Scores and weights are not the whole story

- Nowadays, scores and weights are the rule of choice if one wants to rank objects according to user preferences.
- However, scores and weights have a limited expressive power, since they can only capture those user preferences that “translates into numbers”, which is not always the case (or, at least, doing so is not so natural!)
  - "I prefer having white wine with fish and red wine with meat”
- The study of what are known as qualitative preferences has its roots in the field of economy, in particular decision theory, where scores are usually called “utilities”
  - For more information and references, see the paper by P. Fishburn [Fis99] on the web site

Remark: In the following, when talking about a scoring function S, we just require that S is a function that assigns to each object o a numerical score, S(o)
- Thus, our arguments do not necessarily require “aggregation of partial scores”
The voters’ paradox

- Consider 3 friends (Ann, Joe, and Tom) who rank, each according to his/her own preferences, 3 movies: M1, M2, and M3.
- In order to reach some consensus, they decide to integrate their preferences using the following “majority rule”:

  we collectively prefer Mi over Mj
  if at least 2 of us have ranked Mi higher than Mj

\[
\begin{array}{ccc}
\text{Ann} & \text{Joe} & \text{Tom} \\
M1 & M3 & M2 \\
M2 & M1 & M3 \\
M3 & M2 & M1 \\
\end{array}
\]

\[
\rightarrow M1 \text{ is preferable to } M2 \\
\rightarrow M2 \text{ is preferable to } M3 \\
\rightarrow M3 \text{ is preferable to } M1
\]

No scoring function can be defined!

Irrational Behavior

(this example can be found in [Fis99])

- Consider the lottery \((a, p)\), which pays \(€a\) with probability \(p\) and nothing otherwise.

  Given two lotteries, which one will you choose to play?

- Many people(*) exhibit the following cyclic pattern of preferences:
  - \((€500, 7/24)\) preferable to \((€475, 8/24)\)
  - \((€475, 8/24)\) preferable to \((€450, 9/24)\)
  - \((€450, 9/24)\) preferable to \((€425, 10/24)\)
  - \((€425, 10/24)\) preferable to \((€400, 11/24)\)
  - \((€400, 11/24)\) preferable to \((€500, 7/24)\)

A non-paradoxical case

Consider the following table:

<table>
<thead>
<tr>
<th>ID</th>
<th>Movie</th>
<th>Cinema</th>
<th>Price</th>
</tr>
</thead>
<tbody>
<tr>
<td>01</td>
<td>2001 A Space Odissey</td>
<td>Admiral</td>
<td>10</td>
</tr>
<tr>
<td>02</td>
<td>2001 A Space Odissey</td>
<td>Astra1</td>
<td>12</td>
</tr>
<tr>
<td>03</td>
<td>Wide Eyes Shut</td>
<td>Astra2</td>
<td>9</td>
</tr>
<tr>
<td>04</td>
<td>Wide Eyes Shut</td>
<td>Odeon1</td>
<td>10</td>
</tr>
<tr>
<td>05</td>
<td>Shining</td>
<td>Odeon2</td>
<td>12</td>
</tr>
</tbody>
</table>

and the preference:

given 2 cinemas C1 and C2, I prefer C1 to C2 iff they show the same movie and C1 costs less than C2

We have that o1 is preferred to o2 and o3 to o4; no other preferences can be derived.

Thus, a hypothetical scoring function S should assign an equal score to, say, o3 and o1, and to o3 and o2.

This is because there is no preference between o3 and the first two tuples.

This is impossible: S(o1) = S(o2) = S(o3) contradicts S(o1) > S(o2)!

Qualitative preferences

In order to go beyond scores and weights, we have just to realize that they are only a "quantitative" mean to define preferences.

A much more general (thus, powerful) approach is to consider so-called qualitative preferences.

With qualitative preferences we just require that, given two objects o1 and o2, there exists some criterion to determine whether o1 is preferred to o2 or not.

Since, when a scoring function is available, we prefer o1 to o2 iff S(o1) > S(o2), this shows that qualitative preferences are indeed a generalization of quantitative ones.

Qualitative preferences are a relatively new subject in the context of data management, with "personalization of e-services" being a major motivation to their investigation…
A 1st game with qualitative preferences…

- This evening I would like to go out for dinner
- It’s a special occasion, thus I’m willing to spend even up to 100 €, provided we go to a nice place (good atmosphere, good service and candle-lights), otherwise, say, 50 € would be the ideal target budget
- However, she really likes fish (which is quite expensive)
- As to the location, it would be better not to go downtown (too crowded), she would love a place over the hills
- If the road is not too bad, I could also consider travelling for 1 hour, otherwise it would be preferable to travel for no more than ½ hour, say, so that coming back would be easier
- Formal dressing should not be required
- …
- Ok, let’s start browsing the Yellow Pages…

A 2nd game with qualitative preferences…

- I would like to buy a used car
- I definitely do not like SUV’s and would like to spend about 8,000 €
- Less important to me is the mileage
- Given this, it would be nice if the color is red and if the nominal fuel consumption is no more than 7 litres/100 km
- …
Preferences relations

- Consider a relation \( R(A_1,A_2,\ldots,A_m) \), and let
  \[ \text{Dom}(R) = \text{Dom}(A_1) \times \text{Dom}(A_2) \times \cdots \times \text{Dom}(A_m) \]
  be the domain of values of \( R \) (\( \text{Dom}(A_i) \) being the domain of \( A_i \))

