Week 9: Knowledge Based RSs – Part I

CSX4207/ITX4207: Decision Support and Recommender Systems

ITX4287: Selected Topic in Decision Support and Recommender Systems

Asst. Prof. Dr. Rachsuda Setthawong

Objectives

- To understand the concept of knowledge based filtering approach
- To understand the necessity for this recommendation approach.
- To understand main advantages of knowledge-based recommender systems
- To be familiar with a widely used knowledge-based recommendation algorithm

Outlines

- Main idea and definition of knowledge based filtering
- Motivation and main advantages of knowledge-based recommender systems
- A Constraint Satisfaction Problem (CSP)
- Using Defaults
- Knowledge-based recommendation algorithm
  - Constraint based

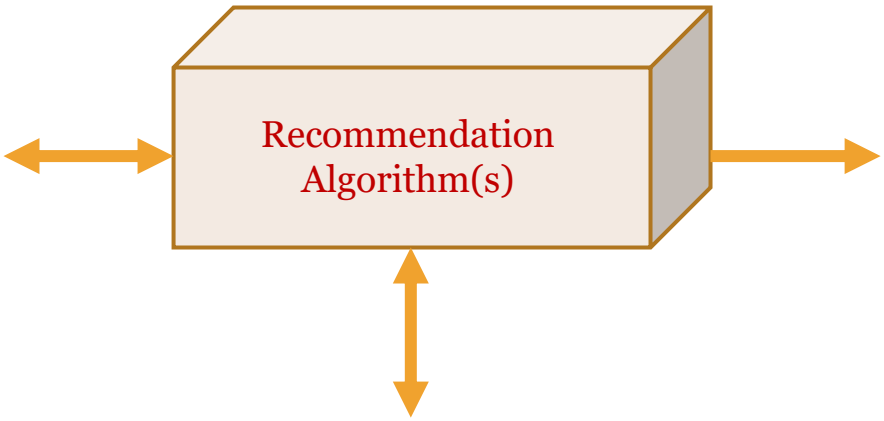
Main Idea

- To match *user requirements* with item's features.
- To use *user interaction* to **refine recommendations**.
- To predict which items the current user will most probably like.

Asst. Prof. Dr. Rachsuda Setthawong

How to Generate Recommendation Using Knowledge Based Filtering Approach

Target
User's Requirements



Recommendations' List

Product's Features

Knowledge based: "Tell me what ***fits*** my needs."

Definition

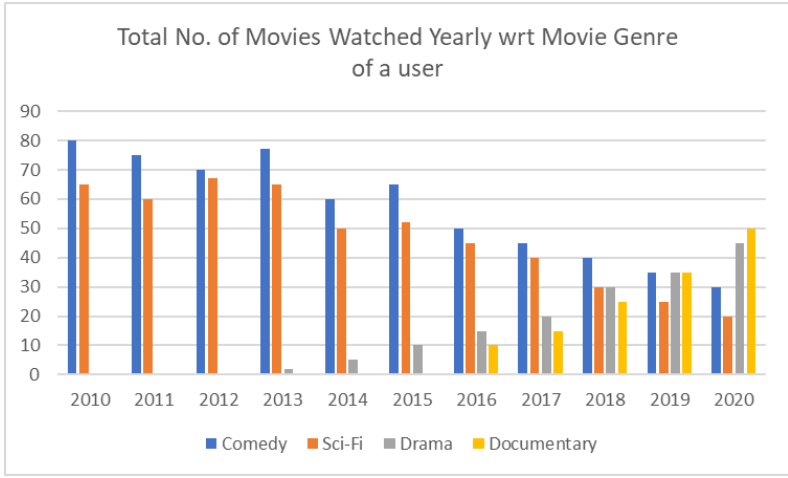
- "Recommenders that *rely on knowledge sources* NOT exploited by collaborative and content-based approaches are by default defined as knowledge-based recommenders" – *Burke (2000) and Felfernig and Burke (2008)*

Asst. Prof. Dr. Rachsuda Setthawong

Motivations

- Situations that collaborative and content-based recommender algorithms do NOT work well:
  - Low number of available ratings (e.g., buying a house)
  - Time spans (e.g., outdate ratings of *technology-related* items)
  - Changes of user preferences over times (e.g., changes in lifestyles, family situations)

- Remark: *Explicit* requirements are needed.
  - E.g.,
    - The max. price of the car is x.
    - The color should be black.



Main Advantages of Knowledge-based Recommender Systems

Limitation in collaborative based RS →

- No ratings required
  - Recommendations are calculated independently of individual user ratings.

Limitation in content-based RS →

- No ramp-up problems
  - Exploit customer requirements and items' feature.
- Highly interactive to the user

Goal

- Suggest items from a catalog that match the user's preferences or hard requirements.

Id	Price (€)	Mpix	Opt-zoom	LCD-size	Movies	Sound	Waterproof
p1	148	8	4x	2.5	no	no	yes
p2	182	8	5x	2.7	yes	yes	no
p3	189	8	10x	2.5	yes	yes	no
p4	196	10	12x	2.7	yes	no	yes
p5	151	7.1	3x	3.0	yes	yes	no
p6	199	9	3x	3.0	yes	yes	no
p7	259	10	3x	3.0	yes	yes	no
p8	278	9.1	10x	3.0	yes	yes	yes

Digital camera dataset

Asst. Prof. Dr. Rachada Setthawong
Kajornkasirat
Jiamthapthaksin, Ph.D.

Knowledge Representation and Reasoning

- How to represent user's requirements and domain knowledge to build a knowledge-based RS
  - Representation of user's requirements:
    - *Desired values* or *value ranges* for an item's features.
    - E.g.,
      - The **price** should be *lower than 300€*.
      - The **usage** of camera is for *sports photography*.

Basic Types of Knowledge-based Recommender Systems

- Constraint-based
- Case-based

A Constraint Satisfaction Problem (CSP)

- Given a-tuple (V, D, C) where
  - V is a set of *variables*
  - D is a set of finite *domains* for these variables
  - C is a set of *constraints* that describes the combinations of values the variables can simultaneously take.

- Goal: Assign a value to each variable in V in a way that *all constraints* are *satisfied*.

Asst. Prof. Dr. Rachsuda Setthawong

A Recommender's Knowledge Base (V)

- $V = V_c \cup V_{PROD}$
  - Requirements of potential customer (V_c)
  - Product properties (V_{PROD})
  - Example,

	Requirement / product property	Domain (D)
V_c	{max-price, usage, photography}	(0...1000) (digital, small-print, large-print) (sports, landscape, portrait, macro)
V_{PROD}	{price, mpix, opt-zoom, lcd-size, movies, sound, waterproof}	(0...1000) (3.0...12.0) (4x...12x) (2.5...3.0) (yes, no) (yes, no) (yes, no)

A Recommender's Knowledge Base (V) – *Cont.*

- $V = V_c \cup V_{PROD}$
  - Requirements of potential customer (V_c)
  - Product properties (V_{PROD})
  - Example,

- May NOT be the SAME variables
- May need Compatibility/Filter constraints for mapping V_c to V_c or V_c to V_{PROD}

	Requirement / product property	Domain (D)
V_c	{max-price, usage, photography}	(0...1000) (digital, small-print, large-print) (sports, landscape, portrait, macro)
V_{PROD}	{price, mpix, opt-zoom, lcd-size, movies, sound, waterproof}	(0...1000) (3.0...12.0) (4x...12x) (2.5...3.0) (yes, no) (yes, no) (yes, no)

Asst. Prof. Dr. Rachsuda Setthawong

Three Sets of Constraints (C)

- $(C = C_R \cup C_F \cup C_{PROD})$
  - Compatibility constraints (C_R)
    - **Derive customer properties** from existing ones
  - Filter constraints (C_F)
    - **Derive product properties** from customer properties
  - Product constraints (C_{PROD})
    - **Currently available products** in form of constraints

