Preference Relations

Prof. Paolo Ciaccia
http://www-db.deis.unibo.it/courses/SI-LS/

04_PreferenceRelations.pdf

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

Ann	Joe	Tom
M1	M3	M2
M2	M1	M3
M3	M2	M1

M1 is preferable to M2

M2 is preferable to M3

M3 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. Tversky. *Intransitivity of preferences*. Psychological Review 76 (1969), pp. 31-48

A non-paradoxical case

- Consider the following table:

ID	Movie	Cinema	Price
o1	2001 A Space Odissey	Admiral	10
o2	2001 A Space Odissey	Astra1	12
o3	Wide Eyes Shut	Astra2	9
o4	Wide Eyes Shut	Odeon1	10
o5	Shining	Odeon2	12

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) contradicits S(o1) > S(o2)!

Qualititative 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 crowdy), 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…

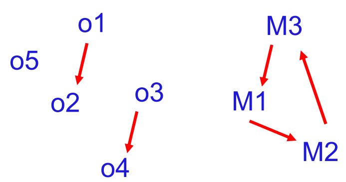
A 2nd game with qualitative preferences…

- 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(A1,A2,…,Am), and let
  Dom(R) = Dom(A1)xDom(A2)x…xDom(Am)
  be the domain of values of R (Dom(Ai) being the domain of Ai)
- A preference relation $\succ$ over R (also called a preference system) is a subset of Dom(R) x Dom(R), that is, a set of pairs of tuples over R
- If (o1,o2) $\in \succ$, we also write o1 $\succ$ o2 and say that o1 is preferred to o2 (also: o1 *dominates* o2)
- Graphically, we can represent a preference relation as a directed graph **$G_{\succ}$(V,E)**, with V = set of objects and E = {(o1,o2): o1 $\succ$ o2 }

ID	Movie	Cinema	Price
o1	2001 A Space Odissey	Admiral	10
o2	2001 A Space Odissey	Astra1	12
o3	Wide Eyes Shut	Astra2	9
o4	Wide Eyes Shut	Odeon1	10
o5	Shining	Odeon2	12



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) $\equiv$ o $\nsucc$ o
Transitivity:	$\forall$o1,o2,o3:	(o1 $\succ$ o2, o2 $\succ$ o3) $\Rightarrow$ o1 $\succ$ o3
Asymmetry:	$\forall$o1,o2:	o1 $\succ$ o2 $\Rightarrow$ o2 $\nsucc$ o1

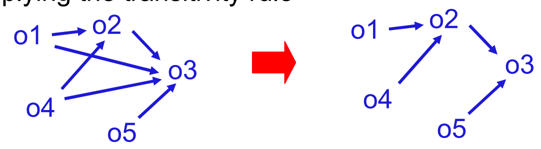
  - Note that transitivity and irreflexivity together imply asymmetry
- As the voters' paradox shows, it is not so strange to have *cyclic* preference relations
- However, in most relevant cases we have that $\succ$ is a:

Strict partial order:
- A preference relation is a strict partial order (s.p.o.) if it is transitive and irreflexive (thus, asymmetric)

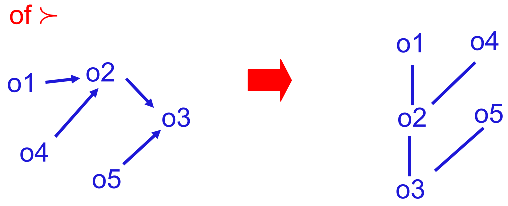
- …indeed, transitivity is not a so strict requirement, as we will see…

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 an s.p.o., and assuming that "o1 above o2" means o1 $\succ$ o2, we can also avoid drawing directed edges, and obtain the so-called Hasse diagram of $\succ$



Indifference relations

- When we have both o1 $\not\succ$ o2 and o2 $\not\succ$ o1, we say that o1 and o2 are *indifferent*, written o1 ~ o2
  - E.g., in the movies example we have o1 ~ o3, o2 ~ o3, etc..
- Since ~ 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(o1) = S(o2) $\Rightarrow$ o1 = o2

Preference relations and scoring functions

- Consider again the Movies table:

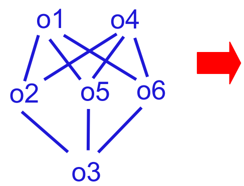
ID	Movie	Cinema	Price
o1	2001 A Space Odissey	Admiral	10
o2	2001 A Space Odissey	Astra1	12
o3	Wide Eyes Shut	Astra2	9
o4	Wide Eyes Shut	Odeon1	10
o5	Shining	Odeon2	12

- We have

  o1 ≻ o2 , o1 ~ o3, o2 ~ o3

  which is sufficient to conclude that ~ is not transitive

- Intuitively, when ~ is transitive, it induces an equivalence relation that can be used to assign the same score to all the equivalent objects



S(o1) = S(o4)

>

S(o2) = S(o5) = S(06)

>

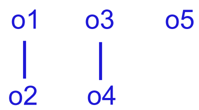
S(03)

A wrong argumentation

- Wait, if we have the Movies table:

ID	Movie	Cinema	Price
o1	2001 A Space Odissey	Admiral	10
o2	2001 A Space Odissey	Astra1	12
o3	Wide Eyes Shut	Astra2	9
o4	Wide Eyes Shut	Odeon1	10
o5	Shining	Odeon2	12

  its Hasse diagram is:



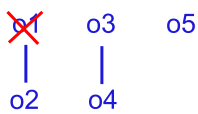
- Thus, we could define a scoring function S that makes o1, o3 and o5 the “top” objects, that is, S(o1) = S(o3) = S(o5) > S(o2) = S(o4)
- What’s wrong about this?

ID	Movie	Cinema	Price
o2	2001 A Space Odissey	Astra1	12
o3	Wide Eyes Shut	Astra2	9
o4	Wide Eyes Shut	Odeon1	10
o5	Shining	Odeon2	12

Answer: assume o1 is deleted: Your s.f. S, no matter how it is defined, still yields:

S(o3) = S(o5) > S(o2) = S(o4)

thus o2 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 Dom(R) be countable
- The classical counterexample (see also [Fis99]) goes as follows:

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

$(x1,y1) \succ (x2,y2)$ if $x1 > x2$ or $x1 = x2$ and $y1 > y2$.

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

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

$S(x1,1) > S(x1,0) > S(x2,1) > S(x1,0)$ whenever $x1 > x2$.

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)

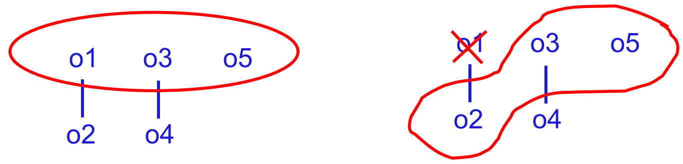
Absolute goodness ➡ Relative goodness

- 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:

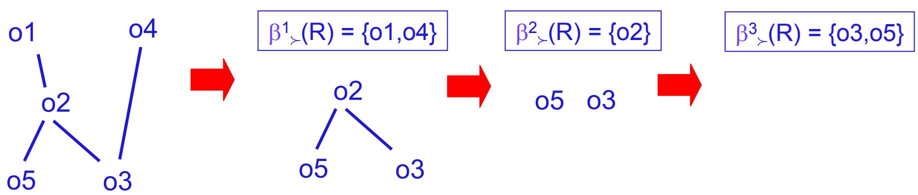
$$\mathbf{Winnow_{\succ}(R) = Best_{\succ}(R) = \beta_{\succ}(R) = \{o \in R \mid \forall o' \in R: o' \nsucc o\}}$$

Ranking

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

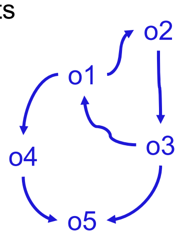
$$\beta^1_{\succ}(R) = \beta_{\succ}(R)$$
$$\beta^2_{\succ}(R) = \beta_{\succ}(R - \beta^1_{\succ}(R))$$
$$\beta^3_{\succ}(R) = \beta_{\succ}(R - \beta^1_{\succ}(R) - \beta^2_{\succ}(R))$$
$$\ldots$$

- Thus, $\beta^1_{\succ}(R)$ are the "top" objects, $\beta^2_{\succ}(R)$ are the "2nd" choices, and so on…



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 $\in \beta^{i}_{\succ}(R)$
- If $\succ$ is not a strict partial order, then we might well have $\beta_{\succ}(R) = \emptyset$, i.e. no undominated object exists



transitive and reflexive

- In this case a possible solution is to take all objects in the "top cycles"
  - E.g., o1, o2, and o3 are "equally good", and all better than o4 and o5