- A preference relation \( \succ \) over \( R \) (also called a preference system) is a subset of \( \text{Dom}(R) \times \text{Dom}(R) \), that is, a set of pairs of tuples over \( R \)

- If \((o_1,o_2) \in \succ\), we also write \( o_1 \succ o_2 \) and say that \( o_1 \) is preferred to \( o_2 \) (also: \( o_1 \) dominates \( o_2 \))

- Graphically, we can represent a preference relation as a directed graph \( G_f(V,E) \), with \( V = \text{set of objects} \) and \( E = \{(o_1,o_2): o_1 \succ o_2 \} \)

### Properties of a preference relation

- As any relation, a preference relation \( \succ \) can be characterized in terms of some basic properties:
  - **Irreflexivity**: \( \forall o: \not (o \succ o) = o \not\succ o \)
  - **Transitivity**: \( \forall o_1,o_2,o_3: (o_1 \succ o_2, o_2 \succ o_3) \Rightarrow o_1 \succ o_3 \)
  - **Asymmetry**: \( \forall o_1,o_2: o_1 \succ o_2 \Rightarrow o_2 \not\succ o_1 \)

  Note that transitivity and irreflexivity together imply asymmetry

- As the voters’ paradox shows, it is not so strange to have cyclic preference relations
- However, in most relevant cases we have that \( \succ \) is a:

  **Strict partial order**:
  - A preference relation is a strict partial order (s.p.o.) if it is transitive and irreflexive (thus, asymmetric)
  - …indeed, transitivity is not a so strict requirement, as we will see…
Hasse diagrams

- If $\succ$ is transitive we can represent the corresponding preference graph in a “transitively-reduced” form, thus omitting all the edges that can be obtained by applying the transitivity rule.

- If $\succ$ is a.s.p.o., and assuming that “$o_1$ above $o_2$” means $o_1 \succ o_2$, we can also avoid drawing directed edges, and obtain the so-called Hasse diagram of $\succ$.

Indifference relations

- When we have both $o_1 \not\succ o_2$ and $o_2 \not\succ o_1$, we say that $o_1$ and $o_2$ are indifferent, written $o_1 \sim o_2$.
  - E.g., in the movies example we have $o_1 \sim o_3$, $o_2 \sim o_3$, etc..

- Since $\sim$ is a relation (called indifference relation), it can be characterized in terms of the properties it has (irreflexive? transitive? asymmetric?)

- In particular, it can be proved that:

  Representability with a scoring function:
  - A preference relation can be represented by a scoring function only if it is a weak order (w.o.), that is, a strict partial order whose corresponding indifference relation is transitive.

  - Note that a linear (total) order is a particular case of weak order for which there are no ties: $S(o_1) = S(o_2) \Rightarrow o_1 = o_2$.
Preference relations and scoring functions

- Consider again the Movies table:

<table>
<thead>
<tr>
<th>ID</th>
<th>Movie</th>
<th>Cinema</th>
<th>Price</th>
</tr>
</thead>
<tbody>
<tr>
<td>01</td>
<td>2001 A Space Odyssey</td>
<td>Admiral</td>
<td>10</td>
</tr>
<tr>
<td>02</td>
<td>2001 A Space Odyssey</td>
<td>Astra1</td>
<td>12</td>
</tr>
<tr>
<td>03</td>
<td>Wide Eyes Shut</td>
<td>Astra2</td>
<td>9</td>
</tr>
<tr>
<td>04</td>
<td>Wide Eyes Shut</td>
<td>Odeon1</td>
<td>10</td>
</tr>
<tr>
<td>05</td>
<td>Shining</td>
<td>Odeon2</td>
<td>12</td>
</tr>
</tbody>
</table>

- We have

  \[ o_1 \succ o_2, \ o_1 \sim o_3, \ o_2 \sim o_3 \]

  which is sufficient to conclude that \( \sim \) is not transitive.

- Intuitively, when \( \sim \) is transitive, it induces an equivalence relation that can be used to assign the same score to all the equivalent objects.

\[ S(o_1) = S(o_4) \]
\[ \quad > \]
\[ S(o_2) = S(o_5) = S(o_6) \]
\[ \quad > \]
\[ S(o_3) \]

A wrong argumentation

- Wait, if we have the Movies table:

  its Hasse diagram is:

\[ o_1 \quad o_3 \quad o_5 \]
\[ \quad o_2 \quad o_4 \]

- Thus, we could define a scoring function \( S \) that makes \( o_1, o_3 \) and \( o_5 \) the "top" objects, that is, \( S(o_1) = S(o_3) = S(o_5) > S(o_2) = S(o_4) \)

- What's wrong about this?

Answer: assume \( o_1 \) is deleted:

Your s.f. \( S \), no matter how it is defined, still yields:

\[ S(o_3) = S(o_5) > S(o_2) = S(o_4) \]

thus \( o_2 \) is not one of the "top" objects!
On weak orders and scoring functions

- Not every weak order can be represented by a scoring function
- A sufficient condition is that \( \text{Dom}(R) \) be countable
- The classical counterexample (see also [Fis99]) goes as follows:

Consider the order \( L \) on \([0,1]^2 \subset \mathbb{R}^2\) (which is uncountable), defined by:

\[(x_1,y_1) \succ (x_2,y_2) \text{ if } x_1 > x_2 \text{ or } x_1 = x_2 \text{ and } y_1 > y_2.\]

Clearly, \( L \) is a weak order (it is also a total order).