- Example,

C_R	{usage=large-print → **max-price > 200**} *customer* property *customer* property
C_F	{usage=large-print → **mpix > 5.0**} *customer* property *product* property
C_{PROD}	{ (id=p1∧ price=148∧ mpix=8.0 ∧ opt-zoom=4 ∧ lcd-size=2.5 ∧ Movies=no ∧ sound=no ∧ waterproof=yes) ∨…∨ (id=p8∧ price=278∧ mpix=9.1 ∧ opt-zoom=10 ∧ lcd-size=3.0 ∧ Movies=yes ∧ sound=yes ∧ waterproof=yes) }

The Customer Requirements (REQ)

- Encoded as unary constraints over the variables in V_c and V_{PROD}

- Example,

REQ	{ max-price=300, usage=large-print, photography=sports }

Asst. Prof. Dr. Rachsuda Setthawong

Defining Knowledge Based Recommendation Problem as a CSP Problem

- **Given:** CSP ($V = V_c \cup V_{PROD}$, D, $C = C_R \cup C_F \cup C_{PROD} \cup REQ$)
- **Find:** A consistent recommendation (RES) wrt CSP
- Example,

RES	{max-price=300, Usage=large-print, photography =sports, id=p8, price=278, mpix=9.1, opt-zoom=10, lcd-size=3.0, Movies=yes, sound=yes, waterproof=yes}	REQ C_{PROD}

Asst. Prof. Dr. Rachsuda Setthawong

A Simple Approach – **Conjunctive Queries**

- View the item selection (σ) problem as a data filtering task.
  - Construct a conjunctive query (a database query with a set of selection criteria connected *conjunctively*.)
  - E.g., **REQ {*usage=large-print, waterproof=yes*}**
    - Apply also constraints (C_R, C_F) if provided:

{usage=large-print → **max-price > 200**}

{usage=large-print → **mpix > 5.0**}

$\sigma_{[\text{mpix} > 5.0,\ \text{price}>200,\ \text{waterproof}=\text{'yes'}]}(P) = \{p8\}$

Id	Price (€)	M-pix	Opt-zoom	LCD-size	Movies	Sound	Waterproof
p1	148	8	4x	2.5	no	no	yes
p2	182	8	5x	2.7	yes	yes	no
p3	189	8	10x	2.5	yes	yes	no
p4	196	10	12x	2.7	yes	no	yes
p5	151	7.1	3x	3.0	yes	yes	no
p6	199	9	3x	3.0	yes	yes	no
p7	259	10	3x	3.0	yes	yes	no
p8	278	9.1	10x	3.0	yes	yes	yes

Asst. Prof. Dr. Rachsuda Setthawong

Ranking Recommended Items

- What if multiple items are retrieved wrt potential user's requirements?
  - E.g., REQ {max-price<200}
    - How to rank them?
      - using similarity measure

Id	Price (€)	Mpix	Opt-zoom	LCD-size	Movies	Sound	Waterproof
p1	148	8	4x	2.5	no	no	yes
p2	182	8	5x	2.7	yes	yes	no
p3	189	8	10x	2.5	yes	yes	no
p4	196	10	12x	2.7	yes	no	yes
P5	151	7.1	3x	3.0	yes	yes	no
p6	199	9	3x	3.0	yes	yes	no
p7	259	10	3x	3.0	yes	yes	no
p8	278	9.1	10x	3.0	yes	yes	yes

Case-based Recommendation Approach

- Similarity measure used to retrieve and rank items wrt user's requirements:

$$similarity(p, REQ) = \frac{\sum_{r \in REQ} w_r * sim(p, r)}{\sum_{r \in REQ} w_r}$$

- where,
  - p : a product (item)
  - r : the requirement (r ∈ REQ)
  - sim(p, r) : the similarity between an *item attribute value* $\phi_r(p)$ to *the customer requirement* r ∈ REQ
  - w_r : the important *weight for requirement* r

sim(p, r)

Note: user's requirement may be represented as value's range.

- More-is-better (MIB) properties

$$sim(p,r) = \frac{\phi_r(p) - \min(r)}{\max(r) - \min(r)}$$

- Less-is-better (LIB) properties

$$sim(p,r) = \frac{\max(r) - \phi_r(p)}{\max(r) - \min(r)}$$

- Alternative similarity measure

$$sim(p,r) = 1 - \frac{|\phi_r(p) - r|}{\max(r) - \min(r)}$$

Certain attr. value from the user

p : a product (item)

r : the requirement (attribute value) (r ∈ REQ)

$\phi_r(p)$: an *item attribute value* for $r \in REQ$

Attribute Examples for sim(p, r)

- More-is-better (MIB) properties

$$sim(p,r) = \frac{\phi_r(p) - \min(r)}{\max(r) - \min(r)}$$

Mega-pixel of a digital camera

- Less-is-better (LIB) properties

$$sim(p,r) = \frac{\max(r) - \phi_r(p)}{\max(r) - \min(r)}$$

Price of an item

- Alternative similarity measure

$$sim(p,r) = 1 - \frac{|\phi_r(p) - r|}{\max(r) - \min(r)}$$

Released year of a movie

An Example of Cases and Similarities

- REQ = {mpix≥9, 150≥price≤300}
- Calculate
  - $sim_{MIB}(p4, r_{mpix})$ = (10-9)/(10-9) = 1
  - $sim_{LIB}(p4, r_{price})$ = (300-196)/(300-150) = 0.693
  - Suppose that *mpix* and *price have* equal weight, similarity(p4, REQ)

$$= \frac{(w_{mpix} * sim(p4, r_{mpix}) + (wpric_e * sim(p4, rprice)))}{w_{mpix} + w_{price}}$$

= [(0.5 * 1) + (0.5 * 0.693)] / 1 = 0.8465

  - similarity(p7, REQ) = ?
  - Rank the suggested items?