Composition of preferences

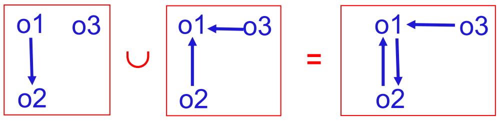
- One of the most appealing aspects of qualitative preferences is that they provide a great flexibility when we come to consider how different preference relations may be composed to yield a composite preference specification
- It has to be remarked, however, that if we insist to obtain a strict partial 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 $\succ_{1,2}$

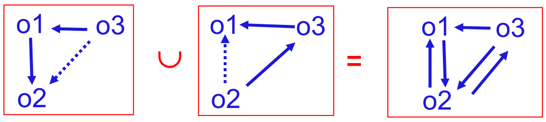
Union ($\succ_{1,2} = \succ_1 \cup \succ_2$)

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



Note that this is not transitive (o3 is not preferred to o2)

- 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

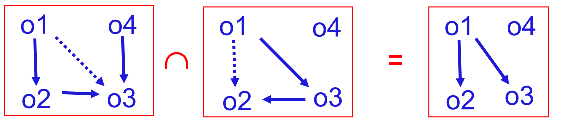


Remind: the inputs are assumed to be s.p.o.'s: dotted edges are implicit in the graph representation

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:

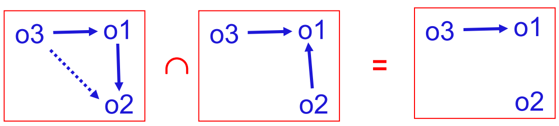


Remind: the inputs are assumed to be s.p.o.'s: dotted edges are implicit in the graph representation

- 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

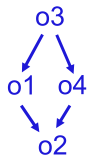
- On the other hand, when both preference relations are weak orders, their intersection is not (in general, it is a strict partial order)



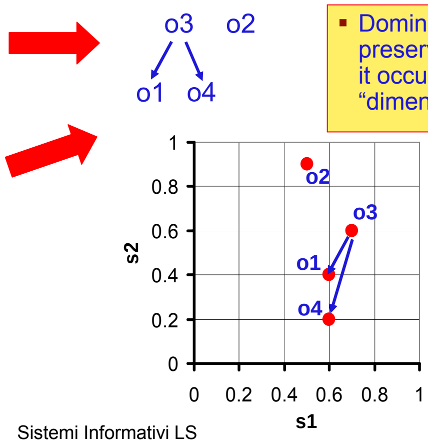
Intersection: from s.f.'s to s.p.o.'s

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

ObjID	s1
o3	0.7
o1	0.6
o4	0.6
o2	0.5



ObjID	s2
o2	0.9
o3	0.6
o1	0.4
o4	0.2

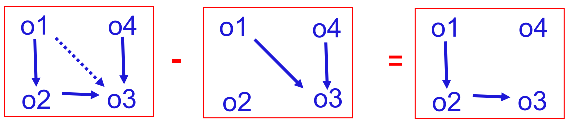


- Domination is preserved iff it occurs in both "dimensions"

Set-theoretic compositions: Difference

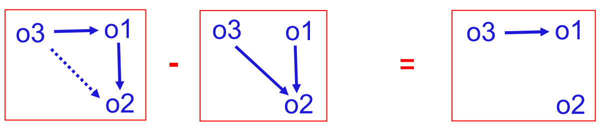
Difference $(\succ_{1,2} = \succ_1 - \succ_2)$

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



We have lost a transitive preference!

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



Prioritized composition

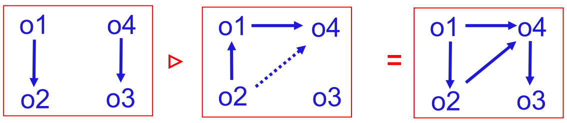
Prioritized composition ($\succ_{1,2} = \succ_1 \rhd \succ_2$)

- Prioritized composition intuitively means:

look first at $\succ_1$, if no preference is given then look at $\succ_2$

$$o1 \succ_{1,2} o2 \equiv (o1 \succ_1 o2) \vee (o1 \sim_1 o2 \wedge o1 \succ_2 o2)$$