Assume there exists a scoring function \( S \) for \( L \). This implies that:

\[S(x_1,1) > S(x_1,0) > S(x_2,1) > S(x_1,0) \text{ whenever } x_1 > x_2.\]

Each interval \((S(x,0), S(x,1))\) will then contain a (different) rational number, \( q(x) \).

The function \( q \) maps from the real interval \([0,1]\) to rational numbers, which leads to the contradiction that the countable set of rational numbers is uncountable.

- On the other hand, there are w.o.’s on uncountable domains that can be represented by a scoring function (e.g. \( > \) on the real line)

Ranking without scores

- If we don’t have scores anymore, it is necessary to slightly change (again!) our point of view about the result of a query
- We still insist to have a ranked list of tuples, however now we have to find another way to define the objects’ ranks
- Indeed, this is not particularly difficult, since a partial order, by definition, induces an order over the objects

- We depart from the view that the goodness of an object depends (only) on the object itself (i.e., on its score);
- Rather, it is something that, in general, might depend on the whole content of the DB (holistic view)

Absolute goodness \[\rightarrow\] Relative goodness

- As the previous example shows, there is a “natural” agreement on which are the (relative) “top” objects…
Best-Matches-Only (BMO) queries

- As a first step, we precisely define the so-called Best-Matches-Only (BMO) queries [Cho02, Kie02, TC02]:

**BMO queries:**
- Given a relation $R$ and a preference relation $\succ$ over $R$, a Best-Matches-Only (BMO) query $Q$ returns all the undominated objects $o$ in $R$, that is, $o$ belongs to the result of $Q$ iff for no object $o'$ in $R$ it is $o' \succ o$.

- [Cho02] and [TC02] have independently introduced equivalent relational operators, respectively called Winnow and Best, to support BMO queries:

$$\text{Winnow}_w(R) = \text{Best}_w(R) = \beta_w(R) = \{o \in R \mid \forall o' \in R: o' \succ o\}$$

Ranking

- Ranking of tuples can be easily obtained by iterating the Best (Winnow) operator.
- Define:

  $$\beta^1_w(R) = \beta_w(R)$$
  $$\beta^2_w(R) = \beta_w(R \setminus \beta^1_w(R))$$
  $$\beta^3_w(R) = \beta_w(R \setminus \beta^1_w(R) \setminus \beta^2_w(R))$$
  ...$

- Thus, $\beta^1_w(R)$ are the "top" objects, $\beta^2_w(R)$ are the "2nd" choices, and so on...

$$\beta^1_w(R) = \{o_1, o_4\}$$
$$\beta^2_w(R) = \{o_2\}$$
$$\beta^3_w(R) = \{o_3, o_5\}$$
Basic properties of the Best operator

- If \( \succ \) is a strict partial order then:
  - \( \beta_\succ(R) \) is always non-empty if \( R \) is non-empty (best objects always exist)
  - For each object \( o \in R \) there is a level \( i \) such that \( o \in \beta_\succ(R) \)
- If \( \succ \) is not a strict partial order, then we might well have \( \beta_\succ(R) = \emptyset \), i.e. no undominated object exists

In this case a possible solution is to take all objects in the "top cycles"
- E.g., \( o_1, o_2, \) and \( o_3 \) are "equally good", and all better than \( o_4 \) and \( o_5 \)

Composition of preferences

- One of the most appealing aspects of qualitative preferences is that they provide a great flexibility when we come to consider how different preference relations may be composed to yield a composite preference specification
- It has to be remarked, however, that if we insist to obtain a strict partial order, then some composition rules cannot be allowed
  - E.g.: reconsider the voters’ paradox: the preferences of each friend lead to an s.p.o., their combination through the “majority rule” is not an s.p.o.
- What if we take the union of 2 or more preference relations? The intersection? The difference?
- What if one preference relation is “more important” than another one?

Further, it is important to distinguish between composition of multiple preferences over the same relation (set of attributes) and composition over different relations
**Set-theoretic compositions: Union**

- Consider 2 preference relations $\succ_1$ and $\succ_2$, both over $R$, and assume that they are both strict partial orders. Their composition is denoted $\succ_{1,2}$.

**Union ($\succ_{1,2} = \succ_1 \cup \succ_2$)**

- The composite preference relation is **not a strict partial order**, since **asymmetry and transitivity are not preserved**. Graphically, we might have:

  \[
  \begin{array}{c}
    \circlearrowleft \circlearrowright \\
    o_1 & o_3 \\
    \downarrow & \downarrow \\
    o_2 \end{array} \cup 
  \begin{array}{c}
    \circlearrowleft \\
    o_1 & o_3 \\
    \downarrow \\
    o_2 \end{array} = 
  \begin{array}{c}
    \circlearrowleft \circlearrowright \\
    o_1 & o_3 \\
    \downarrow & \downarrow \\
    o_2 \end{array}
  \]

  Note that this is not transitive ($o_3$ is not preferred to $o_2$)

- Note that even when both preference relations are **weak orders** (i.e., representable by some scoring function), their union is not guaranteed to be a strict partial order

