge p \leq p' \vee y \geq y' \wedge p < p'$$

the selection $\sigma_{Price<20K}$ commutes with ω_{C_1} but $\sigma_{Price>20K}$ does not.

Other algebraic laws

Other algebraic laws

Distributivity of winnow over Cartesian product

For every r_1 and r_2

$$\omega_C(r_1 \times r_2) = \omega_C(r_1) \times r_2$$

if C refers only to the attributes of r_1.

Commutativity of winnow

If $\forall t_1, t_2.[C_1(t_1, t_2) \Rightarrow C_2(t_1, t_2)]$ is valid and $\succ_{C_1}$ and $\succ_{C_2}$ are SPOs, then for all finite instances r:

$$\omega_{C_1}(\omega_{C_2}(r)) = \omega_{C_2}(\omega_{C_1}(r)) = \omega_{C_2}(r).$$

Semantic query optimization [Cho07b]

Semantic query optimization [Cho07b]

Using information about integrity constraints to:

- eliminate redundant occurrences of winnow.
- make more efficient computation of winnow possible.

Eliminating redundancy

Given a set of integrity constraints F, ω_C is redundant w.r.t. F iff F implies the formula

$$\forall t_1, t_2.\ R(t_1) \wedge R(t_2) \Rightarrow t_1 \sim_C t_2.$$

Integrity constraints

Integrity constraints

Constraint-generating dependencies (CGD) [BCW99, ZO97]

$$\forall t_1 \ldots . \forall t_n . \; [R(t_1) \wedge \cdots \wedge R(t_n) \wedge \gamma(t_1, \ldots t_n)] \Rightarrow \gamma'(t_1, \ldots t_n).$$

Integrity constraints

Constraint-generating dependencies (CGD) [BCW99, ZO97]

$$\forall t_1 . \ldots . \forall t_n . \; [R(t_1) \wedge \cdots \wedge R(t_n) \wedge \gamma(t_1, \ldots t_n)] \Rightarrow \gamma'(t_1, \ldots t_n).$$

CGD entailment

Decidable by reduction to the validity of $\forall$-formulas in the constraint theory (assuming the theory is decidable).

Top-K queries

Top-K queries

Scoring functions

- each tuple t in a relation has numeric scores $f_1(t), \ldots, f_m(t)$ assigned by numeric component scoring functions $f_1, \ldots, f_m$
- the aggregate score of t is $F(t) = E(f_1(t), \ldots, f_m(t))$ where E is a numeric-valued expression
- F is monotone if $E(x_1, \ldots, x_m) \leq E(y_1, \ldots, y_m)$ whenever $x_i \leq y_i$ for all i

Top-K queries

Scoring functions

- each tuple t in a relation has numeric scores $f_1(t), \ldots, f_m(t)$ assigned by numeric component scoring functions $f_1, \ldots, f_m$
- the aggregate score of t is $F(t) = E(f_1(t), \ldots, f_m(t))$ where E is a numeric-valued expression
- F is monotone if $E(x_1, \ldots, x_m) \leq E(y_1, \ldots, y_m)$ whenever $x_i \leq y_i$ for all i

Top-K queries

- return K elements having top F-values in a database relation R
- query expressed in an extension of SQL:

```
SELECT *
FROM R
ORDER BY F DESC
LIMIT K
```

Top-K sets

Top-K sets

Definition

Given a scoring function F and a database relation r, s is a Top-K set if:

- $s \subseteq r$
- $|s| = \min(K, |r|)$
- $\forall t \in s.\ \forall t' \in r - s.\ F(t) \geq F(t')$

There may be more than one Top-K set: one is selected non-deterministically.