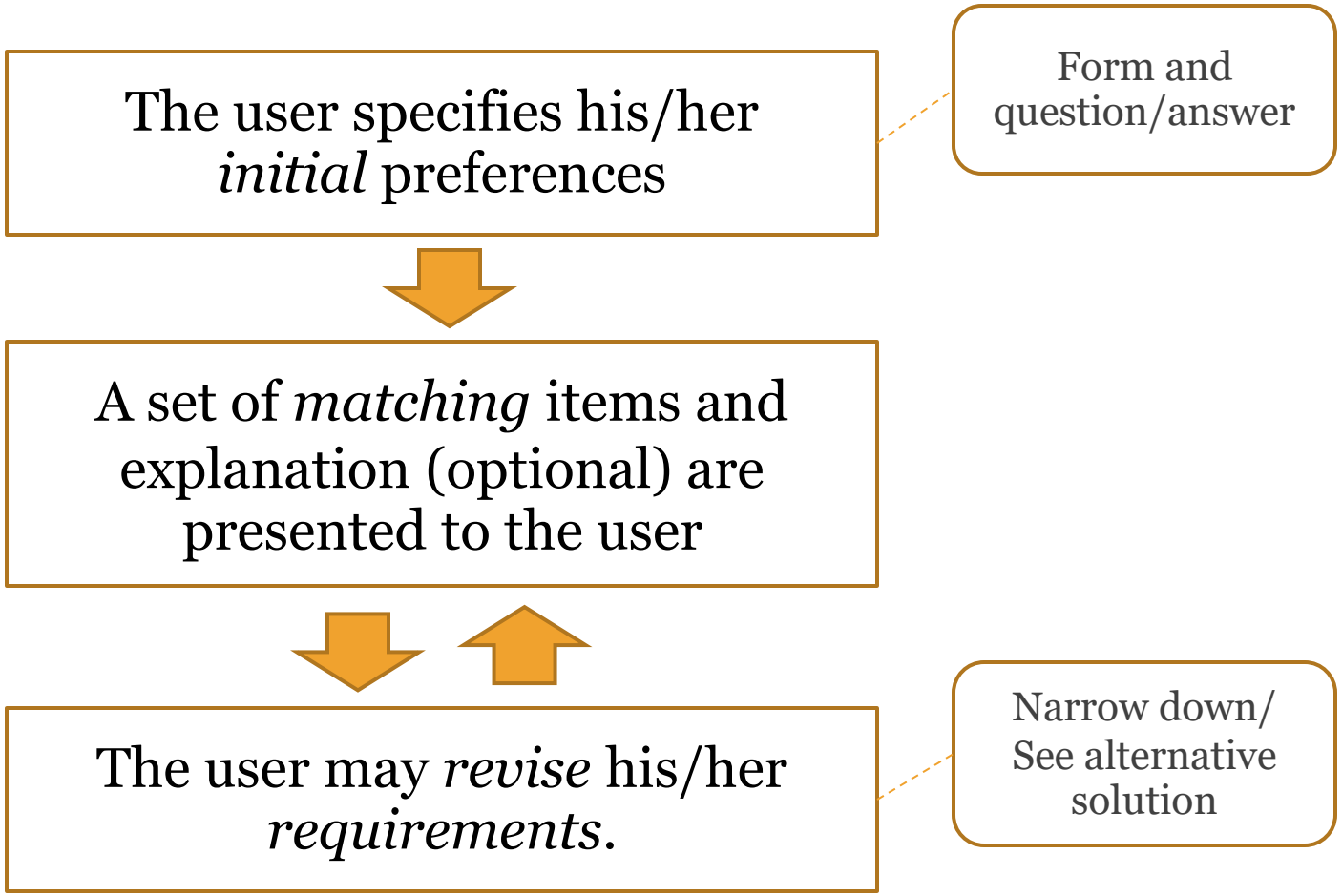
$$similarity(p, REQ) = \frac{\sum_{r \in REQ} w_r * sim(p, r)}{\sum_{r \in REQ} w_r}$$

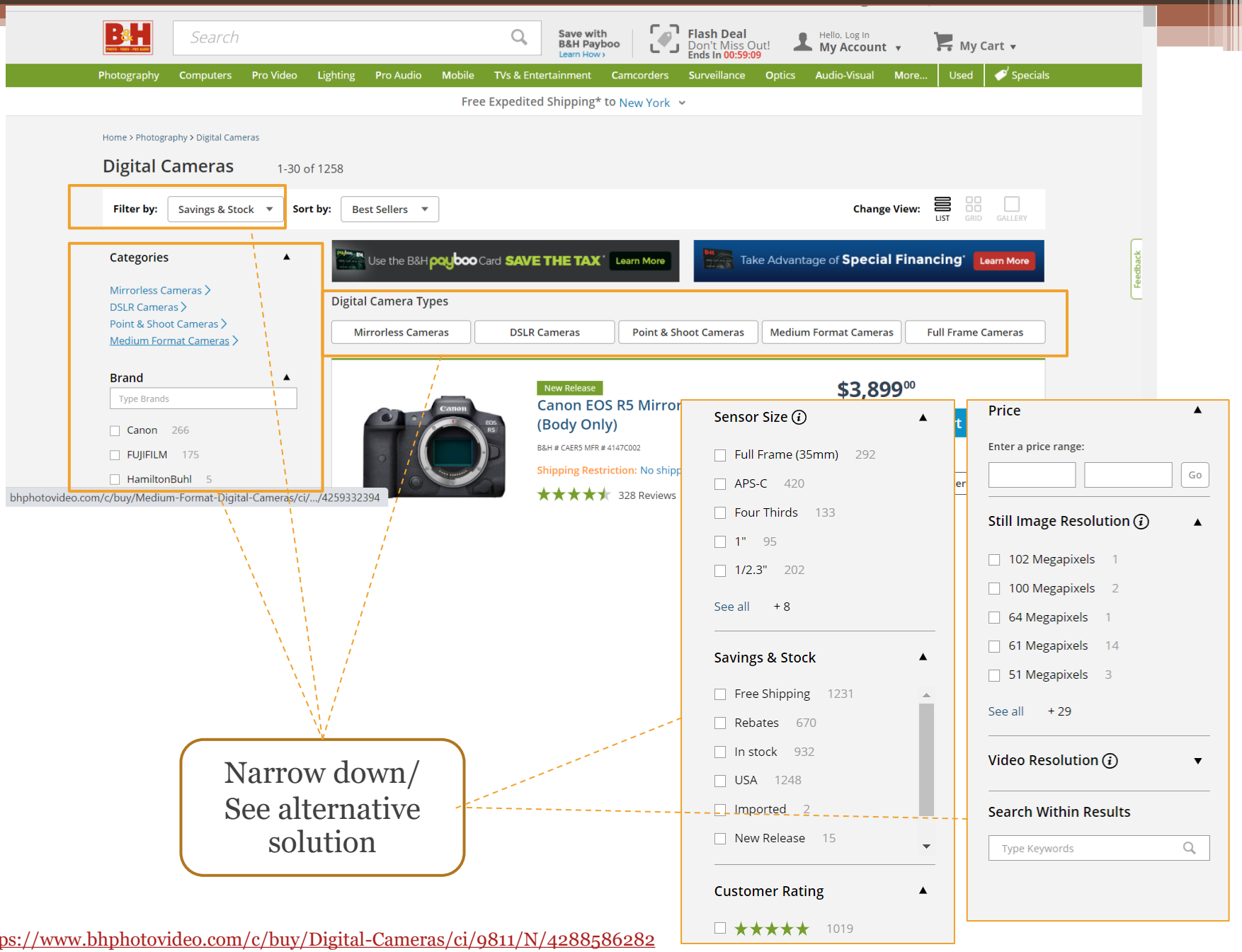
Id	Price (€)	Mpix	Opt-zoom	LCD-size	Movies	Sound	Waterproof
p4	196	10	12x	2.7	yes	no	yes
p7	259	10	3x	3.0	yes	yes	no