**Set-theoretic compositions: Intersection**

**Intersection ($\succ_{1,2} = \succ_1 \cap \succ_2$)**

- The composite preference relation is **still an s.p.o.** As an example:

  \[
  \begin{array}{c}
    \circlearrowleft \circlearrowright \\
    o_1 & o_4 \\
    \downarrow & \downarrow \\
    o_2 & o_3 \end{array} \cap 
  \begin{array}{c}
    \circlearrowleft \circlearrowright \\
    o_1 & o_4 \\
    \downarrow & \downarrow \\
    o_2 & o_3 \end{array} = 
  \begin{array}{c}
    \circlearrowleft \circlearrowright \\
    o_1 & o_4 \\
    \downarrow & \downarrow \\
    o_2 & o_3 \end{array}
  \]

  Remind: the inputs are assumed to be s.p.o.’s: dotted edges are implicit in the graph representation

- Intuitively: with intersection the result is the set of preferences on which the two inputs agree, thus it cannot violate any of the properties of an s.p.o.

  **Exercise**: demonstrate that intersection preserves transitivity

- On the other hand, when both preference relations are **weak orders**, their intersection is **not** (in general, it is a strict partial order)

  \[
  \begin{array}{c}
    \circlearrowleft \circlearrowright \\
    o_3 & o_1 \\
    \downarrow & \downarrow \\
    o_2 \end{array} \cap 
  \begin{array}{c}
    \circlearrowleft \circlearrowright \\
    o_3 & o_1 \\
    \downarrow & \downarrow \\
    o_2 \end{array} = 
  \begin{array}{c}
    \circlearrowleft \circlearrowright \\
    o_3 & o_1 \\
    \downarrow & \downarrow \\
    o_2 \end{array}
  \]
Intersection: from s.f.’s to s.p.o.’s

- We take the intersection of the following weak orders, each represented by a scoring function:

<table>
<thead>
<tr>
<th>ObjID</th>
<th>s1</th>
</tr>
</thead>
<tbody>
<tr>
<td>o3</td>
<td>0.7</td>
</tr>
<tr>
<td>o1</td>
<td>0.6</td>
</tr>
<tr>
<td>o4</td>
<td>0.6</td>
</tr>
<tr>
<td>o2</td>
<td>0.5</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>ObjID</th>
<th>s2</th>
</tr>
</thead>
<tbody>
<tr>
<td>o2</td>
<td>0.9</td>
</tr>
<tr>
<td>o3</td>
<td>0.6</td>
</tr>
<tr>
<td>o1</td>
<td>0.4</td>
</tr>
<tr>
<td>o4</td>
<td>0.2</td>
</tr>
</tbody>
</table>

Domination is preserved iff it occurs in both “dimensions”

Set-theoretic compositions: Difference

Difference ($\succ_{1,2} = \succ_1 - \succ_2$)

- The composite preference relation is not a strict partial order anymore, since transitivity is not preserved:

  - On the other hand, if the inputs are weak orders then transitivity is preserved and the result is a strict partial order:
Prioritized composition

Prioritized composition ($\succ_{1,2} = \succ_1 \triangleright \succ_2$)

- Prioritized composition intuitively means:
  
  **look first at $\succ_1$, if no preference is given then look at $\succ_2$**

- If the inputs are **weak orders**, then the output is also a weak order (thus, it’s **ok** for combining scoring functions!)
- However, if the inputs are generic **strict partial orders**, then the output needs not to be an s.p.o., since **transitivity** is not preserved

Prioritization of scoring functions

- We combine the following scoring functions, giving first priority to the first s.f. and then to the second one:

<table>
<thead>
<tr>
<th>ObjID</th>
<th>s1</th>
<th>s2</th>
</tr>
</thead>
<tbody>
<tr>
<td>o3</td>
<td>0.7</td>
<td>0.9</td>
</tr>
<tr>
<td>o1</td>
<td>0.6</td>
<td>0.9</td>
</tr>
<tr>
<td>o4</td>
<td>0.6</td>
<td>0.4</td>
</tr>
<tr>
<td>o2</td>
<td>0.5</td>
<td>0.2</td>
</tr>
</tbody>
</table>

Priority is given to s1

Priority is given to s2
Prioritization of partial orders

Consider a set of hotels, each with a price (P), a number of stars (S), distance from the town center (D), and number of rooms (R).

Let $\succ_1$ be defined as: "prefer hotel H1 to H2 iff $H1.P \leq H2.P$ and $H1.S \geq H2.S$, with strict inequality for at least one of the two"

Let $\succ_2$ be defined as: "prefer H1 to H2 iff $H1.D \leq H2.D$ and $H1.R \leq H2.R$, with strict inequality for at least one of the two"

<table>
<thead>
<tr>
<th>Hotel</th>
<th>Price</th>
<th>Stars</th>
<th>Distance</th>
<th>Rooms</th>
</tr>
</thead>
<tbody>
<tr>
<td>H1</td>
<td>30 €</td>
<td>2</td>
<td>4 km</td>
<td>30</td>
</tr>
<tr>
<td>H2</td>
<td>35 €</td>
<td>1</td>
<td>2 km</td>
<td>20</td>
</tr>
<tr>
<td>H3</td>
<td>60 €</td>
<td>3</td>
<td>1 km</td>
<td>100</td>
</tr>
<tr>
<td>H4</td>
<td>40 €</td>
<td>4</td>
<td>6 km</td>
<td>50</td>
</tr>
</tbody>
</table>

Although H1 (H2) is preferable to H4 (considering D and R), and H4 to H3 (considering P and S), we cannot say that H1 (H2) is preferable to H3!