Example of Top-2

Example of Top-2

Relation $Car(Make, Year, Price)$

- component scoring functions:

$$f_1(m, y, p) = (y - 2005)$$

$$f_2(m, y, p) = (20000 - p)$$

- aggregate scoring function:
  $F(m, y, p) = 1000 \cdot f_1(m, y, p) + f_2(m, y, p)$

Example of Top-2

Relation *Car*(*Make*, *Year*, *Price*)

- component scoring functions:

$$f_1(m, y, p) = (y - 2005)$$

$$f_2(m, y, p) = (20000 - p)$$

- aggregate scoring function:
  $F(m, y, p) = 1000 \cdot f_1(m, y, p) + f_2(m, y, p)$

Make	Year	Price	Aggregate score
mazda	2009	20000	4000
ford	2009	15000	9000
ford	2007	12000	10000

Example of Top-2

Relation $Car(Make, Year, Price)$

- component scoring functions:

$$f_1(m, y, p) = (y - 2005)$$

$$f_2(m, y, p) = (20000 - p)$$

- aggregate scoring function:
$F(m, y, p) = 1000 \cdot f_1(m, y, p) + f_2(m, y, p)$

Make	Year	Price	Aggregate score
mazda	2009	20000	4000
ford	2009	15000	9000
ford	2007	12000	10000

Computing Top-K

Computing Top-K

Naive approaches

- sort, output the first K-tuples
- scan the input maintaining a priority queue of size K
- ...

Computing Top-K

Naive approaches

- sort, output the first K-tuples
- scan the input maintaining a priority queue of size K
- ...

Better approaches

Computing Top-K

Naive approaches

- sort, output the first K-tuples
- scan the input maintaining a priority queue of size K
- ...

Better approaches

- the entire input does not need to be scanned...

Computing Top-K

Naive approaches

- sort, output the first K-tuples
- scan the input maintaining a priority queue of size K
- ...

Better approaches

- the entire input does not need to be scanned...
- ... provided additional data structures are available

Computing Top-K

Naive approaches

- sort, output the first K-tuples
- scan the input maintaining a priority queue of size K
- ...

Better approaches

- the entire input does not need to be scanned...
- ... provided additional data structures are available
- variants of the threshold algorithm

Threshold algorithm (TA)[FLN03]

Threshold algorithm (TA)[FLN03]

Inputs

- a monotone scoring function $F(t) = E(f_1(t), \ldots, f_m(t))$
- lists S_i, $i = 1, \ldots, m$, each sorted on f_i (descending) and representing a different ranking of the same set of objects

1. For each list S_i in parallel retrieve the current object w in sorted order:
   - (random access) for every $j \neq i$, retrieve $v_j = f_j(w)$ from the list S_j
   - if $d = E(v_1, \ldots, v_m)$ is among the highest K scores seen so far, remember t and d (ties broken arbitrarily)
2. Thresholding:
   - for each i: w_i the last object seen under sorted access in S_i
   - if there are already K top-K objects with score at least equal to the threshold $T = E(f_1(w_1), \ldots, f_m(w_m))$, return collected objects sorted by F and terminate
   - otherwise, go to step 1.

