Conversational Recommender Systems

Francesco Ricci
eCommerce and Tourism Research Laboratory
Automated Reasoning Systems Division
ITC-irst
Trento – Italy
ricci@itc.it
http://ectrl.itc.it

Content

- Typically the advise provided by a clerk to a customer is multi-stage
- Conversations are based on user requests (queries)
- Queries may fail – repair methods
- Query may return too many options – refining/tightening
- Information search and discovery
- Preference-based feedback
- Critiques: non-incremental/incremental, non-adaptive/adaptive
- Empirical Evaluations of Critiques
Customer-Clerk Conversation

- **Customer:** I'd like to find a 3 or 4 stars hotel with the price up to 100
- **Clerk:** There are too many hotels with these characteristics. Would you like to have facilities like car parking or air conditioning?
- **Customer:** Yes, I need car parking and restaurant
- **Clerk:** I'm sorry but now there is no match! BUT: there are 40 hotels in the desired category with parking, restaurant, but you have to consider a price up to 110, OR there is one hotel with your desired category and price which has a restaurant but no parking
- **Customer:** Let me see this cheap option [she sees it]
- **Customer:** Let me now see the expensive ones
- **Clerk:** Would you like to have internet connection in your room?
- **Customer:** Yes [and she gets 10 suggestions]

Single-Shot vs Conversational

- **Single-shot**
  - Queries within even a single session are treated independently
- **Conversational**
  - The recommender supports a recommendation dialog of multiple conversational moves [Linden et al., 1997]
    [Thomson et al., 2004]
  - In recognition that
    - customers are rarely able to specify all their requirements up-front
    - customers are rarely satisfied with the initial recommendations
Session Dependency

- Pure content-based and collaborative filtering techniques are context independent
  - The recommendation depends uniquely on the history of previous “purchases”
- A conversational recommender system must take into account the requirements specified during the conversation
  - These requirements (plus some other implicit variables) enter into the definition of the CONTEXT of the recommendation.

Session-Independent Recommendations

- User Models, e.g.
  - demographics
  - purchase history
  - browsing history (prev. sessions)
  - ratings
  - content profiles

- Product database
- registration information, purchases, click stream, product consumption evaluation (ratings)
- content profile updates
Session-Dependent Recommendations

queries (e.g. keywords; answers to questions; values entered into forms)
recommendation evaluation (feedback)
click stream
context (time and place)

Failing Queries in Recommender Systems

- Queries in a recommender systems, allow a user to express his/her needs, preferences and constraints about the desired item
  - I’d like an accommodation close to the golf course, with sauna, solarium and price less than 110 Euro
- BUT sometime... a query may return: no result
Filter-Based Recommenders

- Are recommenders based on query processing that search for exact matching
- They always bring the risk of “no results found”
- A system that behaves in this way seem like a stubborn clerk
- This system does not leave much options to the user (stonewalling).

Google

- Queries almost never fail!
- Implicit Relaxation
Dealing with failing queries

- Dealing with failing queries means either
  - to avoid the problem (google example) using a query model that never fails
  - Supporting the user to escape from this dead end
- Some approaches to deal with the problem
  - If the query fails show some popular options
  - Similarity based retrieval
  - Query relaxation: cooperate with the user to explain the cause of failure and suggest repair actions

Anticipating the Failure
Solution 1: Similarity Based retrieval

- Query is considered as a pattern, i.e., a partially specified good example
- A similarity metric over the product data type must be defined. That must tell you:
  - How similar is a 3-stars and a 4-stars hotel?
  - Is it more similar a 3-stars hotel to a 4-stars one or a hotel with parking to one without parking?
- Products are sorted according to their similarities with the query pattern.

<table>
<thead>
<tr>
<th>Case</th>
<th>Location code</th>
<th>Bedrooms</th>
<th>Rec'p rooms</th>
<th>Type</th>
<th>floors</th>
<th>Condition</th>
<th>Price</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>8</td>
<td>2</td>
<td>1</td>
<td>terraced</td>
<td>1</td>
<td>poor</td>
<td>20,500</td>
</tr>
<tr>
<td>2</td>
<td>5</td>
<td>2</td>
<td>1</td>
<td>terraced</td>
<td>1</td>
<td>fair</td>
<td>25,000</td>
</tr>
<tr>
<td>3</td>
<td>5</td>
<td>1</td>
<td>2</td>
<td>semi</td>
<td>2</td>
<td>good</td>
<td>48,000</td>
</tr>
<tr>
<td>4</td>
<td>5</td>
<td>1</td>
<td>2</td>
<td>terraced</td>
<td>2</td>
<td>good</td>
<td>41,000</td>
</tr>
</tbody>
</table>