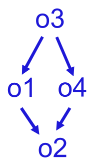
- If the inputs are weak orders, then the output is also a weak order (thus, it's ok for combining scoring functions!)
- However, if the inputs are generic strict partial orders, then the output needs not to be an s.p.o., since transitivity is not preserved



Prioritization of scoring functions

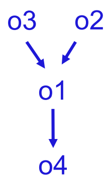
- We combine the following scoring functions, giving first priority to the first s.f. and then to the second one:

ObjID	s1
o3	0.7
o1	0.6
o4	0.6
o2	0.5



Priority is given to s1

ObjID	s2
o2	0.9
o3	0.9
o1	0.4
o4	0.2



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"

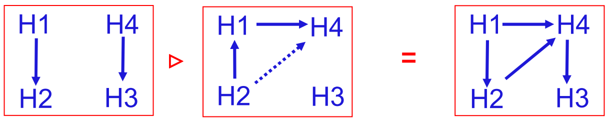
Good and cheap!

- 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"

Small and central!

Hotel	Price	Stars	Distance	Rooms
H1	30 €	2	4 km	30
H2	35 €	1	2 km	20
H3	60 €	3	1 km	100
H4	40 €	4	6 km	50

- 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!



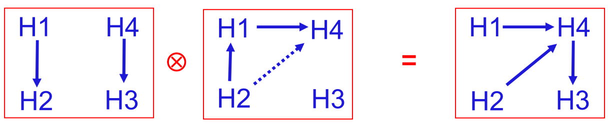
Pareto composition

- Pareto composition ($\otimes$) is defined on the Cartesian product of two schemas R1 and R2, each coming with its own preference relation, $\succ_1$ and $\succ_2$, respectively
- The intuitive meaning is:

prefer p = (o1,o2) to p' = (o1',o2') 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

$$(o1,o2) \succ_1 \otimes \succ_2 (o1',o2') \equiv (o1' \not\succ_1 o1) \wedge (o2' \not\succ_2 o2) \wedge (o1 \succ_1 o1' \vee o2 \succ_2 o2')$$

- 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 = (rest, price, rating)
  o’ = (rest’, price’, rating’)
  P = (price < price’) and (rating ≥ rating’)
  prefer a restaurant iff 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):

Constructor	Comments	Examples
High(E)	higher values are better	High(Rating)
Low(E)	lower values are better	Low(10*Price + Rooms)
Around(E,v)	v is a “target value”	Around(Price, 40 €)
Between(E,[v1,v2])	[v1,v2] is a “target interval”	Between(Price,[30 €,40 €])

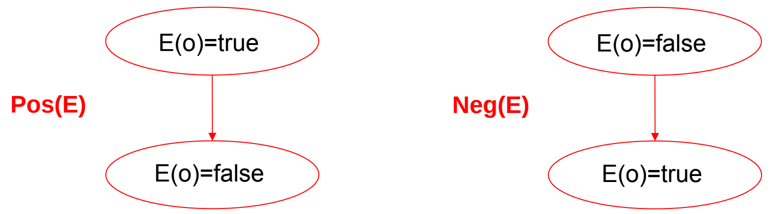
- Notice that High(E) = Low(-E) and Around(E,v) = Low(|E-v|)
- Between(E,[v1,v2]):
  - all values within the target interval are indifferent
  - o is better than o’ iff E(o) is closer than E(o’) to [v1,v2]
- In all cases we obtain a weak order

Algebraic specification (2)

2. Boolean base preferences (E is a Boolean expression):

Constructor	Comments	Examples
Pos(E)	values satisfying E are better	Pos(Price < 30 €)
Neg(E)	values not satisfying E are better	Neg(Cuisine='chinese' AND Price > 20 €)

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



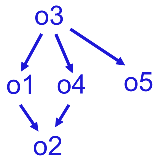
Algebraic specification (3)

- 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 o1 and o2 are substitutable iff they:
  - Are dominated by the same objects
  - Dominate the same objects



o1 and o4 are substitutable
o1 and o5 are not

- It is easy to see that substitutablity is an equivalence relation, which is always contained in ~
- Given $\succ$, we denote with $\approx$ the corresponding SV-equivalence relation

Pareto and prioritized accumulation

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

Pareto accumulation

$$(o1,o2) \succ_1 \otimes_{SV} \succ_2 (o1',o2') \equiv ((o1 \succ_1 o1' \vee o1 \approx_1 o1') \wedge (o2 \succ_2 o2')) \vee ((o1 \succ_1 o1') \wedge (o2 \succ_2 o2' \vee o2 \approx_2 o2'))$$