MIB: $sim(p, r) = \frac{\phi_r(p) - \min(r)}{\max(r) - \min(r)}$

LIB: $sim(p, r) = \frac{\max(r) - \phi_r(p)}{\max(r) - \min(r)}$

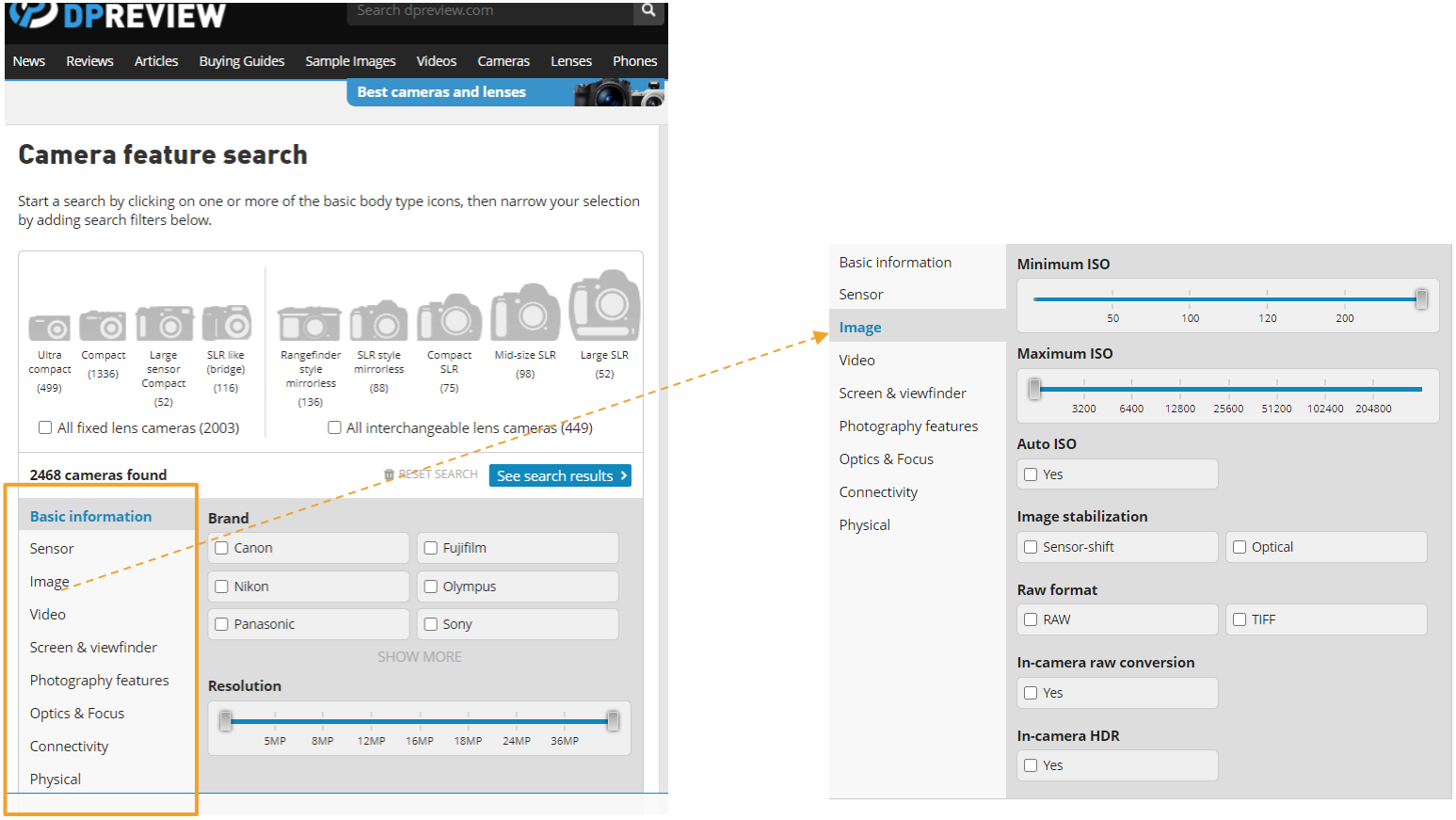
Interacting with Constraint-based Recommenders (Steps)





Narrow down/
See alternative
solution

Ref: https://www.bhphotovideo.com/c/buy/Digital-Cameras/ci/9811/N/4288586282



Ref: https://www.dpreview.com/products/search/cameras#!

Using Defaults During Recommendation Process

- For naïve user who doesn't know about features of items, using defaults will give him/her a reasonable suggestion

- Side effect: Ethical issue since it may lead the user to choose certain options. (e.g., select items with over specification)

Ways to Generate Defaults

- Static defaults
  - One default per customer property
  - E.g., default(usage) = large-print
- Dependent defaults
  - One default per different combinations of potential customer requirements
  - E.g., default(max-price $_{(usage=small-print)}$) = 300
- Derived defaults
  - Exploit existing *interaction logs* for the automated derivation of default values

Asst. Prof. Dr. Rachsuda Setthawong

An Example of Derived Defaults

Customer Interaction Log

customer (user)	price	opt-zoom	lcd-size
cu_1	400	10x	3.0
cu_2	300	10x	3.0
cu_3	150	4x	2.5
cu_4	200	5x	2.7
cu_5	200	5x	2.7

price=400

$opt\text{-}zoom_{default}$=?

A new user

Alternatives for Derived Defaults: 1-Nearest neighbor

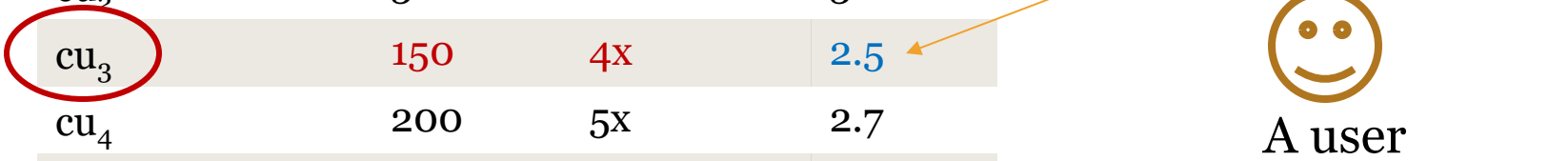
- Most similar entry in the *interactive log* to the user's REQ

$$similarity(p, REQ) = \frac{\sum_{r \in REQ} w_r * sim(p, r)}{\sum_{r \in REQ} w_r}$$

- E.g., REQ = {r1: price=150, r2: opt-zoom=4x};

Lcd-size$_{default}$=?

customer (user)	price	opt-zoom	lcd-size
cu_1	400	10x	3.0
cu_2	300	10x	3.0
cu_3	150	4x	2.5
cu_4	200	5x	2.7
cu_5	200	5x	2.7



Asst. Prof. Dr. Rachsuda Setthawong

Alternatives for Derived Defaults: Weighted Majority Voter

- The default value is based on the **majority voting** of neighbor items (k-NN).

$$similarity(p, REQ) = \frac{\sum_{r \in REQ} w_r * sim(p, r)}{\sum_{r \in REQ} w_r}$$

- E.g., REQ = {r1:price=400};
- $\text{opt-zoom}_{\text{default}}$ = ?

customer (user)	price	opt-zoom	lcd-size
cu_1	400	**10x**	3.0
cu_2	300	**10x**	3.0
cu_3	150	4x	2.5
cu_4	200	5x	2.7
cu_5	200	5x	2.7

$\text{opt-zoom}_{\text{default}}$=?

A user

For n=3,

3-NN(REQ)={cu_1, cu_2, cu_4}
Majority Voting = Max(10x, 10x, 5x)
$\text{opt-zoom}_{\text{default}}$=10x

Asst. Prof. Dr. Rachsuda Setthawong

A Major Problem with Using 1-NN and Weighted Majority Voters for Derived Defaults

- CANNOT guarantee that the result is a *non-empty* set.

customer (user)	price	opt-zoom	lcd-size
cu_1	400	10x	3.0
cu_2	300	10x	3.0
cu_3	150	4x	2.5
cu_4	200	5x	2.7
cu_5	200	5x	2.7

REQ = {r1:opt-zoom=3x};
$lcd\text{-}size_{default}$ = ?

For n=3,
3-NN(REQ)={cu_3, cu_4, cu_5}

$lcd\text{-}size_{default}$=2.7

Id	Price (€)	Mpix	Opt-zoom	LCD-size	Movies	Sound	Waterproof
p1	148	8	4x	2.5	no	no	yes
p2	182	8	5x	**2.7**	yes	yes	no
p3	189	8	10x	2.5	yes	yes	no
p4	196	10	12x	**2.7**	yes	no	yes
p5	151	7.1	**3x**	3.0	yes	yes	no
p6	199	9	**3x**	3.0	yes	yes	no
p7	259	10	**3x**	3.0	yes	yes	no
p8	278	9.1	10x	3.0	yes	yes	yes

$\sigma_{[opt\text{-}zoom=3x,\ lcd\text{-}size=2.7]}(P) = \varnothing$

Soln. slide 35

Asst. Prof. Dr. Rachsuda Setthawong

Selecting the Next Questions
(Propose *Default* for Next Property) – Approach 1

- E.g., what is the **first** recommended PROPERTY interested by the user?
- Pre-condition: user interaction log is available.
- Approach 1: using the principle of frequent usage (popularity)

$$popularity(attribute, pos) = \frac{\#selections(attr, pos)}{\#sessions}$$

Session-id	pos:1	pos:2	pos:3	pos:4	pos:5	pos:6	…
1	**price**	opt-zoom	mpix	movies	lcd-size	sound	
2	**price**	opt-zoom	mpix	movies	lcd-size	-	
3	**price**	Mpix	opt-zoom	lcd-size	movies	sound	
4	**mpix**	price	opt-zoom	lcd-size	movies	-	
5	**mpix**	price	lcd-size	opt-zoom	movies	sound	

popularity(price, pos:1)=3/5 =0.6

popularity(mpix, pos:1)=2/5 =0.4

Asst. Prof. Dr. Rachsuda Setthawong

Selecting the Next Questions (Propose *Default* for Next Property) – Approach 2

- E.g., *Given current REQ* contains properties {price, opt-zoom},

  what is the **next** recommended PROPERTY interested by the user?