Probe case = query

<table>
<thead>
<tr>
<th>Case</th>
<th>Location code</th>
<th>Bedrooms</th>
<th>Rec'p rooms</th>
<th>Type</th>
<th>floors</th>
<th>Condition</th>
<th>Price</th>
</tr>
</thead>
<tbody>
<tr>
<td>5</td>
<td>8</td>
<td>2</td>
<td>2</td>
<td>semi</td>
<td>2</td>
<td>fair</td>
<td>20,500</td>
</tr>
</tbody>
</table>

= matching attribute
Property Example

<table>
<thead>
<tr>
<th>Case</th>
<th>Location code</th>
<th>Bedrooms</th>
<th>Recip rooms</th>
<th>Type</th>
<th>floors</th>
<th>Condition</th>
<th>Price</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>6</td>
<td>2</td>
<td>1</td>
<td>terraced</td>
<td>1</td>
<td>poor</td>
<td>20,500</td>
</tr>
<tr>
<td>2</td>
<td>8</td>
<td>2</td>
<td>1</td>
<td>terraced</td>
<td>1</td>
<td>fair</td>
<td>25,000</td>
</tr>
<tr>
<td>3</td>
<td>5</td>
<td>1</td>
<td>2</td>
<td>semi</td>
<td>2</td>
<td>good</td>
<td>48,000</td>
</tr>
<tr>
<td>4</td>
<td>5</td>
<td>1</td>
<td>2</td>
<td>terraced</td>
<td>2</td>
<td>good</td>
<td>41,000</td>
</tr>
</tbody>
</table>

- Case 4 is surely a bad option, since case 3 is more similar to the query.
- Case 1 and case 2 differ in the condition and price. Hence if price is more important than condition case 1 should be preferred to case 2.
- Case 2 and 3 match the query for completely different features, how to decide if case 2 is better than case 3 or vice versa?

Similarity-Based Relaxation Drawbacks

- Similarity metric has to defined for all data types
- Similarity must be tuned, e.g., weighting feature importance (according to the domain knowledge or user preferences)
- It implicitly relaxes query constraints when no item matches completely the query
- Do not give any summary information about how to “repair” the query (it is not cooperative)
- Hence the user can miss the relaxation that he would like the best.
Another Example

Ex: \( q = \{\text{prolog, language, comparison, survey, rating}\} \) fails to retrieve any record (web page)

<table>
<thead>
<tr>
<th>( q' )</th>
<th>url1</th>
<th>prolog</th>
<th>comparison</th>
<th>survey</th>
</tr>
</thead>
<tbody>
<tr>
<td>url2</td>
<td>language</td>
<td>comparison</td>
<td>survey</td>
<td>rating</td>
</tr>
<tr>
<td>url3</td>
<td>prolog</td>
<td>language</td>
<td>survey</td>
<td></td>
</tr>
<tr>
<td>url4</td>
<td>language</td>
<td>comparison</td>
<td>survey</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>( q'' )</th>
<th>url5</th>
<th>prolog</th>
<th>language</th>
<th>comparison</th>
<th>survey</th>
</tr>
</thead>
</table>

but there are results for \( q' = \{\text{prolog, comparison, survey}\} \) or \( q'' = \{\text{prolog, language, comparison}\} \)

Formal Definition of the Problem

- Let \( q \) be a query with empty result size.
- **Maximal succeeding subquery problem** \( q' \): returns some results, and there is no other succeeding subquery \( q'' \) that contains \( q' \)
- **Minimal failing subquery** \( q^* \): is a failing subquery of \( q \) but any of its subqueries are succeeding
Solution 2: Relaxation of Boolean Queries

- Godfrey [1997] studied extensively the problem of empty result set for Boolean queries, i.e., queries that contain a set of key-words and fail to return any item

- **Maximal succeeding subquery problem**
  - one of these succeeding subquery can be found in $O(|q|)$,
  - two in $O(|q|^2)$,
  - all makes the problem intractable

- **Minimal failing subquery problem:** similar results as above

- Even if computationally tractable, is the user able to understand the suggestion?

---

Example: showing all minimal failing subqueries

*ShowMe:* Please enter your query:

*User:* price $\leq$ 1000, month = august, region = ireland, persons = 2, duration = 14, type = skiing, accom = flat, transport = plane

*ShowMe:* There are no matches for the following combinations of constraints in your query:

  - price $\leq$ 1000, region = ireland
  - month = august, type = skiing
  - region = ireland, persons = 2
  - region = ireland, type = skiing
  - region = ireland, transport = plane
  - type = skiing, transport = plane
  - accom = flat, transport = plane
  - price $\leq$ 1000, month = august, transport = plane
  - price $\leq$ 1000, persons = 2, transport = plane
  - price $\leq$ 1000, duration = 14, transport = plane
  - price $\leq$ 1000, duration = 14, type = skiing, accom = flat

To solve this problem, you need to relax one of the constraints in each of the unmatched combinations.