TA in action

Aggregate score

$$F(t) = P_1(t) + P_2(t)$$

OID	P_1
5	50
1	35
3	30
2	20
4	10

OID	P_2
3	50
2	40
1	30
4	20
5	10

Priority queue

TA in action

Aggregate score

$$F(t) = P_1(t) + P_2(t)$$

OID	P_1
5	50
1	35
3	30
2	20
4	10

OID	P_2
3	50
2	40
1	30
4	20
5	10

Priority queue

T=100

TA in action

Aggregate score

$$F(t) = P_1(t) + P_2(t)$$

OID	P_1
5	50
1	35
3	30
2	20
4	10

OID	P_2
3	50
2	40
1	30
4	20
5	10

Priority queue

5:60

T=100

TA in action

Aggregate score

$$F(t) = P_1(t) + P_2(t)$$

Priority queue

OID	P_1
5	50
1	35
3	30
2	20
4	10

OID	P_2
3	50
2	40
1	30
4	20
5	10

3:80
5:60

T=100

TA in action

Aggregate score

$$F(t) = P_1(t) + P_2(t)$$

Priority queue

OID	P_1
5	50
1	35
3	30
2	20
4	10

OID	P_2
3	50
2	40
1	30
4	20
5	10

3:80
5:60

T=75

TA in action

Aggregate score

$$F(t) = P_1(t) + P_2(t)$$

OID	P_1
5	50
1	35
3	30
2	20
4	10

OID	P_2
3	50
2	40
1	30
4	20
5	10

Priority queue

3:80
1:65
5:60

T=75

TA in action

Aggregate score

$$F(t) = P_1(t) + P_2(t)$$

OID	P_1
5	50
1	35
3	30
2	20
4	10

OID	P_2
3	50
2	40
1	30
4	20
5	10

Priority queue

3:80
1:65
5:60
2:60

T=75

TA in databases

TA in databases

- objects: tuples of a single relation r
- single-attribute component scoring functions
- sorted list access implemented through indexes
- random access to all lists implemented by primary index access to r that retrieves entire tuples

Optimizing Top-K queries [LCIS05]

Optimizing Top-K queries [LCIS05]

Goals

- integrating Top-K with relational query evaluation and optimization
- replacing blocking by pipelining

Optimizing Top-K queries [LCIS05]

Goals

- integrating Top-K with relational query evaluation and optimization
- replacing blocking by pipelining

Example

SELECT *
FROM Hotel h, Restaurant r, Museum m
WHERE c_1 AND c_2 AND c_3
ORDER BY $f_1 + f_2 + f_3$
LIMIT K

Optimizing Top-K queries [LCIS05]

Goals

- integrating Top-K with relational query evaluation and optimization
- replacing blocking by pipelining

Example

SELECT *
FROM Hotel h, Restaurant r, Museum m
WHERE c_1 AND c_2 AND c_3
ORDER BY $f_1 + f_2 + f_3$
LIMIT K

Is there a better evaluation plan than materialize-then-sort?

Partial ranking of tuples

Partial ranking of tuples

Model

- set of component scoring functions $P = \{f_1, \ldots, f_m\}$ such that $f_i(t) \leq 1$ for all t
- aggregate scoring function $F(t) = E(f_1(t), \ldots, f_m(t))$
- how to rank intermediate tuples?

Partial ranking of tuples

Model

- set of component scoring functions $P = \{f_1, \ldots, f_m\}$ such that $f_i(t) \leq 1$ for all t
- aggregate scoring function $F(t) = E(f_1(t), \ldots, f_m(t))$
- how to rank intermediate tuples?

Ranking principle

Given $P_0 \subseteq P$,

$$\bar{F}_{P_0}(t) = E(g_1(t), \ldots, g_m(t))$$

where

$$g_i(t) = \begin{cases} f_i(t) & \text{if } f_i \in P_0 \\ 1 & \text{otherwise} \end{cases}$$

Relations with rank

Relations with rank

Rank-relation R_{P_0}

- relation R
- monotone aggregate scoring function F (the same for all relations)
- set of component scoring functions $P_0 \subseteq P$
- order:

$$t_1 >_{R_{P_0}} t_2 \equiv \bar{F}_{P_0}(t_1) > \bar{F}_{P_0}(t_2)$$

Ranking intermediate results

Ranking intermediate results

Operators

- rank operator μ_f: ranks tuples according to an additional component scoring function f
- standard relational algebra operators suitably extended to work on rank-relations

Ranking intermediate results

Operators

- rank operator μ_f: ranks tuples according to an additional component scoring function f
- standard relational algebra operators suitably extended to work on rank-relations

Operator	Order
$\mu_f(R_{P_0})$	$t_1 >_{\mu_f(R_{P_0})} t_2 \equiv \bar{F}_{P_0 \cup \{f\}}(t_1) > \bar{F}_{P_0 \cup \{f\}}(t_2)$
$R_{P_1} \cap S_{P_2}$	$t_1 >_{R_{P_1} \cap S_{P_2}} t_2 \equiv \bar{F}_{P_1 \cup P_2}(t_1) > \bar{F}_{P_1 \cup P_2}(t_2)$