- Pre-condition: user interaction log is available.
- Approach 2: using the **weighted majority voters**

Session-id	pos:1	pos:2	pos:3	pos:4	pos:5	pos:6	…
1	price	opt-zoom	**mpix**	movies	lcd-size	sound	
2	price	opt-zoom	**mpix**	movies	lcd-size	-	
3	price	Mpix	opt-zoom	lcd-size	movies	sound	
4	mpix	price	opt-zoom	lcd-size	movies	-	
5	mpix	price	lcd-size	opt-zoom	movies	sound	

Previous selection of the current user = {price, opt-zoom}

2-NN={1,2}

Next property (*pos:3*) =*mpix*

Dealing with Unsatisfiable Requirements and Empty Result Sets - 1/2

- E.g., Given
  - REQ = {
    - r1: price <= 150,
    - r2: opt-zoom = 5x,
    - r3: sound = yes,
    - r4: waterproof = yes}
  - P = {p_1, p_2, p_3, p_4, p_5, p_6, p_7, p_8}

Id	Price (€)	Mpix	Opt-zoom	LCD-size	Movies	Sound	Waterproof
p1	148	8	4x	2.5	no	no	yes
p2	182	8	5x	2.7	yes	yes	no
p3	189	8	10x	2.5	yes	yes	no
p4	196	10	12x	2.7	yes	no	yes
p5	151	7.1	3x	3.0	yes	yes	no
p6	199	9	3x	3.0	yes	yes	no
p7	259	10	3x	3.0	yes	yes	no
p8	278	9.1	10x	3.0	yes	yes	yes

- IF: $\sigma_{[\text{price} <= 150,\ \text{opt-zoom} = 5x,\ \text{sound} = \text{yes},\ \text{waterproof} = \text{yes}]}(P) = \varnothing$

- Solution: *Relax constraints until* the recommended product is met.

Dealing with Unsatisfiable Requirements and Empty Result Sets – 2/2

- **Assumption**: user's requirements V_C are *directly related* to item properties V_{PROD}.

- Basic idea: identifying and resolving requirements-immanent conflicts induced by the set of product P.
  - Step 1: provide the user *a minimal set of requirements*, once *changed*, leading to a solution.
  - Step 2: propose the adaptations of the initial requirements that guarantee a solution.

Step 1: Model-based Diagnosis (MBD) – 1/2

- The basis for automatically identification and repair of ***minimal*** *sets of* ***faulty*** *requirements*
  - The diagnosis task is to *identify* r ∈ REQ that may cause ***solution-not-found***.
  - **Output**: A diagnosis (d) is a minimal set of users requirements whose repair (adaptation) will allow the retrieval of a recommendation.
    - So that, once repair, at least one solution is found: $\sigma[_{\text{REQ-d}}](P) \neq \varnothing$

Step 1: Model-based Diagnosis (MBD) – 2/2

- Given
  - $P = \{p_1, p_2, \ldots, p_n\}$
  - $REQ = \{r_1, r_2, \ldots, r_m\}$
- Goal: Calculate a set of diagnoses $\Delta = \{d_1, d_2, \ldots, d_k\}$,

  where $\sigma[_{REQ - di}](P) \neq \varnothing \ \forall d_i \in \Delta$

- Finding diagnoses $d_i \in \Delta$ is based on the determination and resolution of conflict sets.

Conflict Set (CS)

- A subset $\{r_1, r_2, ..., r_l\} \subseteq$ REQ, s.t. $\sigma_{[CS]}(P)=\varnothing$

- Is *Minimal* **iff** there does NOT exist a CS′ with CS′ $\subset$ CS.

Revisiting the Problem:
Dealing with unsatisfiable requirements and empty result sets

- E.g., Given
  - REQ = {
    - r1: price <= 150,
    - r2: opt-zoom = 5x,
    - r3: sound = yes,
    - r4: waterproof = yes}
  - P = {p_1, p_2, p_3, p_4, p_5, p_6, p_7, p_8}

Id	Price (€)	Mpix	Opt-zoom	LCD-size	Movies	Sound	Waterproof
p1	148	8	4x	2.5	no	no	yes
p2	182	8	5x	2.7	yes	yes	no
p3	189	8	10x	2.5	yes	yes	no
p4	196	10	12x	2.7	yes	no	yes
p5	151	7.1	3x	3.0	yes	yes	no
p6	199	9	3x	3.0	yes	yes	no
p7	259	10	3x	3.0	yes	yes	no
p8	278	9.1	10x	3.0	yes	yes	yes

- IF: $\sigma_{[\text{price <= 150, opt-zoom = 5x, sound = yes, waterproof = yes}]}(P) = \varnothing$
- Solution: Finding 1.1) the Minimal Conflict Set (CS), then used it to find
  1.2) Diagnoses $d_i \in \Delta$ and propose them to the user
  (for *relaxing* user *requirements*)

Step 1.1: Finding the Minimal Conflict Set (CS)

- Given
  - REQ = {
    - r1: price <= 150,
    - r2: opt-zoom = 5x,
    - r3: sound = yes,
    - r4: waterproof = yes}

Id	Price (€)	Mpix	Opt-zoom	LCD-size	Movies	Sound	Waterproof
p1	148	8	4x	2.5	no	no	yes
p2	182	8	5x	2.7	yes	yes	no
p3	189	8	10x	2.5	yes	yes	no
p4	196	10	12x	2.7	yes	no	yes
p5	151	7.1	3x	3.0	yes	yes	no
p6	199	9	3x	3.0	yes	yes	no
p7	259	10	3x	3.0	yes	yes	no
p8	278	9.1	10x	3.0	yes	yes	yes

- $CS1 = \{r_1, r_2\}, \quad \sigma_{[CS1]}(P) = \varnothing$
- $CS2 = \{r_2, r_4\}, \quad \sigma_{[CS2]}(P) = \varnothing$
- $CS3 = \{r_1, r_3\}, \quad \sigma_{[CS3]}(P) = \varnothing$

QUICKXPLAIN (P, REQ)

- Find the conflict set (CS).