By relaxing transport you can eliminate 6 of the unmatched combinations

[McSherry, 2004]
Maximal succeeding subquery

- Showing the minimal failing subqueries requires to the user a huge effort: understanding how to repair the original query
  - The user must look at all the minimal failing queries
  - Then determine a constraint to remove in each of these
- Presenting to the user (some) maximal succeeding subqueries requires considerably less efforts
  - The user must only choose one of them!
Interactive Query Management Architecture

[Mirzadeh et al., 2004]
[Mirzadeh et al., 2005]

Intelligent Query Manager

1. Retrieve candidate products $P$ using content query (filter-based)
2. If too many candidates,
   Compute & return query tightening suggestions
3. If no candidates,
   Compute & return query relaxation suggestions
4. Otherwise,
   Return $P$
Relaxation

- **Feature Filtering**: identify constraints that can be relaxed.
- **Single Constraint Relaxation**: for each constraint, find a relaxed form.
- **Alternative Query Execution**: a set of relaxed queries is evaluated and the new number of items retrieved is counted.
- **Analysis of Results**: for each query if there are still empty result sets, then change the relaxation.

---

Our “Naïve” Approach

- A technology that suggests to the user what are the **simplest** successful relaxations:
  - Look for all relaxed sub-queries that change one single constraint and produce some results
  - Present all these relaxed queries to the users without sorting them
  - If two or more constraints should be relaxed this is done only if they belong to the same Abstraction Hierarchy (they refer to the same concept from the user point of view)
  - If a successful relaxation is found on a feature then all the more abstract features are not considered (“suggests only the more specific”).
Query Language

- An Item space is represented as \( X = \prod_{i=1}^{n} X_i \) where each \( X_i \) is a set of literals.
- \( x_i \) refers to a feature of \( X \) with its associated domain \( X_i \).
- A query is a conjunction of constraints over features:
  \[ q = c_1 \land c_2 \land \ldots \land c_m \] where \( m \leq n \) and
  \[
  c_k = \begin{cases} x_k = \text{yes} & \text{if } x_k \text{ is boolean} \\ x_k = v & \text{if } x_k \text{ is nominal} \\ l \leq x_k \leq u & \text{if } x_k \text{ is numerical} \end{cases}
  \]
- Example:
  \[ q: \{ \text{country} = \text{Italia}, \text{area} = \text{Val di Fassa}, \text{city} = \text{Canazei}, 2 \leq \text{category} \leq 3, 25 \leq \text{cost} \leq 40, \text{parking} = \text{yes} \} \]

Feature Abstraction Hierarchy

- Each FAH (Feature Abstraction Hierarchy) is a relationship among a set of features of an item space (e.g. accommodation).
- **Example 1 – exact functional dependency:** Destinations are described with "country", "county" and "city" features. FAH(country, county, city) means that country is more abstract than county, and county is more abstract than city.
- **Example 2 - approximate functional dependency:** Accommodations are described with "category" (1 star, 2 star ...) and "price" features. The FAH(category, price) expresses the fact that the price is more fine grain (less abstract) than the category to express the cost.
Conditional Entropy

- FAH can be defined:
  - Exploiting domain expertise
  - Identifying logical dependencies
  - By identifying approximate functional dependency in the data (e.g. by conditional entropy)
- Conditional entropy measures (possibly reduced) unpredictability of an event given knowledge of a (different) event
- For two random variables $X$ and $Y$, the conditional entropy of $X$ given $Y$ is
  \[
  H(X | Y) = -\sum_{x \in X} \sum_{y \in Y} p(X = x, Y = y) \log[p(X = x | Y = y)]
  \]
- $H(X|Y) \leq H(X)$: i.e., the knowledge of $Y$ always reduces uncertainty on $X$
- $H(X|Y) = 0$: i.e., iff there is a functional dependency between $Y$ and $X$ ($X$ is a function of $Y$) then conditional entropy is 0.
- $H(X|Y)$ very close to 0 means that $Y$ almost determine $X$

Query Relaxation: Example

1) Constraints in a query are grouped according to the FAHs defined for the item space of the query
2) In each group of constraints, the relaxation algorithm considers first the constraint on the less abstract feature
3) If a succeeding sub-query is found, ok it consider another group, otherwise it relaxes another feature (the next more abstract)
Example: cont.

\[
\{\text{country}=\text{Italy}, \text{area}=\text{Val di Fassa}, \text{city}=\text{Canazei}, 2<= \text{category} <=2, 25<= \text{cost} <=40, \text{parking}=\text{yes}\}
\]