Example

Example

Query

```
SELECT *
FROM S
ORDER BY f1 + f2 + f3
LIMIT 1
```

Example

Query

```
SELECT *
FROM S
ORDER BY f1 + f2 + f3
LIMIT 1
```

Unranked relation S

A	f_1	f_2	f_3
1	0.7	0.8	0.9
2	0.9	0.85	0.8
3	0.5	0.45	0.75

Example

Query

```
SELECT *
FROM S
ORDER BY f1 + f2 + f3
LIMIT 1
```

Unranked relation S

A	f_1	f_2	f_3
1	0.7	0.8	0.9
2	0.9	0.85	0.8
3	0.5	0.45	0.75

Rank-relation $S_{\{f_1\}}$

A	$\bar{F}_{\{f_1\}}$
2	2.9
1	2.7
3	2.5

Pipelined execution

Pipelined execution

A	f_1	f_2	f_3	$\bar{F}_{\{f_1\}}$
2	0.9	0.85	0.8	2.9
1	0.7	0.8	0.9	2.7
3	0.5	0.45	0.75	2.5

$\uparrow$ $IndexScan_{f_1}$

Pipelined execution

A	f_1	f_2	f_3	$\bar{F}_{\{f_1\}}$
2	0.9	0.85	0.8	2.9
1	0.7	0.8	0.9	2.7
3	0.5	0.45	0.75	2.5

$$\xrightarrow{\mu_{f_2}}$$

A	$\bar{F}_{\{f_1,f_2\}}$
2	2.75
1	2.5
3	1.95

$\uparrow IndexScan_{f_1}$

Pipelined execution

A	f_1	f_2	f_3	$\bar{F}_{\{f_1\}}$
2	0.9	0.85	0.8	2.9
1	0.7	0.8	0.9	2.7
3	0.5	0.45	0.75	2.5

$\uparrow IndexScan_{f_1}$

$\xrightarrow{\mu_{f_2}}$

A	$\bar{F}_{\{f_1,f_2\}}$
2	2.75
1	2.5
3	1.95

$\downarrow \mu_{f_3}$

A	$\bar{F}_{\{f_1,f_2,f_3\}}$
2	2.55
1	2.4

Algebraic laws for rank-relation operators

Algebraic laws for rank-relation operators

Splitting for μ

$$R_{\{f_1,f_2,\dots,f_m\}} \equiv \mu_{f_1}(\mu_{f_2}(\dots(\mu_{f_m}(R))\dots))$$

Algebraic laws for rank-relation operators

Splitting for μ

$$R_{\{f_1,f_2,\ldots,f_m\}} \equiv \mu_{f_1}(\mu_{f_2}(\ldots(\mu_{f_m}(R))\ldots))$$

Commutativity of μ

$$\mu_{f_1}(\mu_{f_2}(R_{P_0})) \equiv \mu_{f_2}(\mu_{f_1}(R_{P_0}))$$

Algebraic laws for rank-relation operators

Splitting for μ

$$R_{\{f_1, f_2, \ldots, f_m\}} \equiv \mu_{f_1}(\mu_{f_2}(\ldots(\mu_{f_m}(R))\ldots))$$

Commutativity of μ

$$\mu_{f_1}(\mu_{f_2}(R_{P_0})) \equiv \mu_{f_2}(\mu_{f_1}(R_{P_0}))$$

Commutativity of μ with selection

$$\sigma_C(\mu_f(R_{P_0})) \equiv \mu_f(\sigma_C(R_{P_0}))$$

Algebraic laws for rank-relation operators

Splitting for μ

$$R_{\{f_1,f_2,\ldots,f_m\}} \equiv \mu_{f_1}(\mu_{f_2}(\ldots(\mu_{f_m}(R))\ldots))$$