Input: trusted knowledge (items) P; Set of requirements REQ

Output: minimal conflict set CS

if $\sigma_{[REQ]}(P) \neq \emptyset$ or $REQ = \emptyset$ **then** return $\emptyset$

else return QX'(P, $\emptyset$, $\emptyset$, REQ);

Function QX'(P, B, Δ, REQ)

if $\Delta \neq \emptyset$ and $\sigma_{[B]}(P) = \emptyset$ **then** return $\emptyset$;

if $REQ = \{r\}$ **then** return $\{r\}$;

let $\{r_1, ..., r_n\} = REQ$;

let k be n/2;

$REQ_1 \leftarrow r_1, ..., r_k$ and $REQ_2 \leftarrow r_{k+1}, ..., r_n$;

$\Delta_2 \leftarrow$ QX'(P, $B \cup REQ_1$, REQ_1, REQ_2);

$\Delta_1 \leftarrow$ QX'(P, $B \cup \Delta_2$, Δ_2, REQ_2);

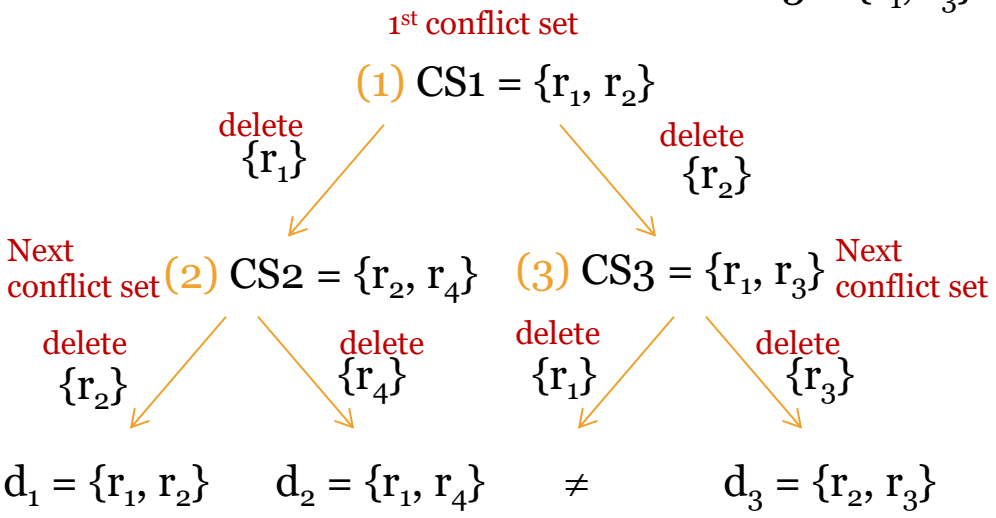
return $\Delta_1 \cup \Delta_2$;

Asst. Prof. Dr. Rachsuda Setthawong

Step 1.2: Calculating Diagnoses d_i

Conflict set:
CS1 = $\{r_1, r_2\}$,
CS2 = $\{r_2, r_4\}$,
CS3 = $\{r_1, r_3\}$

- Calculated by resolving conflicts in the given set of requirements.
  - By deleting one of the elements from CS.
  - Obtaining a corresponding diagnosis (an outcome)



- From the example,
  - $\Delta = \{ d_1:\{r_1,r_2\}, d_2:\{r_1,r_4\}, d_3:\{r_2,r_3\} \}$

REQ = {
r1: price <= 150,
r2: opt-zoom = 5x,
r3: sound = yes,
r4: waterproof = yes}

Id	Price (€)	Mpix	Opt-zoom	LCD-size	Movies	Sound	Waterproof
p1	148	8	4x	2.5	no	no	yes
p2	182	8	5x	2.7	yes	yes	no
p3	189	8	10x	2.5	yes	yes	no
p4	196	10	12x	2.7	yes	no	yes
p5	151	7.1	3x	3.0	yes	yes	no
p6	199	9	3x	3.0	yes	yes	no
p7	259	10	3x	3.0	yes	yes	no
p8	278	9.1	10x	3.0	yes	yes	yes

Step 2: Proposing Repairs for Unsatisfiable Requirements

- Derive alternative repair action by *querying the product* table P *with* $\Pi_{[attributes(d)]}\sigma_{[\mathbf{REQ\text{-}d}]}(P)$.
  - Example: *repair alternatives* for the requirements in

    REQ: {r1: price <= 150, r2: opt-zoom = 5x, r3: sound = yes, r4: waterproof = yes}

	r1	r2	r3	r4
repair	**price**	**opt-zoom**	**sound**	**waterproof**
repair1	278	10x	√	√
repair2	182	√	√	no
repair3	√	4x	no	√

Δ = { d_1:{r_1, r_2}, d_2:{r_1, r_4}, d_3:{r_2, r_3} }

Atfer repair	Recom-mended ID	Price (€)	Mpix	Opt-zoom	LCD -size	Movies	Sound	Waterproof
repair1	p8	278	9.1	10x	3.0	yes	yes	yes
repair2	p2	182	8	5x	2.7	yes	yes	no
repair3	p1	148	8	4x	2.5	no	no	yes

Relaxed constraints

√: matched constraints

Better Solution: MINRELAX(P, REQ)

- Find a complete set of diagnoses (integrating the steps 1.1 and 1.2)
  - NO *explicit conflict sets* extraction is required.

ALGORITHM: *MinRelax*
In: A query Q, a product catalog P
Out: Set of minimal relaxations *minRS* for Q

MinRS = $\emptyset$
forall $p_i \in P$ **do**
  PSX = Compute the product-specific relaxation $PSX(Q, p_i)$
  % Check relaxations that were already found
  SUB = $\{r \in MinRS \mid r \subset PSX\}$
  if $SUB \neq \emptyset$
    % Current relaxation is superset of existing relaxation
    continue with next p_i
  endif
  SUPER = $\{r \in MinRS \mid \text{PSX} \subset r\}$
  if $SUPER \neq \emptyset$
    % Remove supersets
    $MinRS = MinRS \setminus SUPER$
  endif
  % Store the new relaxation
  $MinRS = MinRS \cup \{PSX\}$
endfor
return MinRS

Fig. 4. Algorithm for determining all minimal relaxations.

Reference: https://web-ainf.aau.at/pub/jannach/files/Journal_AICOM09.pdf

Asst. Prof. Dr. Rachsuda Setthawong

Product-Specific-Relaxation (PSXs)'s Idea

ID	USB	Firewire	Price	Resolution	Make
p1	true	false	400	5 MP	Canon
p2	false	true	500	5 MP	Canon
p3	true	false	200	4 MP	Fuji
p4	false	true	400	5 MP	HP

Fig. 2. Product database of digital cameras.

Let the user's query consist of the following requirements (atoms) which results in a theoretical search space of $2^4 = 16$ combinations of atoms.

Q = { usb = true (Q1), firewire = true (Q2), price < 300 (Q3), resolution >= 5 MP (Q4)}

ID	p1	p2	p3	p4
Q1	1	0	1	0
Q2	0	1	0	1
Q3	0	0	1	0
Q4	1	1	0	1

Product-specific relaxation for p1

Fig. 3. Evaluating the subqueries individually.

The list of product-specific-relaxation (PSXs) = <{Q2,Q3},{Q1,Q3},{Q2,Q4},{Q1,Q3}>

The set of ***minimal*** relaxation is {{Q1,Q3},{Q2,Q3},{Q2,Q4}}

(Duplicated PSX)

Example - MINRELAX(P, REQ) 1/9

REQ: {
r1: price <= 150,
r2: opt-zoom = 5x,
r3: sound = yes,
r4: waterproof = yes}

Id	Price (€)	Opt-zoom	Sound	Waterproof	product-specific-relaxation (PSXs)
p1	148	4x	no	yes	{r2, r3}
p2	182	5x	yes	no	{r1, r4}
p3	189	10x	yes	no	{r1, r2, r4}
p4	196	12x	no	yes	{r1, r2, r3}
p5	151	3x	yes	no	{r1, r2, r4}
p6	199	3x	yes	no	{r1, r2, r4}
p7	259	3x	yes	no	{r1, r2, r4}
p8	278	10x	yes	yes	{r1, r2}

Asst. Prof. Dr. Rachsuda Setthawong

Example - MinRelax(P, REQ) 2/9

ALGORITHM: *MinRelax*
In: A query Q, a product catalog P
Out: Set of minimal relaxations *minRS* for Q

MinRS = $\emptyset$
forall $p_i \in P$ **do**
  PSX = Compute the product-specific relaxation $PSX(Q, p_i)$
  % Check relaxations that were already found
  SUB = $\{r \in MinRS \mid r \subset PSX\}$
  if $SUB \neq \emptyset$
    % Current relaxation is superset of existing relaxation
    continue with next p_i
  endif
  SUPER = $\{r \in MinRS \mid \text{PSX} \subset r\}$
  if $SUPER \neq \emptyset$
    % Remove supersets
    $MinRS = MinRS \setminus SUPER$
  endif
  % Store the new relaxation
  $MinRS = MinRS \cup \{PSX\}$
endfor
return MinRS

Fig. 4. Algorithm for determining all minimal relaxations.