<table>
<thead>
<tr>
<th>Country</th>
<th>Region</th>
<th>City</th>
<th>category</th>
<th>Cost</th>
<th>Parking</th>
</tr>
</thead>
<tbody>
<tr>
<td>Italy</td>
<td>Val di Fassa</td>
<td>Canazei</td>
<td>2</td>
<td>33</td>
<td>no</td>
</tr>
<tr>
<td>Italy</td>
<td>Val di Fassa</td>
<td>Canazei</td>
<td>2</td>
<td>38</td>
<td>no</td>
</tr>
<tr>
<td>Italy</td>
<td>Val di Fassa</td>
<td>Canazei</td>
<td>3</td>
<td>45</td>
<td>yes</td>
</tr>
<tr>
<td>Italy</td>
<td>Val di Fassa</td>
<td>Moena</td>
<td>2</td>
<td>34</td>
<td>yes</td>
</tr>
<tr>
<td>Italy</td>
<td>Val di Fassa</td>
<td>Moena</td>
<td>2</td>
<td>45</td>
<td>yes</td>
</tr>
</tbody>
</table>

\[q_1:\{\text{country}=\text{Italy}, \text{region}=\text{Val di Fassa}, 2<=\text{category}<=2, 25<=\text{cost}<=40, \text{parking}=\text{yes}\}\]

The system suggests to expand the search to the whole Val di Fassa

\[q_2:\{\text{country}=\text{Italy}, \text{region}=\text{Val di Fassa}, \text{city}=\text{Canazei}, 2<=\text{category}<=3, 25<=\text{cost}<=50, \text{parking}=\text{yes}\}\]

The system suggests to consider packages that cost a more and in a better category

\[q_3:\{\text{country}=\text{Italy}, \text{region}=\text{Val di Fassa}, \text{city}=\text{Canazei}, 2<=\text{category}<=2, 25<=\text{cost}<=40\}\]

The system suggests to relax the request of having parking at the hotel

---

On-line Evaluation

- 40 users tried to plan their vacation in Trentino Using NutKing
- Half of them used a version with IQM – NutKing+
- The other half used a version that did not support relaxation mechanism – NutKing-

<table>
<thead>
<tr>
<th>Objective Measures</th>
<th>NutKing-</th>
<th>NutKing+</th>
</tr>
</thead>
<tbody>
<tr>
<td>Queries submitted by a user</td>
<td>20 ± 19.2</td>
<td>13.4 ±9.3 *</td>
</tr>
<tr>
<td># of constraints in a query</td>
<td>4.7 ±1.2</td>
<td>4.4 ± 1.1</td>
</tr>
<tr>
<td>Avg query result size</td>
<td>42.0 ± 61.2</td>
<td>9.8 ±14.3**</td>
</tr>
<tr>
<td># of times relaxation suggested</td>
<td>n.a.</td>
<td>6.3 ± 3.6</td>
</tr>
<tr>
<td># of times the user accepted a suggested relaxation</td>
<td>n.a.</td>
<td>2.8 ± 2.1</td>
</tr>
</tbody>
</table>
Off-line Evaluation

- Using the queries really submitted by the users
- Re-execute the queries to test different relaxation algorithms

<table>
<thead>
<tr>
<th>Catalogue</th>
<th># of users’ queries</th>
<th># of failing queries</th>
<th># succ. subq. found without using FAH</th>
<th># succ. subq. found by using FAH</th>
</tr>
</thead>
<tbody>
<tr>
<td>Accommodation</td>
<td>186</td>
<td>112 (60%)</td>
<td>73 (65%)</td>
<td>94 (84%)</td>
</tr>
<tr>
<td>Location</td>
<td>116</td>
<td>64 (55%)</td>
<td>29 (45%)</td>
<td>50 (78%)</td>
</tr>
<tr>
<td>Event</td>
<td>92</td>
<td>57 (62%)</td>
<td>39 (68%)</td>
<td>52 (91%)</td>
</tr>
<tr>
<td>Sport</td>
<td>102</td>
<td>49 (48%)</td>
<td>31 (63%)</td>
<td>46 (94%)</td>
</tr>
<tr>
<td>Total</td>
<td>496</td>
<td>279 (56%)</td>
<td>172 (62%)</td>
<td>232 (83%)</td>
</tr>
</tbody>
</table>

Relaxation Applicability: Accommodation Search

- All queries
- Failing queries
- Failing queries repaired by the algorithm
Empirical Evaluation: Location catalogue

- All queries
- Failing queries
- Failing queries repaired by the algorithm

Interactive Query Management Architecture

- Client
- Query
- Interactive Query Management
  - Tighten
  - Relax
  - Result Analyzer
  - Query Translation
  - Data Processing Engine
  - Product Catalogues

-A Set of Products
- Suggested Features
- Relaxed Queries

[Mirzadeh et al., 2004]
[Mirzadeh et al., 2005]
Query Tightening

**Issue**: when a large number of products satisfy a query $q$ – the user cannot browse all of them

**Solution**: For each remaining feature (i.e., not already constrained in $q$) compute a score (= how good the feature is to reduce the result set) and suggest to the user highly scored features

- **Measures of goodness for a feature**:
  - $H$: Estimated entropy of the feature values
  - $P$: Popularity of feature (frequency of usage)
  - Feature utility = (how likely it is that feature would be used) * (expected reduction in the number of results) = $P \times H$