Prioritized accumulation

$$o1 \succ_1 \triangleright_{SV} \succ_2 o2 \equiv (o1 \succ_1 o2) \vee (o1 \approx_1 o2 \wedge o1 \succ_2 o2)$$

- 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 $\otimes_{SV}$
  - >> for $\triangleright_{SV}$

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:

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

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:

Stile in {SUV,coupe} and Price ≤ 30

CarID	Make	Model	Style	Color	Price	Mileage
C5	Porsche	Cayenne	SUV	Red	25	70
C6	Mercedes	CLK 5.0	coupe	Red	30	45
C7	Porsche	Cayenne	SUV	Black	30	60
C8	Nissan	350Z	coupe	Black	25	25

Stile not in {SUV,coupe} and Price ≤ 30

CarID	Make	Model	Style	Color	Price	Mileage
C1	Toyota	Corolla	sedan	Red	18	30
C9	VW	Passat GLS	sedan	Gray	15	35

Stile in {SUV,coupe} and Price > 30

CarID	Make	Model	Style	Color	Price	Mileage
C2	BMW	325	coupe	Blue	35	20
C4	Mercedes	CLK 5.0	coupe	Silver	40	35

Stile not in {SUV,coupe} and Price > 30

CarID	Make	Model	Style	Color	Price	Mileage
C3	BMW	745	sedan	White	45	25

Example of preference expressions (3)

- Each class is then refined using the 2nd level preference:

  Low(Price)

- Within the top-level class we get the weak order:

CarID	Make	Model	Style	Color	Price	Mileage
C5	Porsche	Cayenne	SUV	Red	25	70
C8	Nissan	350Z	coupe	Black	25	25

CarID	Make	Model	Style	Color	Price	Mileage
C6	Mercedes	CLK 5.0	coupe	Red	30	45
C7	Porsche	Cayenne	SUV	Black	30	60

Example of preference expressions (4)

- The two final preferences:

  Pos(Color=‘red’) & Low(Mileage)

  lead to the following (partial) preference graph:

CarID	Make	Model	Style	Color	Price	Mileage
C5	Porsche	Cayenne	SUV	Red	25	70

CarID	Make	Model	Style	Color	Price	Mileage
C8	Nissan	350Z	coupe	Black	25	25

CarID	Make	Model	Style	Color	Price	Mileage
C6	Mercedes	CLK 5.0	coupe	Red	30	45

CarID	Make	Model	Style	Color	Price	Mileage
C7	Porsche	Cayenne	SUV	Black	30	60

Example of preference expressions (5)

- The complete preference graph is:

CarID	Make	Model	Style	Color	Price	Mileage
C5	Porsche	Cayenne	SUV	Red	25	70

CarID	Make	Model	Style	Color	Price	Mileage
C8	Nissan	350Z	coupe	Black	25	25

CarID	Make	Model	Style	Color	Price	Mileage
C6	Mercedes	CLK 5.0	coupe	Red	30	45

CarID	Make	Model	Style	Color	Price	Mileage
C7	Porsche	Cayenne	SUV	Black	30	60

CarID	Make	Model	Style	Color	Price	Mileage
C9	VW	Passat GLS	sedan	Gray	15	35

CarID	Make	Model	Style	Color	Price	Mileage
C2	BMW	325	coupe	Blue	35	20

CarID	Make	Model	Style	Color	Price	Mileage
C1	Toyota	Corolla	sedan	Red	18	30

CarID	Make	Model	Style	Color	Price	Mileage
C4	Mercedes	CLK 5.0	coupe	Silver	40	35

CarID	Make	Model	Style	Color	Price	Mileage
C3	BMW	745	sedan	White	45	25

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:

  E1: (Pos(Style in {SUV,coupe}) & Neg(Price>30)) >> Low(Price)
- What if we use the simplest expression:

  E2: Pos(Style in {SUV,coupe}) >> Low(Price)

  i.e., dropping the Neg(Price>30) preference?
- Let's look at the orders corresponding to E1 and E2, respectively, on our sample relation….