Good and cheap!
Small and central!

Pareto composition

Pareto composition ($\otimes$) is defined on the Cartesian product of two schemas $R_1$ and $R_2$, each coming with its own preference relation, $\succ_1$ and $\succ_2$, respectively.

The intuitive meaning is:
prefer $p = (o_1,o_2)$ to $p' = (o_1',o_2')$ iff $p$ is not dominated by $p'$ neither in $\succ_1$ nor in $\succ_2$, and dominates $p'$ in at least one of the two cases.

$$(o_1,o_2) \succ_1 \otimes \succ_2 (o_1',o_2') = (o_1' \succ_1 o_1) \land (o_2' \succ_2 o_2) \land (o_1' \succ_1 o_2' \succ_2 o_2')$$

If the inputs are weak orders, then the result is a strict partial order.

On the other hand, if the inputs are strict partial orders, transitivity is not preserved.
### Specification of preference relations

Two main approaches have been pioneered in the DB field

- **Logical** (J. Chomicki [Cho02]):
  
  First-order formula $P$ with built-in predicates
  
  $o \succ_P o' \iff P(o, o')$

  $o = (\text{rest, price, rating})$
  
  $o' = (\text{rest', price', rating'})$
  
  $P = (\text{price < price'})$ and $(\text{rating} \geq \text{rating'})$

  *prefer a restaurant if it has a lower price and a not worse rating*

- **Algebraic** (W. Kiessling [Kie02])
  
  - Base preferences + Composition operators
  
  - Less powerful but more intuitive than first order formulas

### Algebraic specification (1)

- A slightly modified version of (part of) Kiessling algebra

1. **Numerical base preferences** ($E$ is a numerical expression):

<table>
<thead>
<tr>
<th>Constructor</th>
<th>Comments</th>
<th>Examples</th>
</tr>
</thead>
<tbody>
<tr>
<td>High($E$)</td>
<td>higher values are better</td>
<td>High(Rating)</td>
</tr>
<tr>
<td>Low($E$)</td>
<td>lower values are better</td>
<td>Low(10*Price + Rooms)</td>
</tr>
<tr>
<td>Around($E, v$)</td>
<td>$v$ is a “target value”</td>
<td>Around(Price, 40 €)</td>
</tr>
<tr>
<td>Between($E, [v_1, v_2]$)</td>
<td>$[v_1, v_2]$ is a “target interval”</td>
<td>Between(Price,[30 €,40 €])</td>
</tr>
</tbody>
</table>

- Notice that $\text{High}(E) = \text{Low}(-E)$ and $\text{Around}(E,v) = \text{Low}(|E-v|)$

- Between($E, [v_1, v_2]$):
  
  - all values within the target interval are indifferent
  
  - $o$ is better than $o'$ iff $E(o)$ is closer than $E(o')$ to $[v_1, v_2]$

- In all cases we obtain a **weak order**
2. Boolean base preferences (E is a Boolean expression):

<table>
<thead>
<tr>
<th>Constructor</th>
<th>Comments</th>
<th>Examples</th>
</tr>
</thead>
<tbody>
<tr>
<td>Pos(E)</td>
<td>values satisfying E are better</td>
<td>Pos(Price &lt; 30 €)</td>
</tr>
<tr>
<td>Neg(E)</td>
<td>values not satisfying E are better</td>
<td>Neg(Cuisine='chinese' AND Price &gt; 20 €)</td>
</tr>
</tbody>
</table>

- Notice that Neg(E) = Pos(not(E))
- In both cases, we obtain a weak order with 2 levels

![Diagram showing Pos(E) and Neg(E) with examples E(o)=true and E(o)=false]

A distinguishing feature of the algebra is that it always yields an s.p.o.
Composition operators, such as Pareto and prioritization, are however defined in a more restrictive way
To avoid any confusion, we call them Pareto accumulation and Prioritized accumulation, respectively
A basic notion for their definition is that of substitutable values
Substitutable values

Substitutable Values:
- We say that two objects/values $o_1$ and $o_2$ are substitutable if they:
  - Are dominated by the same objects
  - Dominate the same objects

It is easy to see that substitutability is an equivalence relation, which is always contained in $\sim$.
- Given $\succ$, we denote with $\equiv$ the corresponding SV-equivalence relation.

Pareto and prioritized accumulation

- We just need to replace indifference with SV-equivalence in both definitions.