- Query tightening is an option for users that do not want to rely on system automatic ranking of results (look at the top products).
Scoring a Feature: Entropy and Popularity

- Entropy of a feature $x_i$ on the result set $R$ of a query $q$

\[ H^i_R = - \sum_{v \in R} p^i(v, R) \log(p^i(v, R)) \]

\[ p^i(v, R) = \frac{\text{# of items in } R \text{ with } x_i = v}{|R|} \]

- Popularity of a feature

\[ p^i = \frac{\text{# of queries constraining } x_i}{\text{total number of queries}} \]

Entropy vs. Popularity Example

The “best” feature, according to entropy is “Hotel size” but it is probably less interesting than hotels in the “Center”. 

\[ H(\text{Center}) = -0.9 \log(0.9) - 0.1 \log(0.1) = 0.141 \]

\[ H(\text{Hotel size}) = -0.3 \log(0.3) - 0.25 \log(0.25) - 0.45 \log(0.45) = 0.463 \]
Scoring a Feature: Utility

- **Model**: systems states $S_1, ..., S_M$ systems actions $A_1, ..., A_N$, and $P(S_j | A_i, E)$ probability to reach state $S_j$ applying action $A_i$ (with evidence $E$).
- We assume that $A_i$ = "system suggests $x_i$ for tightening"
- $S_{2i-1} = "user accepts $A_i"; S_{2i} = "user rejects $A_i"

![Diagram of system states and actions]

**Expected Utility of action $A_i$, given the evidence $E$ is**

$$EU(A_i | E) = \sum_{j=1}^{M} U(S_j) p(S_j | A_i, E)$$

- $U(S_{2i-1}) = H_{R_i}$: the utility of the accept tightening state is the feature entropy in the result set
- $U(S_{2i}) = 0$: the utility of the reject tightening state is 0 (the system made an unnecessary suggestion)

$$p(S_{2i-1} | A_i, E) = \beta p_i$$

- The probability of accepting the suggestion is considered to be proportional to $p_i$, the popularity of the feature, and $\beta$ is a parameter determined experimentally so that the overall acceptance rate is that found in real interaction
Query Tightening Evaluation

- Interactions are simulated – you cannot try all different scoring mechanisms with real users.
- For each test item (products) we simulate a user trying to find an item with that characteristics.
- Simulated interactions are initialized with a starting query that constrains some features to have the values found in the test item.
- If the query count is above a threshold then a feature is suggested (the feature, not yet constrained, that has the best score).
- The simulated user may use the recommended feature $x_i$ and tighten the query (adding a constraint on that feature) – the decision is taken with probability $\beta p_i$.
- We measure the (average) interaction length and the (average) size of the result set at each iteration.

Example

<table>
<thead>
<tr>
<th>Country</th>
<th>Region</th>
<th>City</th>
<th>Category</th>
<th>Cost</th>
<th>Parking</th>
</tr>
</thead>
<tbody>
<tr>
<td>Italy</td>
<td>Val di Fassa</td>
<td>Canazei</td>
<td>2</td>
<td>33</td>
<td>no</td>
</tr>
<tr>
<td>Italy</td>
<td>Val di Fassa</td>
<td>Canazei</td>
<td>2</td>
<td>38</td>
<td>no</td>
</tr>
<tr>
<td>Italy</td>
<td>Val di Fassa</td>
<td>Canazei</td>
<td>3</td>
<td>45</td>
<td>yes</td>
</tr>
<tr>
<td>Italy</td>
<td>Val di Fassa</td>
<td>Moena</td>
<td>2</td>
<td>34</td>
<td>yes</td>
</tr>
<tr>
<td>Italy</td>
<td>Val di Fassa</td>
<td>Moena</td>
<td>2</td>
<td>45</td>
<td>yes</td>
</tr>
</tbody>
</table>

Test item

- Italy Val di Fassa Canazei 2 38 no

Simulation trace

1) First feature = “city” – the simulated user builds the query $Q_1 = \text{city=}\text{canazei}$ – the result set size = 3
2) If the algorithm suggests feature “category” – then simulated user will tighten the current query $Q_2 = \text{city=}\text{canazei AND category=}2$ (with probability $\beta \cdot \text{Popularity(category)}$) – the result set size is 2
Evaluation: Accommodation Catalogue

User always accept to tighten

User accepts to tighten with probability $\beta p^i$

$\beta$ is such that the overall acceptance probability is close to that measured in experiments with users

---

Evaluation

Destinations – Dietorecs catalogue

Accommodations – Dietorecs catalogue
Summary: Interactive Query Management

- In conversational recommender systems based on query processing:
  - The system: suggests how to modify the query
  - The user modifies (or not) the query
- The strong assumption of this interaction style is that the user is "knowledgeable" about what is searching and she is trying to locate the searched item
- This is not always true the user may use the system to discover what she want, to learn something about the product.
Travel Decision Making and Information Seeking

- Customers’ objectives are frequently ill-defined and sometimes contradictory
- Travel product and services are highly fragmented and diverse - it is difficult to transparently communicate the content of a repository to the user
- Users frequently try to determine the limits, constraints and capabilities of a system, or tend to browse around.