E1: (Pos(Style in {SUV,coupe}) & Neg(Price>30)) >> Low(Price)

CarID	Make	Model	Style	Color	Price	Mileage
C5	Porsche	Cayenne	SUV	Red	25	70
C8	Nissan	350Z	coupe	Black	25	25

CarID	Make	Model	Style	Color	Price	Mileage
C6	Mercedes	CLK 5.0	coupe	Red	30	45
C7	Porsche	Cayenne	SUV	Black	30	60

CarID	Make	Model	Style	Color	Price	Mileage
C9	VW	Passat GLS	sedan	Gray	15	35

CarID	Make	Model	Style	Color	Price	Mileage
C2	BMW	325	coupe	Blue	35	20

CarID	Make	Model	Style	Color	Price	Mileage
C1	Toyota	Corolla	sedan	Red	18	30

CarID	Make	Model	Style	Color	Price	Mileage
C4	Mercedes	CLK 5.0	coupe	Silver	40	35

CarID	Make	Model	Style	Color	Price	Mileage
C3	BMW	745	sedan	White	45	25

E2: Pos(Style in {SUV,coupe}) >> Low(Price)

CarID	Make	Model	Style	Color	Price	Mileage
C5	Porsche	Cayenne	SUV	Red	25	70
C8	Nissan	350Z	coupe	Black	25	25

CarID	Make	Model	Style	Color	Price	Mileage
C6	Mercedes	CLK 5.0	coupe	Red	30	45
C7	Porsche	Cayenne	SUV	Black	30	60

CarID	Make	Model	Style	Color	Price	Mileage
C2	BMW	325	coupe	Blue	35	20

CarID	Make	Model	Style	Color	Price	Mileage
C4	Mercedes	CLK 5.0	coupe	Silver	40	35

CarID	Make	Model	Style	Color	Price	Mileage
C9	VW	Passat GLS	sedan	Gray	15	35

CarID	Make	Model	Style	Color	Price	Mileage
C1	Toyota	Corolla	sedan	Red	18	30

CarID	Make	Model	Style	Color	Price	Mileage
C3	BMW	745	sedan	White	45	25

Comments

- The preference relation induced by E2 is a weak order, that due to E1 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, E1 returns:

CarID	Make	Model	Style	Color	Price	Mileage
C9	VW	Passat GLS	sedan	Gray	15	35

CarID	Make	Model	Style	Color	Price	Mileage
C2	BMW	325	coupe	Blue	35	20

whereas the result of E2 is:

CarID	Make	Model	Style	Color	Price	Mileage
C2	BMW	325	coupe	Blue	35	20

Evaluation of queries with qualitative pref.'s

- 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.!

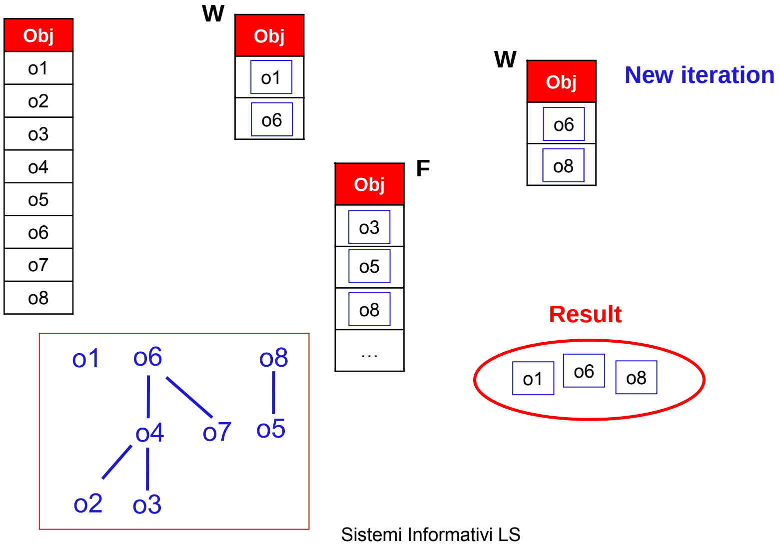
- The BNL algorithm builds on the simplest way to compute the top objects of a strict partial order (basically: a nested-loops self-join):
  - For each object o, compare o with every other object
  - If none of them dominates o, then o is part of the result

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



BNL: some comments

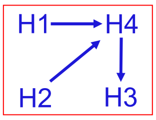
- Experimental results in [BKS01] show that BNL is CPU-bound, i.e., its performance deteriorates if W grows
- This is because in this case BNL executes too many objects' comparisons
- On the other hand, BNL has a relatively low I/O cost