Pareto accumulation

\[(o_1, o_2) \succ_1 \Theta_{SV} \succ_2 (o_1', o_2') = ((o_1 \succ_1 o_1' \lor o_1 \equiv_1 o_1') \land (o_2 \succ_2 o_2')) \lor ((o_1 \succ_1 o_1') \land (o_2 \succ_2 o_2' \lor o_2 \equiv_2 o_2'))\]

Prioritized accumulation

\[o_1 \succ_1 \succ_{SV} \succ_2 o_2 = (o_1 \succ_1 o_2) \lor (o_1 \equiv_1 o_2 \land o_1 \succ_2 o_2)\]

- It can be proved that the resulting preference relations are both s.p.o.'s.
- For convenience, in algebraic expression we use the symbols:
  - $\&$ for $\Theta_{SV}$
  - $\gg$ for $\succ_{SV}$
Example of preference expressions (1)

1. Low(Price) & High(Rating)
2. (Pos(Cuisine='italian') >> Neg(Price>40 €)) & Low(dist(Address,'Bologna'))
3. (Pos(Style in {SUV,coupe}) & Neg(Price>30)) >> Low(Price)
   >> (Pos(Color='red') & Low(Mileage))

- Let’s work out the 3rd expression, considering the following relation:

<table>
<thead>
<tr>
<th>CarID</th>
<th>Make</th>
<th>Model</th>
<th>Style</th>
<th>Color</th>
<th>Price</th>
<th>Mileage</th>
</tr>
</thead>
<tbody>
<tr>
<td>C1</td>
<td>Toyota</td>
<td>Corolla</td>
<td>sedan</td>
<td>Red</td>
<td>18</td>
<td>30</td>
</tr>
<tr>
<td>C2</td>
<td>BMW</td>
<td>325</td>
<td>coupe</td>
<td>Blue</td>
<td>35</td>
<td>20</td>
</tr>
<tr>
<td>C3</td>
<td>BMW</td>
<td>745</td>
<td>sedan</td>
<td>White</td>
<td>45</td>
<td>25</td>
</tr>
<tr>
<td>C4</td>
<td>Mercedes</td>
<td>CLK 5.0</td>
<td>coupe</td>
<td>Silver</td>
<td>40</td>
<td>35</td>
</tr>
<tr>
<td>C5</td>
<td>Porsche</td>
<td>Cayenne</td>
<td>SUV</td>
<td>Red</td>
<td>25</td>
<td>70</td>
</tr>
<tr>
<td>C6</td>
<td>Mercedes</td>
<td>CLK 5.0</td>
<td>coupe</td>
<td>Red</td>
<td>40</td>
<td>45</td>
</tr>
<tr>
<td>C7</td>
<td>Porsche</td>
<td>Cayenne</td>
<td>SUV</td>
<td>Black</td>
<td>30</td>
<td>60</td>
</tr>
<tr>
<td>C8</td>
<td>Nissan</td>
<td>350Z</td>
<td>coupe</td>
<td>Black</td>
<td>25</td>
<td>25</td>
</tr>
<tr>
<td>C9</td>
<td>VW</td>
<td>Passat GLS</td>
<td>sedan</td>
<td>Gray</td>
<td>15</td>
<td>35</td>
</tr>
</tbody>
</table>

Example of preference expressions (2)

- We start by considering the two most important preferences:
  Pos(Style in {SUV,coupe}) & Neg(Price>30)

- These define an s.p.o. with 4 classes of objects:
Example of preference expressions (3)

- Each class is then refined using the 2nd level preference:
  \[ \text{Low(Price)} \]
- Within the top-level class we get the weak order:

\[
\begin{array}{|c|c|c|c|c|c|}
\hline
\text{CarID} & \text{Make} & \text{Model} & \text{Style} & \text{Color} & \text{Price} & \text{Mileage} \\
\hline
C5 & Porsche & Cayenne & SUV & Red & 25 & 70 \\
C8 & Nissan & 350Z & coupe & Black & 25 & 25 \\
\hline
\end{array}
\]

Example of preference expressions (4)

- The two final preferences:
  \[ \text{Pos(Color='red')} \& \text{Low(Mileage)} \]

lead to the following (partial) preference graph:

\[
\begin{array}{|c|c|c|c|c|c|}
\hline
\text{CarID} & \text{Make} & \text{Model} & \text{Style} & \text{Color} & \text{Price} & \text{Mileage} \\
\hline
C5 & Porsche & Cayenne & SUV & Red & 25 & 70 \\
C8 & Nissan & 350Z & coupe & Black & 25 & 25 \\
\hline
C6 & Mercedes & CLK 5.0 & coupe & Red & 30 & 45 \\
C7 & Porsche & Cayenne & SUV & Black & 30 & 60 \\
\hline
\end{array}
\]
The complete preference graph is:

Example of preference expressions (5)

Preference modeling

- Given a language for expressing preferences, it is not always immediate to reason on the orders induced by different language expressions.
- For instance, consider (part of) our previous example:
  
  \[ E_1: (\text{Pos}(\text{Style in \{SUV, coupe\}}) \& \text{Neg}(\text{Price}>30)) >> \text{Low(Price)} \]

- What if we use the simplest expression:
  
  \[ E_2: \text{Pos}(\text{Style in \{SUV, coupe\}}) >> \text{Low(Price)} \]
  
  i.e., dropping the \text{Neg}(\text{Price}>30) preference?

- Let’s look at the orders corresponding to \( E_1 \) and \( E_2 \), respectively, on our sample relation….
The preference relation induced by $E_2$ is a weak order, that due to $E_1$ is not.

Although in our example the best objects are the same (namely, C5 and C8), this is not always the case.

Assume all SUV and coupe cost more than 30.