Previous approaches have assumed the user is searching for something (service/product) that has certain attributes.

---

Information Search Process

```
Need Recognition
       /    \       /
      /      \     /
     /        \   /
    /          \ /
   /           \
Insufficient

Internal Information Search

External Information Search

Evaluation

Final Destination

Postpurchase Evaluation
[Insufficient] [Sufficient]

[Crotts, 1999]
```
Exploratory Search

Information Seeking Features

- There is no single best strategy for finding information
- The strategy depends on:
  - the nature of the information the user is seeking,
  - the nature and the structure of the content repository,
  - the search tools available,
  - the user familiarity with the information and the terminology used in the repository,
  - and the ability of the user to use the search tools competently.
Attribute-Based Search
When an Attribute-Based Search May Fail

- The user is seeking suggestions, hints, and inspiration rather than options that must optimize a collection of decision criteria.
- The user does not have knowledge of the tourism jargon that is typically used in the description of travel products and services.
- The user can be intimidated and even not able to use advanced search tools based on queries – conjunction of constraints.
- The preferences are not defined before the search process but are “constructed” while learning about available products [Bettman et al., 1998].

The User Interface

[Diagram of the User Interface]

[Ricci et al., 2005b]  
http://dietorecs.itc.it
Case/Session Model

Case

Collaborative Features

travel-party = single, budget= 20-40, sport=T, eno-gastronomy=T

Content Queries

DESTINATION where rockclimbing=T AND hiking=T

HOTEL where category=3 AND parking=T AND cost<40

Cart

Destinations

name=Canazei, rockclimbing=T, hiking=T, museum=F, canoeing=T

name=Molveno, rockclimbing=T, hiking=T, museum=F, canoeing=F

Accommodations

type=hotel, name=Inns, category=3, cost = 35, parking=T
Preference-base Feedback

- Preference-based feedback is a simple type of feedback, allowing the user to indicate a simple preference for one product suggestion over another:
  - for example, “show me more like product B”
- It can often be used in situations where the user has very limited understanding of the product features, and yet can indicate a product preference
- The downside is that offers the recommender system very limited information with which to inform the next cycle.

Seeking for Inspiration

![Diagram](image-url)

I-Like(c_i) → Retrieval → Selection → Explanation

user → (c_1, c_2, c_3, c_4, c_5, c_6) → Browsed Cases → Presentation

Case Base
**First Retrieval.** With a seed case \( c \) (the current case or a random case) the system searches for the \( M \) most similar cases in the case base.

**Case Selection.** The \( M \) cases retrieved from the memory are analyzed to select smaller subset of candidates to be presented to the user: minimize the sum of the similarities between them.
The Logical Components

**Explanation.** Identify the attributes that are peculiar to one case and are not common among the six selected cases.

**Presentation.** Selects some images, taken from the products/services included in the travel, to illustrate pictorially the case content.
User Feedback. The user browses the offers, and eventually to provide a positive feedback on one of these cases. This feed-back, i.e., the liked case is given as input for next retrieval.

Second retrieval. The procedure described above is repeated, but the seed case is now the case that received positive feedback, and the number of cases retrieved from the case base \((M)\) is decrease by a factor \(0 < \lambda < 1\).
Seeking for Inspiration (1)

User’s selection

Seeking for Inspiration (2)

User’s selection
Case Similarity

Similarity of two case nodes is given by the similarity of the children nodes – recursively down to the leaves.

Empirical Evaluation

- The experimental user evaluation took place during March and April 2004
- 150 users – mainly university students between 22 and 40 years – equally distributed across Italy, Austria and USA
- Training phase: ‘finding a hotel in a given regional area at a given price range’
- Test phase: ‘finding an appropriate holiday trip to Tyrol at a given daily compound budget per person’
- The proposed recommendation function was one among a range of those supported by Dietorecs
- Maximum time allowed 50 min
- The interaction session was automatically logged
- The user was asked to complete an evaluation questionnaire.
Evaluation Results

- Exactly 50% of the users – independently from their nationality – did consult the feature “Seeking for inspiration”
- No differences with respect to the share of finished travel plans or single items they found – with respect to other recommendation functions
- No worse perception of the dimensions ease-of-use, outcome/efficiency, reliability, or overall system satisfaction for those users who used the Seeking for Inspiration function.