- 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 $\succ_{1,2} = \succ_1 \otimes \succ_2$

Hotel	Price	Stars	Distance	Rooms
H1	30 €	2	4 km	30
H2	35 €	1	2 km	20
H4	40 €	4	6 km	50
H3	60 €	3	1 km	100



Remind: H1 $\nsucc_{1,2}$ H3
H2 $\nsucc_{1,2}$ H3

- Assume we read tuples in this order: H1, H2, H4, and H3
- The BNL algorithm would compute $\beta_{\succ 1,2}$(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:

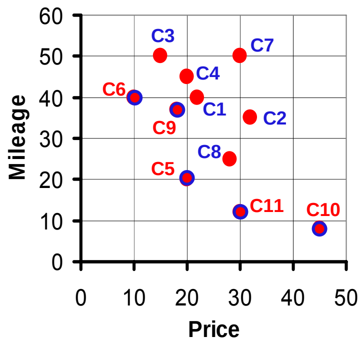
Skyline Query
- Given a relation R(A1,A2,…,Am)
- Determine the *Skyline* of R, that is, the set of objects o such that there is no o' $\in$R:

$$\forall j = 1,\dots,m: o'.Aj \leq o.Aj \wedge \exists i: o'.Ai < o.Ai$$

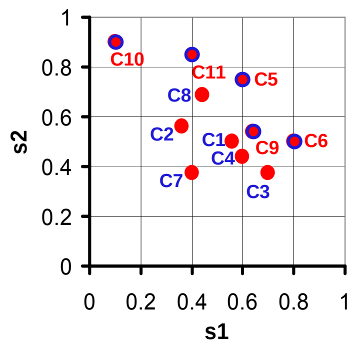
- In computational geometry, Skyline queries are also known as the "maximal vectors problem"; for multiple criteria optimization problems, their result is a set of so-called Pareto optimal solutions

A Skyline example (1)

- In the attribute space…

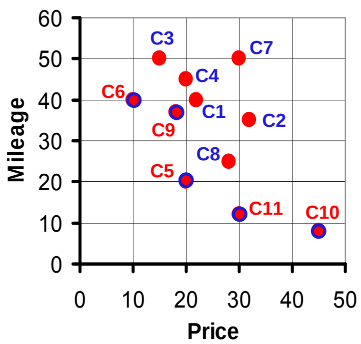


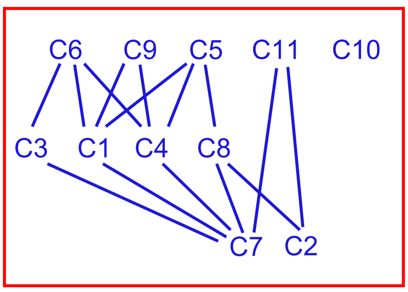
- In the score space…
  - No matter how we define scores, the Skyline doesn't change!
  - I.e., the Skyline is insensitive to "stretching" of coordinates



A Skyline example (2)

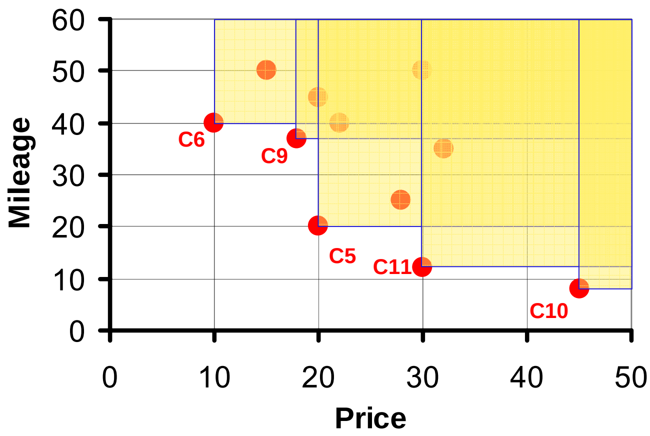
- Let us see what the underlying strict partial order looks like…





Dominance regions

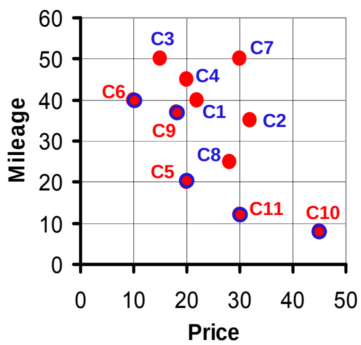
- Each object o in the Skyline has an associated dominance region, defined as the set of points in Dom(R) that are dominated by o



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_∞** distance functions ■



Skyline points are also called "potential nearest neighbors" since, whatever d you will use, the 1-NN will be one of them!