Then, $E_1$ returns:

\[
\begin{array}{|c|c|c|c|c|c|}
\hline
\text{CarID} & \text{Make} & \text{Model} & \text{Color} & \text{Price} & \text{Mileage} \\
\hline
C9 & VW & Passat GLS & Gray & 15 & 35 \\
\hline
\end{array}
\]

whereas the result of $E_2$ is:

\[
\begin{array}{|c|c|c|c|c|c|}
\hline
\text{CarID} & \text{Make} & \text{Model} & \text{Color} & \text{Price} & \text{Mileage} \\
\hline
C2 & BMW & 325 & Blue & 35 & 20 \\
\hline
\end{array}
\]

The issue of efficiently evaluating a query with qualitative preferences has been investigated since 2001.

What we see in the following are two basic approaches:

**General:**
- it can compute the result of a BMO query for any preference relation that is a strict partial order.

**Skyline queries:**
- these are a subset of BMO queries where the preference relation is the Pareto composition of a set of weak orders (thus, a strict partial order).

In both cases it has to be kept in mind that the problem is “difficult”, in the sense that the (theoretical) worst-case complexity is $\Theta(N^2)$ for a DB with $N$ objects.

Proof: just take $\succ = \emptyset$, i.e., the empty preference relation!
The Block-Nested-Loops (BNL) algorithm

- We are given a relation $R$ with $N$ tuples, a preference relation $\succ$ over $R$, $\succ$ being a strict partial order, and want to determine $\beta_\succ(R)$, i.e., all the undominated objects in $R$ according to $\succ$.

The BNL algorithm has been proposed in [BKS01] for Skyline queries, however it works for any s.p.o.

- The BNL algorithm builds on the simplest way to compute the top objects of a strict partial order (basically: a nested-loops self-join):
  - For each object $o$, compare $o$ with every other object
  - If none of them dominates $o$, then $o$ is part of the result

The logic of the BNL algorithm

- BNL allocates a buffer (window) $W$ in main memory, whose size is a design parameter
- It starts by sequentially reading the data file
- Every new object $o$ that is read from the data file is compared with the objects that are currently in $W$
  - If some objects $o'$ in $W$ dominates $o$, then $o$ is discarded
  - If $o$ dominates some object $o'$ in $W$, all such objects $o'$ are removed from $W$ and $o$ is inserted into $W$
  - If $o$ is indifferent to all objects in $W$, $o$ is inserted in $W$. However, if no space in $W$ is left, then $o$ is written to a temporary file $F$
- After all objects have been processed, if $F$ is empty the algorithm stops, otherwise a new iteration is started by taking $F$ as the input stream
- The objects that were inserted in $W$ when $F$ was empty can be immediately output, since they have been compared with all objects
BNL: an example

- Assume $W$ has size = 2

![Diagram showing an example of BNL](image)

BNL: some comments

- Experimental results in [BKS01] show that BNL is CPU-bound, i.e., its performance deteriorates if $W$ grows
- This is because in this case BNL executes too many objects’ comparisons
- On the other hand, BNL has a relatively low I/O cost

- Performance is also negatively affected by a growing size of the result
- In [BKS01], where BNL is evaluated only for Skyline queries, it is shown that this in turn depends on the number of attributes and on their correlation
  - Negatively correlated attributes, like Price and Mileage, lead to larger result sets
- [BKS01] also introduces some variants of BNL, among which BNL-sol, that manages $W$ as a self-organizing list
  - The idea is to first compare incoming objects with those in $W$ (called “killer” objects) that have been found to dominate several other objects
BNL needs transitivity

- Let’s consider again the “best hotels” example, with $\triangleright_{1,2} = \triangleright_1 \circ \triangleright_2$

<table>
<thead>
<tr>
<th>Hotel</th>
<th>Price</th>
<th>Stars</th>
<th>Distance</th>
<th>Rooms</th>
</tr>
</thead>
<tbody>
<tr>
<td>H1</td>
<td>30 €</td>
<td>2</td>
<td>4 km</td>
<td>30</td>
</tr>
<tr>
<td>H2</td>
<td>35 €</td>
<td>1</td>
<td>2 km</td>
<td>20</td>
</tr>
<tr>
<td>H4</td>
<td>40 €</td>
<td>4</td>
<td>6 km</td>
<td>50</td>
</tr>
<tr>
<td>H3</td>
<td>60 €</td>
<td>3</td>
<td>1 km</td>
<td>100</td>
</tr>
</tbody>
</table>

- Assume we read tuples in this order: H1, H2, H4, and H3
- The BNL algorithm would compute $\beta_{1,2}(\text{Hotels})$ as follows:
  - Read H1: insert in the window
  - Read H2: insert in the window
  - Read H4: discard
  - Read H3: insert in the window
- Result: H1, H2, and H3?

---

Skyline queries

- We introduce Skyline queries over $m$-dimensional attribute spaces, assuming for simplicity that the “target point” is the origin (0,0,…,0)
  - Generalization to the case when the values of some attributes need to be maximized and to arbitrary target points is immediate
  - Similarly, it is immediate to define Skyline queries over the [0,1]$^m$ score space, for which the target point is (1,1,…,1)
- Since for Skyline queries the preference relation is the Pareto composition of a set of weak orders, we have:

  **Skyline Query**
  
  - Given a relation $R(A_1,A_2,…,A_m)$
  - Determine the Skyline of $R$, that is, the set of objects $o$ such that there is no $o' \in R$:
    $$\forall j = 1,…,m: o'.A_j \leq o.A_j \land \exists i: o'.A_i < o.A_i$$

- In computational geometry, Skyline queries are also known as the “maximal vectors problem”; for multiple criteria optimization problems, their result is a set of so-called Pareto optimal solutions
A Skyline example (1)

- In the attribute space…

- In the score space…
  - No matter how we define scores, the Skyline doesn’t change!
  - I.e., the Skyline is insensitive to “stretching” of coordinates

A Skyline example (2)

- Let us see what the underlying strict partial order looks like…
**Dominance regions**

- Each object $o$ in the Skyline has an associated **dominance region**, defined as the set of points in $\text{Dom}(R)$ that are dominated by $o$.