Commutativity of μ

$$\mu_{f_1}(\mu_{f_2}(R_{P_0})) \equiv \mu_{f_2}(\mu_{f_1}(R_{P_0}))$$

Commutativity of μ with selection

$$\sigma_C(\mu_f(R_{P_0})) \equiv \mu_f(\sigma_C(R_{P_0}))$$

Distributivity of μ over Cartesian product

$\mu_f(R_{P_1} \times S_{P_2}) \equiv \mu_f(R_{P_1}) \times S_{P_2}$ if f refers only to the attributes of R.

Part III

Preference management

Outline of Part III

3. Preference management
    - Preference modification

Preference modification

Preference modification

Goal

Given a preference relation $\succ$ and additional preference or indifference information I, construct a new preference relation $\succ'$ whose contents depend on $\succ$ and I.

Preference modification

Goal

Given a preference relation $\succ$ and additional preference or indifference information I, construct a new preference relation $\succ'$ whose contents depend on $\succ$ and I.

General postulates

- fulfillment: the new information I should be completely incorporated into $\succ'$
- minimal change: $\succ$ should be changed as little as possible
- closure:
  - order-theoretic properties of $\succ$ should be preserved in $\succ'$ (SPO, WO)
  - finiteness or finite representability of $\succ$ should also be preserved in $\succ'$

Preference revision [Cho07a]

Preference revision [Cho07a]

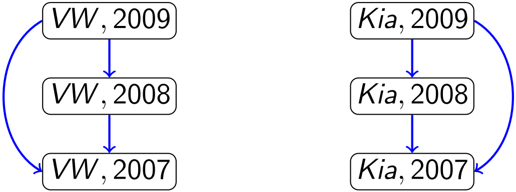
Setting

- new information: revising preference relation $\succ_0$
- composition operator θ: union, prioritized or Pareto composition
- composition eliminates (some) preference conflicts
- additional assumptions: interval orders
- $\succ' = TC(\succ_0 \, \theta \succ)$ to guarantee SPO

Preference revision [Cho07a]

Setting

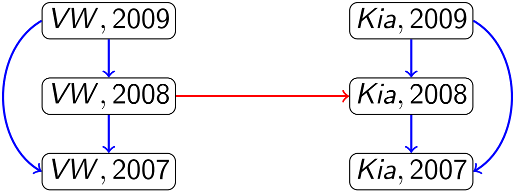
- new information: revising preference relation $\succ_0$
- composition operator θ: union, prioritized or Pareto composition
- composition eliminates (some) preference conflicts
- additional assumptions: interval orders
- $\succ' = TC(\succ_0 \, \theta \succ)$ to guarantee SPO



Preference revision [Cho07a]

Setting

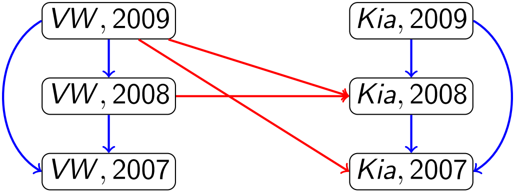
- new information: revising preference relation $\succ_0$
- composition operator θ: union, prioritized or Pareto composition
- composition eliminates (some) preference conflicts
- additional assumptions: interval orders
- $\succ' = TC(\succ_0 \, \theta \succ)$ to guarantee SPO



Preference revision [Cho07a]

Setting

- new information: revising preference relation $\succ_0$
- composition operator θ: union, prioritized or Pareto composition
- composition eliminates (some) preference conflicts
- additional assumptions: interval orders
- $\succ' = TC(\succ_0 \, \theta \succ)$ to guarantee SPO



Preference contraction [MC08]

Preference contraction [MC08]