Off-line Evaluation

- We made a simulation (leave-one-in)
- A target case is selected (among 45 cases – DieToRecs cases acquired with real user interactions – first evaluation)
- Then the simulation starts from a random seed case
- We measured the number of steps required to find the target case among the six cases displayed
- We varied the metric used
  - Only collaborative features (i.e., general characteristics of the travel and the traveler)
    - Using a flat (vector-based) representation
    - Using a tree-based representation
  - Using the collaborative features and the products in the cart
Similarity does not go to 1 because sometimes the process converges not to the target case but to a case only similar to the target.

Better results for the structured metric working on the full case description.
Comments

- There is no “best” strategy for all users and all decision making situations
- Recommender systems must offer multiple search strategies
- The same user may use two antipodal strategies in different situations (even in the same recommendation session)
- The suggested approach requires a smaller cognitive load than analytical search strategies
- The approach complements other search strategies – based on keywords or structured queries.

Summary of Benefits

- Advantages:
  - drastic reduction in user input;
  - the capability of supporting users that do not have yet a clear and stable preference set;
  - the great potential in helping user in understanding and eliciting their preferences and finally the straightforward and simple interaction model.
- Disadvantages: does not fit all users
Comparison-Based Method: Summary

1. new items are **recommended** to the user based on the current query
2. the user **reviews** the recommendations and indicates a preferred case
3. the users feedback is used to **revise** the query for the next recommendation cycle
4. The recommendation session terminates either when the user is presented with a suitable item or when they give up.

Example: Comparison-Based

<table>
<thead>
<tr>
<th>Features</th>
<th>C1</th>
<th>C2</th>
<th>C3</th>
<th>C4</th>
</tr>
</thead>
<tbody>
<tr>
<td>Manufacturer</td>
<td>Dell</td>
<td>Sony</td>
<td>Compaq</td>
<td>Fujitsu</td>
</tr>
<tr>
<td>Memory</td>
<td>256</td>
<td>128</td>
<td>128</td>
<td>256</td>
</tr>
<tr>
<td>Monitor Type</td>
<td>15</td>
<td>14</td>
<td>15</td>
<td>12</td>
</tr>
<tr>
<td>Processor</td>
<td>Celeron</td>
<td>Laptop</td>
<td>Laptop</td>
<td>Laptop</td>
</tr>
<tr>
<td>Speed</td>
<td>700</td>
<td>600</td>
<td>700</td>
<td>650</td>
</tr>
<tr>
<td>Disk</td>
<td>20</td>
<td>20</td>
<td>40</td>
<td>20</td>
</tr>
<tr>
<td>Price</td>
<td>1300</td>
<td>999</td>
<td>1200</td>
<td>1000</td>
</tr>
</tbody>
</table>

- If the user selects C3 and has been presented all these four cases we might exploit the fact that she has “preferred” C2 to the others.
Comparison-based algorithm

1. define Comparison-Based-Recommend(Q, CB, k)
2. begin
3. Do
4. R ItemRecommend(Q, CB, k)
5. c_p UserReview(R, CB)
6. Q QueryReview(Q, c_p, R)
7. until UserAccepts(c_p)
8. end

9. define ItemRecommend(Q, CB, k)
10. begin
11. CB sort cases in CB in decreasing order of their sim to Q
12. R top k cases in CB'
13. return R
14. end

15. define UserReview(R, CB)
16. begin
17. c_p user selects best case from R
18. CB CB - R
19. return c_p
20. define QueryReview(Q, c_p, R)
21. begin
22. R' R - {c_p}
23. For each f_i in c_p
24. Q update(Q, f_i, R')
25. end For
26. return Q
27. end

[McGinty and Smyth, 2002]

Updating the Query

- **MLT - More Like This**: take each feature of the preferred case, f_i \in cp, as a new query feature
- **pMLT - Partial More Like This**: transferring a feature value from the preference case if none of the rejected cases have the same feature value
- **wMLT - Weighted More Like This**: transfers all preferred features, (f_i \in cp), to the new query but weights them s.t. features (values) that are specific to the selected case get higher weights (e.g. “manufacturer” get weight 1 and “type” \frac{1}{4})
- **LLT - Less Like This**: exclude cases that have the feature-values equal to the rejected cases (more details in the paper)

[McGinty and Smyth, 2002]
Comparison of query update methods

- The weighted more like this method is the best.

Critiquing
Critiquing Interaction

- It allows a user to provide feedback at the level of an individual product feature (whereas comparison based approach uses only a preference at the level of case)
  - “show me more like product A but cheaper”
- During the next cycle those products that are similar to Product A but that have a lower price will be shown
- Critiquing is popular because it is informative enough to efficiently guide the recommender system through a complex product-space
- Yet it does not overburden the use when it comes to providing feedback.

[McGinty and Smyth, 2003]
Comparison-Based and Critiquing

- Same as before but when a good example is provided as feedback a critique is also given (e.g. the price should be less than 2000)
- In the first step, all items that fail to satisfy the critique (those with price features > 2000) are eliminated
- In the second step, the remaining items are ranked according to their similarity to the updated query, and the top k are selected.