**What’s so special about Skyline queries?**

- The relevance of Skyline queries is that each object of the Skyline is the 1-NN of the target point under a suitable chosen distance function!
- Intuitively: if $o$ is in the Skyline, there is no point “between $o$ and the target”.

**Proof:** It is sufficient to consider **weighted $L_\infty$ distance functions**.

Skyline points are also called “potential nearest neighbors” since, whatever $d$ you will use, the 1-NN will be one of them!
Computing the Skyline with R-trees

- If we have an index over the attributes of the Skyline, we can use it to avoid scanning the whole DB.
- The BBS (Branch and Bound Skyline) algorithm [PTF+03] is reminiscent of kNNOptimal, in that it accesses index nodes by increasing values of MinDist (in the following the query/target point coincides with the origin) and of next-NN, in that PQ keeps both objects and nodes.
  - For computational economy, [PTF+03] evaluates distances using L1 (Manhattan distance).
- We can make the following simple observation \((0 = (0,\ldots,0)):\)

\[
\text{Given two objects } o_1 \text{ and } o_2, \text{ if } o_1 < o_2, \text{ then } L1(0,o_1) < L1(0,o_2).
\]

- Another relevant observation is:

\[
\text{If the region } \text{Reg}(N) \text{ of node } N \text{ lies in the dominance region of an object } o, \text{ then } N \text{ cannot contain any Skyline point (we say that "o dominates N".)}
\]

- In PQ we also store key(N), i.e., the MBR of N, in order to check if N is dominated by some object o.

The BBS algorithm

**Input:** index tree with root node RN

**Output:** SL, the set of Skyline objects

1. Initialize PQ with \([\text{ptr}(RN), \text{Dom}(R), 0]\); // starts from the root node
2. SL := \(\emptyset\); // the Skyline is initially empty
3. while PQ \(\neq \emptyset\): // until the queue is not empty…
4. \([\text{ptr}(\text{Elem}), \text{key}(\text{Elem}), d_{\text{MIN}}(0, \text{Reg}(\text{Elem}))]\) := DEQUEUE(PQ);
5. if no point in SL dominates Elem then:
6. if Elem is an object o then: SL := SL \(\cup\) \{o\}
7. else: \{Read(Elem); // …node Elem might contain Skyline points
8. if Elem is a leaf then: \{ for each point o in Elem:
9. if no point in SL dominates o then:
10. ENQUEUE(PQ, [ptr(o), key(o), L1(0, key(o))]) \}
11. else: \{ for each child node Nc of Elem:
12. if no point in SL dominates Nc then:
13. ENQUEUE(PQ, [ptr(Nc), key(Nc), d_{\text{MIN}}(0, \text{Reg}(Nc))]) \}\};
14. return SL;
15. end.
BBS in action

- Nodes are numbered following the order in which they are accessed

Some experimental results (from [PTF+03])

- NN is an algorithm from [KRS02], also based on R-trees

Experimental setup
- Independent (uniform) and anti-correlated datasets
- Dimensionality $d \in [2, 5]$
- Cardinality $N=1M$ tuples
- Node size = 4Kbytes
  - $C = 204$ when $d=2$
  - $C = 94$ when $d=5$
- Pentium 4, 2.4GHz CPU
  - 512Mbytes RAM
Correctness and Optimality of BBS

- The correctness of BBS is easy to prove, since the algorithm only discards nodes that are found to be dominated by some point in the Skyline.
- An interesting observation is that, when an object \( o \) is inserted into SL, then \( o \) is guaranteed to be part of the final result (i.e., \( o \) is never removed from SL).
  - This is a direct consequence of accessing nodes by increasing values of MinDist and of inserting an object into SL only when it becomes the first element of PQ.
- Optimality of BBS (which we do not formally prove) means: BBS only reads nodes that intersect the “Skyline search region”; this is the complement of the union of the dominance regions of Skyline points.

Variants of Skyline queries

- [PTF+03] introduces some variants of basic Skyline queries:
  1. **Ranked skyline queries**
     ranking within the Skyline with a scoring function
  2. **Constrained skyline queries**
     limiting the search region
  3. **K-dominating queries**
     the \( k \) objects that dominate the largest number of other objects
Final considerations

- Although the application of qualitative preferences in DB's is a relatively new subject, it has gained increasing popularity since it is a very powerful and promising generalization of the "scores and weights" approach.
- There are a number of interesting variants of the basic scenarios we have considered here, such as:
  - Conditional preferences
  - Algorithms for non-transitive preference relations
  - Approximate algorithms for Skyline and, more in general, BMO queries
  - Preference elicitation, i.e., the process of asking the right, most effective, questions, to the user, so as to quickly narrow the search space
  - This is tightly related to the problem of designing effective user interfaces for preference specification
  - Skyline-based data analysis (e.g., which are the attributes that make an object part of the Skyline?)
  - …