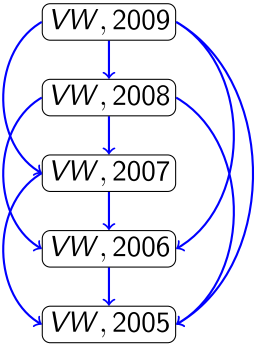
Setting

- new information: contractor relation *CON*
- $\succ'$: maximal subset of $\succ$ disjoint with *CON*

Preference contraction [MC08]

Setting

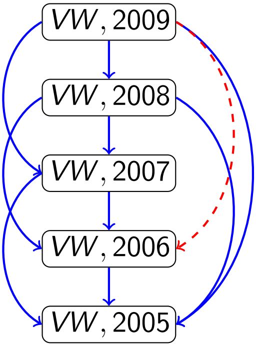
- new information: contractor relation CON
- $\succ'$: maximal subset of $\succ$ disjoint with CON



Preference contraction [MC08]

Setting

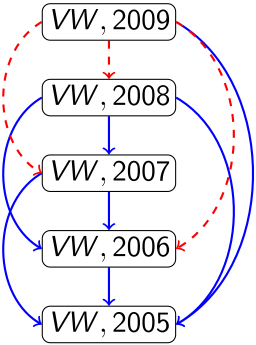
- new information: contractor relation CON
- $\succ'$: maximal subset of $\succ$ disjoint with CON



Preference contraction [MC08]

Setting

- new information: contractor relation CON
- $\succ'$: maximal subset of $\succ$ disjoint with CON



Substitutability [BGS06]

Substitutability [BGS06]

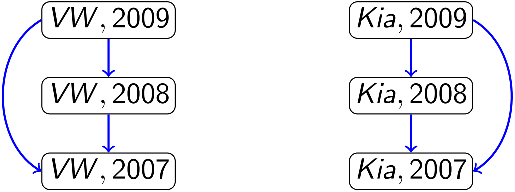
Setting

- new information: set of indifference pairs
- additional preferences are added to convert indifference to restricted indifference
- achieving object substitutability

Substitutability [BGS06]

Setting

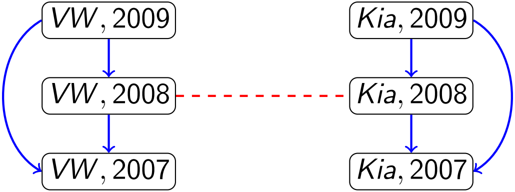
- new information: set of indifference pairs
- additional preferences are added to convert indifference to restricted indifference
- achieving object substitutability



Substitutability [BGS06]

Setting

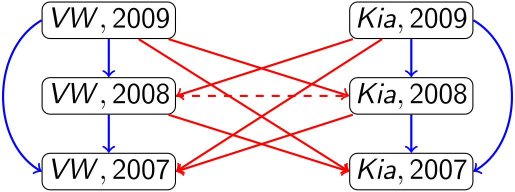
- new information: set of indifference pairs
- additional preferences are added to convert indifference to restricted indifference
- achieving object substitutability



Substitutability [BGS06]

Setting

- new information: set of indifference pairs
- additional preferences are added to convert indifference to restricted indifference
- achieving object substitutability



Part IV

Advanced topics

Outline of Part IV

Prospective research topics

Prospective research topics

Definability

Given a preference relation $\succ_C$, how to construct a definition of a scoring function F representing $\succ_C$, if such a function exists?

Prospective research topics

Definability

Given a preference relation $\succ_C$, how to construct a definition of a scoring function F representing $\succ_C$, if such a function exists?

Extrinsic preference relations

Preference relations that are not fully defined by tuple contents:

$$x \succ y \equiv BMW(x) \wedge Kia(y)$$

where BMW and Kia are database relations.

Prospective research topics

Definability

Given a preference relation $\succ_C$, how to construct a definition of a scoring function F representing $\succ_C$, if such a function exists?

Extrinsic preference relations

Preference relations that are not fully defined by tuple contents:

$$x \succ y \equiv BMW(x) \wedge Kia(y)$$

where BMW and Kia are database relations.