Id	product-specific-relaxation (PSXs)
p1	{r2, r3}
p2	{r1, r4}
p3	{r1, r2, r4}
p4	{r1, r2, r3}
p5	{r1, r2, r4}
p6	{r1, r2, r4}
p7	{r1, r2, r4}
p8	{r1, r2}

PSX(p1) = {r2, r3}
SUB = ∅, SUPER = ∅
MinRS = {{r2, r3}}

Asst. Prof. Dr. Rachsuda Setthawong

Example - MinRelax(P, REQ) 3/9

ALGORITHM: *MinRelax*
In: A query Q, a product catalog P
Out: Set of minimal relaxations *minRS* for Q

MinRS = $\emptyset$
forall $p_i \in P$ **do**
  PSX = Compute the product-specific relaxation $PSX(Q, p_i)$
  % Check relaxations that were already found
  SUB = $\{r \in MinRS \mid r \subset PSX\}$
  if $SUB \neq \emptyset$
    % Current relaxation is superset of existing relaxation
    continue with next p_i
  endif
  SUPER = $\{r \in MinRS \mid \text{PSX} \subset r\}$
  if $SUPER \neq \emptyset$
    % Remove supersets
    $MinRS = MinRS \setminus SUPER$
  endif
  % Store the new relaxation
  $MinRS = MinRS \cup \{PSX\}$
endfor
return MinRS

Fig. 4. Algorithm for determining all minimal relaxations.

Id	product-specific-relaxation (PSXs)
p1	{r2, r3}
p2	{r1, r4}
p3	{r1, r2, r4}
p4	{r1, r2, r3}
p5	{r1, r2, r4}
p6	{r1, r2, r4}
p7	{r1, r2, r4}
p8	{r1, r2}

PSX(p2) = {r1, r4}
SUB = ∅, SUPER = ∅
MinRS = { {r2, r3}, {r1, r4} }

Asst. Prof. Dr. Rachsuda Setthawong

Example - MinRelax(P, REQ) 4/9

ALGORITHM: *MinRelax*
In: A query Q, a product catalog P
Out: Set of minimal relaxations *minRS* for Q

MinRS = $\emptyset$
forall $p_i \in P$ **do**
  PSX = Compute the product-specific relaxation $PSX(Q, p_i)$
  % Check relaxations that were already found
  SUB = $\{r \in MinRS \mid r \subset PSX\}$
  if $SUB \neq \emptyset$
    % Current relaxation is superset of existing relaxation
    continue with next p_i
  endif
  SUPER = $\{r \in MinRS \mid \text{PSX} \subset r\}$
  if $SUPER \neq \emptyset$
    % Remove supersets
    $MinRS = MinRS \setminus SUPER$
  endif
  % Store the new relaxation
  $MinRS = MinRS \cup \{PSX\}$
endfor
return MinRS

Fig. 4. Algorithm for determining all minimal relaxations.

Id	product-specific-relaxation (PSXs)
p1	{r2, r3}
p2	{r1, r4}
p3	{r1, r2, r4}
p4	{r1, r2, r3}
p5	{r1, r2, r4}
p6	{r1, r2, r4}
p7	{r1, r2, r4}
p8	{r1, r2}

PSX(p3) = {r1, r2, r4}
SUB = {r1, r4}
MinRS = { {r2, r3}, {r1, r4} }

Asst. Prof. Dr. Rachsuda Setthawong

Example - MinRelax(P, REQ) 5/9

ALGORITHM: *MinRelax*
In: A query Q, a product catalog P
Out: Set of minimal relaxations *minRS* for Q

MinRS = $\emptyset$
forall $p_i \in P$ **do**
  PSX = Compute the product-specific relaxation $PSX(Q, p_i)$
  % Check relaxations that were already found
  SUB = $\{r \in MinRS \mid r \subset PSX\}$
  if $SUB \neq \emptyset$
    % Current relaxation is superset of existing relaxation
    continue with next p_i
  endif
  SUPER = $\{r \in MinRS \mid \text{PSX} \subset r\}$
  if $SUPER \neq \emptyset$
    % Remove supersets
    $MinRS = MinRS \setminus SUPER$
  endif
  % Store the new relaxation
  $MinRS = MinRS \cup \{PSX\}$
endfor
return MinRS

Fig. 4. Algorithm for determining all minimal relaxations.

Id	product-specific-relaxation (PSXs)
p1	{r2, r3}
p2	{r1, r4}
p3	{r1, r2, r4}
p4	{r1, r2, r3}
p5	{r1, r2, r4}
p6	{r1, r2, r4}
p7	{r1, r2, r4}
p8	{r1, r2}

PSX(p4) = {r1, r2, r3}
SUB = {r2, r3}
MinRS = { {r2, r3}, {r1, r4} }

Asst. Prof. Dr. Rachsuda Setthawong

Example - MinRelax(P, REQ) 6/9

ALGORITHM: *MinRelax*
In: A query Q, a product catalog P
Out: Set of minimal relaxations *minRS* for Q

MinRS = $\emptyset$
forall $p_i \in P$ **do**
  PSX = Compute the product-specific relaxation $PSX(Q, p_i)$
  % Check relaxations that were already found
  SUB = $\{r \in MinRS \mid r \subset PSX\}$
  if $SUB \neq \emptyset$
    % Current relaxation is superset of existing relaxation
    continue with next p_i
  endif
  SUPER = $\{r \in MinRS \mid \text{PSX} \subset r\}$
  if $SUPER \neq \emptyset$
    % Remove supersets
    $MinRS = MinRS \setminus SUPER$
  endif
  % Store the new relaxation
  $MinRS = MinRS \cup \{PSX\}$
endfor
return MinRS

Fig. 4. Algorithm for determining all minimal relaxations.

Id	product-specific-relaxation (PSXs)
p1	{r2, r3}
p2	{r1, r4}
p3	{r1, r2, r4}
p4	{r1, r2, r3}
p5	{r1, r2, r4}
p6	{r1, r2, r4}
p7	{r1, r2, r4}
p8	{r1, r2}

PSX(p5) = {r1, r2, r4}
SUB = {r1, r4}
MinRS = { {r2, r3}, {r1, r4} }

Asst. Prof. Dr. Rachsuda Setthawong

Example - MinRelax(P, REQ) 7/9

ALGORITHM: *MinRelax*
In: A query Q, a product catalog P
Out: Set of minimal relaxations *minRS* for Q

MinRS = $\emptyset$
forall $p_i \in P$ **do**
  PSX = Compute the product-specific relaxation $PSX(Q, p_i)$
  % Check relaxations that were already found
  SUB = $\{r \in MinRS \mid r \subset PSX\}$
  if $SUB \neq \emptyset$
    % Current relaxation is superset of existing relaxation
    continue with next p_i
  endif
  SUPER = $\{r \in MinRS \mid \text{PSX} \subset r\}$
  if $SUPER \neq \emptyset$
    % Remove supersets
    $MinRS = MinRS \setminus SUPER$
  endif
  % Store the new relaxation
  $MinRS = MinRS \cup \{PSX\}$
endfor
return MinRS

Fig. 4. Algorithm for determining all minimal relaxations.