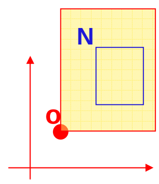
Computing the Skyline with R-trees

- 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 ($\mathbf{0}$ = (0,..,0)):

> Given two objects o1 and o2, if o1 $\succ$ o2, then L1($\mathbf{0}$,o1) < L1($\mathbf{0}$,o2)

- Another relevant observation is:

> If the region Reg(N) of node N lies in the dominance region of an object o, then N 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

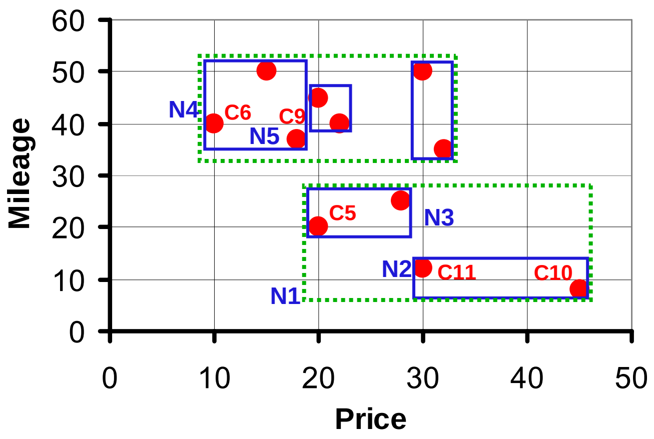
The BBS algorithm

Input: index tree with root node RN
Output: SL, the set of Skyline objects

```
Initialize PQ with [ptr(RN),Dom(R),0];     // starts from the root node
SL := ∅;                    // the Skyline is initially empty
while PQ ≠ ∅:               // until the queue is not empty…
    [ptr(Elem), key(Elem), dMIN(0,Reg(Elem))] := DEQUEUE(PQ);
    If no point in SL dominates Elem then:
        if Elem is an object o then: SL := SL ∪ {o}
            else: { Read(Elem);   // …node Elem might contain Skyline points
                if Elem is a leaf then: { for each point o in Elem:
                    if no point in SL dominates o then:
                                ENQUEUE(PQ,[ptr(o), key(o), L1(0,key(o))]) }
                            else: { for each child node Nc of Elem:
                    if no point in SL dominates Nc then:
                            ENQUEUE(PQ,[ptr(Nc), key(Nc), dMIN(0,Reg(Nc))]) }};
return SL;
end.
```

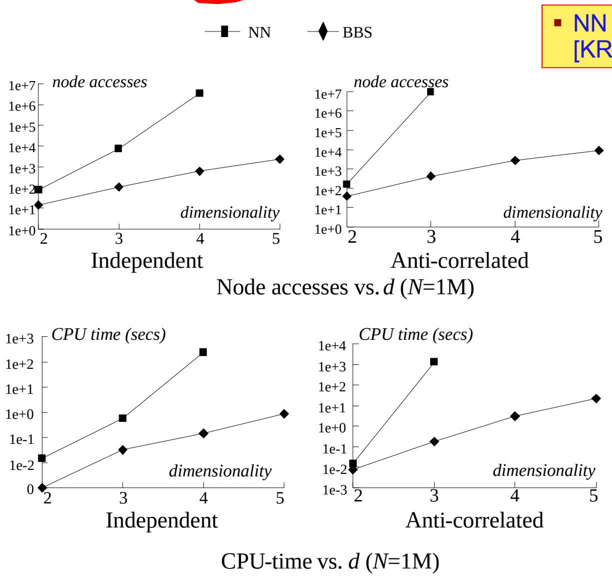
BBS in action

- 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

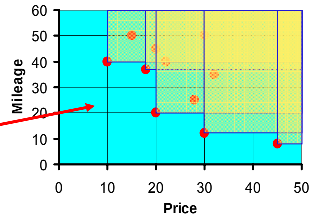
Node size = 4Kbytes
(C = 204 when d=2;
C = 94 when d=5)

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

The Skyline search region



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?)
  - …