Incomplete preferences

- tuple scores and probabilities [SIC08, ZC08]
- uncertain tuple scores
- disjunctive preferences: $a \succ b \vee a \succ c$

Preference modification

- beyond revision and contraction: merging, arbitration,...
- general parametric framework?
- conflict resolution

Preference modification

- beyond revision and contraction: merging, arbitration,...
- general parametric framework?
- conflict resolution

Variations

- preference and similarity: *"find the objects similar to one of the best objects"*

Preference modification

- beyond revision and contraction: merging, arbitration,...
- general parametric framework?
- conflict resolution

Variations

- preference and similarity: *"find the objects similar to one of the best objects"*

Applications

- preference queries as decision components: workflows, event systems
- personalization of query results
- preference negotiation: applying contraction

Acknowledgments

Denis Mindolin
Sławek Staworko
Xi Zhang

M. Baudinet, J. Chomicki, and P. Wolper.
Constraint-Generating Dependencies.
Journal of Computer and System Sciences, 59:94–115, 1999.
Preliminary version in ICDT'95.

W-T. Balke, U. Güntzer, and W. Siberski.
Exploiting Indifference for Customization of Partial Order Skylines.
In *International Database Engineering and Applications Symposium (IDEAS)*, pages 80–88, 2006.

S. Börzsönyi, D. Kossmann, and K. Stocker.
The Skyline Operator.
In *IEEE International Conference on Data Engineering (ICDE)*, pages 421–430, 2001.

J. Chomicki, P. Godfrey, J. Gryz, and D. Liang.
Skyline with Presorting.
In *IEEE International Conference on Data Engineering (ICDE)*, 2003.
Poster.

J. Chomicki.

Preference Formulas in Relational Queries.
ACM Transactions on Database Systems, 28(4):427–466, December 2003.

J. Chomicki.
Database Querying under Changing Preferences.
Annals of Mathematics and Artificial Intelligence, 50(1-2):79–109, 2007.

J. Chomicki.
Semantic optimization techniques for preference queries.
Information Systems, 32(5):660–674, 2007.

R. Fagin, A. Lotem, and M. Naor.
Optimal Aggregation Algorithms for Middleware.
Journal of Computer and System Sciences, 66(4):614–656, 2003.

P. Godfrey, R. Shipley, and J. Gryz.
Algorithms and Analyses for Maximal Vector Computation.
VLDB Journal, 16:5–28, 2007.

W. Kießling.

Foundations of Preferences in Database Systems.
In *International Conference on Very Large Data Bases (VLDB)*, pages 311–322, 2002.

W. Kießling and G. Köstler.
Preference SQL - Design, Implementation, Experience.
In *International Conference on Very Large Data Bases (VLDB)*, pages 990–1001, 2002.

C. Li, K. C-C. Chang, I.F. Ilyas, and S. Song.
RankSQL: Query Algebra and Optimization for Relational Top-k Queries.
In *ACM SIGMOD International Conference on Management of Data*, pages 131–142, 2005.

D. Mindolin and J. Chomicki.
Maximal Contraction of Preference Relations.
In *National Conference on Artificial Intelligence*, pages 492–497, 2008.

M.A. Soliman, I.F. Ilyas, and K. C-C. Chang.
Probabilistic Top-k and Ranking-Aggregate Queries.

ACM Transactions on Database Systems, 33(3):13:1–13:54, 2008.

X. Zhang and J. Chomicki.
On the Semantics and Evaluation of Top-k Queries in Probabilistic Databases.
In *International Workshop on Database Ranking (DBRank)*. IEEE Computer Society (ICDE Workshops), 2008.

X. Zhang and Z. M. Ozsoyoglu.
Implication and Referential Constraints: A New Formal Reasoning.
IEEE Transactions on Knowledge and Data Engineering, 9(6):894–910, 1997.