Id	product-specific-relaxation (PSXs)
p1	{r2, r3}
p2	{r1, r4}
p3	{r1, r2, r4}
p4	{r1, r2, r3}
p5	{r1, r2, r4}
p6	{r1, r2, r4}
p7	{r1, r2, r4}
p8	{r1, r2}

PSX(p6) = {r1, r2, r4}
SUB = {r1, r4}
MinRS = { {r2, r3}, {r1, r4} }

Asst. Prof. Dr. Rachsuda Setthawong

Example - MinRelax(P, REQ) 8/9

ALGORITHM: *MinRelax*
In: A query Q, a product catalog P
Out: Set of minimal relaxations *minRS* for Q

MinRS = $\emptyset$
forall $p_i \in P$ **do**
  PSX = Compute the product-specific relaxation $PSX(Q, p_i)$
  % Check relaxations that were already found
  SUB = $\{r \in MinRS \mid r \subset PSX\}$
  if $SUB \neq \emptyset$
    % Current relaxation is superset of existing relaxation
    continue with next p_i
  endif
  SUPER = $\{r \in MinRS \mid \text{PSX} \subset r\}$
  if $SUPER \neq \emptyset$
    % Remove supersets
    $MinRS = MinRS \setminus SUPER$
  endif
  % Store the new relaxation
  $MinRS = MinRS \cup \{PSX\}$
endfor
return MinRS

Fig. 4. Algorithm for determining all minimal relaxations.

Id	product-specific-relaxation (PSXs)
p1	{r2, r3}
p2	{r1, r4}
p3	{r1, r2, r4}
p4	{r1, r2, r3}
p5	{r1, r2, r4}
p6	{r1, r2, r4}
p7	{r1, r2, r4}
p8	{r1, r2}

PSX(p7) = {r1, r2, r4}
SUB = {r1, r4}
MinRS = { {r2, r3}, {r1, r4} }

Asst. Prof. Dr. Rachsuda Setthawong

Example - MinRelax(P, REQ) 9/9

ALGORITHM: *MinRelax*
In: A query Q, a product catalog P
Out: Set of minimal relaxations *minRS* for Q

MinRS = ∅
forall $p_i \in P$ **do**
  PSX = Compute the product-specific relaxation $PSX(Q, p_i)$
  % Check relaxations that were already found
  SUB = $\{r \in MinRS \mid r \subset PSX\}$
  if $SUB \neq \emptyset$
    % Current relaxation is superset of existing relaxation
    continue with next p_i
  endif
  SUPER = $\{r \in MinRS \mid \text{PSX} \subset r\}$
  if $SUPER \neq \emptyset$
    % Remove supersets
    $MinRS = MinRS \setminus SUPER$
  endif
  % Store the new relaxation
  $MinRS = MinRS \cup \{PSX\}$
endfor
return MinRS

Fig. 4. Algorithm for determining all minimal relaxations.

Id	product-specific-relaxation (PSXs)
p1	{r2, r3}
p2	{r1, r4}
p3	{r1, r2, r4}
p4	{r1, r2, r3}
p5	{r1, r2, r4}
p6	{r1, r2, r4}
p7	{r1, r2, r4}
p8	{r1, r2}

PSX(p8) = {r1, r2}
SUB = ∅, SUPER = ∅
MinRS =
{ {r2, r3}, {r1, r4}, {r1, r2} }

Then, can further apply "Step 2: Proposing Repairs for Unsatisfiable Requirements" (slide no. 44)

Ranking the Items/Utility-based Recommendation – 1/3

- Rank recommendations before presenting them to the user.
- Based on the multi-attribute utility theory (MAUT),
  - Evaluates each item wrt its utility for the customer.
    - E.g.,

domain	Evaluation measures (dimensions)
Digital camera	Quality, economy
Financial services	Availability, risk, profit

Asst. Prof. Dr. Rachsuda Setthawong

Ranking the Items/Utility-based Recommendation - 2/3

$$utility(p) = \sum_{j=1}^{\#(dimensions)} interest(j) * contribution(p,j)$$

Customer-specific preferences (***interest***)

User defined

customer (user)	quality	economy
cu_1	80%	20%
cu_2	40%	60%

Scoring rules (for calculating ***contribution***)

Knowledge base

	value	quality	economy
Price	≤250 >250	5 10	10 5
mpix	≤8 >8	4 10	10 6
opt-zoom	≤9 >9	6 10	9 6
lcd-size	≤2.7 >2.7	6 9	10 5
movies	yes no	10 3	7 10
sound	yes no	10 7	8 10
waterproof	yes no	10 8	6 10

Example of Calculation *Contribution(p_1, j)*

Id	Price (€)	Mpix	Opt-zoom	LCD-size	Movies	Sound	Water proof
p1	148	8	4x	2.5	no	no	yes

	value	quality	economy
Price	**≤250** >250	**5** 10	**10** 5
mpix	**≤8** >8	**4** 10	**10** 6
opt-zoom	**≤9** >9	**6** 10	**9** 6
lcd-size	**≤2.7** >2.7	**6** 9	**10** 5
movies	yes **no**	10 **3**	7 **10**
sound	yes **no**	10 **7**	8 **10**
waterproof	**yes** no	**10** 8	**6** 10

contribution(p_1, quality)
= 5+4+6+6+3+7+10
= 41

contribution(p_1, economy)
= 10+10+9+10+10+10+6
= 65

Asst. Prof. Dr. Rachsuda Setthawong

Ranking the Items/Utility-based Recommendation - 3/3

$$utility(p) = \sum_{j=1}^{\#(dimensions)} interest(j) * contribution(p,j)$$

Rank of Recommended Products

Customer-specific preferences (***interest***)

customer (user)	quality	economy
cu_1	80%	20%
cu_2	40%	60%

From previous slide's calculation:
contribution(p_1, quality) = 41
contribution(p_1, economy) = 65

Then,
utility(p_1) for cu_1 = (0.8*41) + (0.2*65) = 45.8
utility(p_1) for cu_2 = (0.4*41) + (0.6*65) = 55.4

Product utilities for customer cu_1 and cu_2

product	contribution(p_i, quality)	contribution(p_i, economy)	utility(p_i) for cu_1	utility(p_i) for cu_2
p_1	41	65	45.8 [8]	55.4 [6]
p_2	49	64	52.0 [7]	58.0 [1]
p_3	53	61	54.6 [5]	57.8 [2]
p_4	58	55	57.4 [4]	56.2 [4]
p_5	53	60	54.4 [6]	57.2 [3]
p_6	58	55	57.4 [3]	56.2 [5]
p_7	63	50	60.4 [2]	55.2 [7]
p_8	69	43	63.8 [1]	53.4 [8]

Approaches for Determining a *Customer's Degree of Interest*

- User-defined preferences
  - The user *explicitly specifies* his/her preferences beforehand.
- Utility-based preferences
  - *Apply the scoring rules* in the table.
  - E.g., REQ = {

    r_1: price <=200,

    r_2: mpix=8.0,

    r_3: opt-zoom=10x,

    r_4: lcd-size<=2.7 }
    - Interest(quality) = 5+4+10+6 = 25
    - Interest(economy) = 10+10+6+10 = 36
    - *Relative* Interest(quality) = 25/(25+36) = 0.41
    - *Relative* Interest(economy) = 36/(25+36) = 0.59

Example of *scoring rules*

	value	quality	economy
Price	**≤250**	**5**	**10**
	>250	10	5
mpix	**≤8**	**4**	**10**
	>8	10	6
opt-zoom	≤9	6	9
	>9	**10**	**6**
lcd-size	**≤2.7**	**6**	**10**
	>2.7	9	5
movies	yes	10	7
	no	3	10
sound	yes	10	8
	no	7	10
waterproof	yes	10	6
	no	8	10