Comparison-based algorithm and critiques

```
1. define Comparison-Based-Recommend(Q, CB, k)
2. begin
3. Do
4. R ← ItemRecommend(Q, CB, k)
5. c_p ← UserReview(R, CB)
6. Q ← QueryRevise(Q, c_p, R)
7. until UserAccepts(c_p)
8. end
9. define ItemRecommend(Q, CB, k)
10. begin
11. CB′ ← sort cases in CB in decreasing order of their sim to Q
12. R ← top k cases in CB′
13. return R
14. end
15. define UserReview(R, CB)
16. begin
17. c_p ← user selects best case from R
18. CB ← CB − R
19. return c_p
20. define QueryRevise(Q, c_p, R)
21. begin
22. R′ ← R − {c_p}
23. For each f_i ∈ c_p
24. Q ← update(Q, f_i, R′)
25. end For
26. return Q
27. end
```

[McGinty and Smyth, 2003]
Incorporating the critique

- [McGinty and Smyth, 2003] use the following approach
- Before re-ranking the cases, discard those not satisfying the critique

\[
\text{define ItemRecommend} (Q, CB, cr, k)
\begin{align*}
\text{begin} \\
CB' & \leftarrow \{\text{cases in CB that satisfy cr}\} \\
CB'' & \leftarrow \text{sort CB' in decreasing order of sim to Q} \\
R & \leftarrow \text{top k cases in CB''} \\
\text{return R}
\end{align*}
\text{end}
\]

Cases shown

- Cases shown to the user once will not be shown anymore (line 18 of comparison-based algorithm)
- This is typical of McGinty and Smyth
- Other approaches (Pu & Faltings, Ricci & Nguyen) leave in the case base all the products/cases that are shown to the user
- Rationale: product selection is not a linear process – the user may revise her preferences and reconsider products that were discarded some steps earlier.
Problems: with “more like this” strategy

- During each cycle the recommender select those products that are most similar to the user’s query
- Similarity-based approach can lead to potentially redundant product suggestions
  - products may be suggested to the user that very similar to each other
  - if one of these products in unsuitable then there is good chance that the alternative suggestions will be equally unsatisfactory
- For this reason “seeking for inspiration” provide 6 different case (not the most similar)
- For this reason McGinty and Smyth have introduced “Adaptive Selection”.

Adaptive Selection

- Adaptive selection also seeks to introduce diversity into the recommendation process
  - tuning the degree of diversity and similarity used during product selection in response to user feedback
- Adaptive selection attempts to mirror how real-life sales assistants adapt their recommendations in response to user feedback by balancing the importance of similarity and diversity:
  - sometimes diverse suggestions will be made in an attempt to identify the user’s broad interests,
  - other times less diverse, more similar suggestions will be made in order to focus in on some narrow set of products that appear to be of interest to the user.

[McGinty & Smith, 2003]
Critiquing with Adaptive Selection

- If the user selects the carried preference (the case selected in the previous interaction) in the current cycle it must mean that they are unhappy with the k-1 new alternatives in that cycle
  - Hence more diverse products should be shown
- If the user ignores the carried preference and selects one of the newly recommended items then the recommender must be correctly focused
  - Hence products similar to that selected by the user

Diversity: Bounded Greedy

- A first pass over the recommendable items ranks the available product cases according to their similarity to the current user query
- A second pass sequentially transfers selected cases from this ranked list to the recommendation list:
  - the case that is selected (next) is the one maximizes the product of its similarity to the target query and its diversity relative to the cases that have already been selected

\[
\text{RelDiv}(c, C) = \frac{\sum_{i=1}^{n} (1 - \text{Sim}(c, C_i))}{n}
\]

If C=\{\} then RelDiv(c, C) = 1

[Smyth and McClave, 2001]
1. define \textbf{Comparison-Based-Recommend}(q, CB, k)
2. \(i_{p+1}, i_p \leftarrow \text{null}\)
3. \textbf{do}
4. \(R \leftarrow \text{ItemRecommend}(q, CB, c, k, i_p, i_{p+1})\)
5. \(i_{p+1} \leftarrow i_p\)
6. \(<i_{p}, c> \leftarrow \text{UserReview}(R, CB)\)
7. \(q \leftarrow \text{QueryRevise}(q, i_p, R)\)
8. \textbf{until UserAccepts}(i_p)
9. define \textbf{QueryRevise}(q, i_p, R)
10. \(q \leftarrow i_p\)
11. return \(q\)
12. define \textbf{UserReview}(R, CB)
13. \(i_p \leftarrow \text{user's preferred case from } R\)
14. \(c \leftarrow \text{user critique for some } f \text{ in } i_p\)
15. \(R \leftarrow R - \{i_p\} \text{ carrying the preference}\)
16. \(CB \leftarrow CB - R\)
17. return \(<i_p, c>\)

18. define \textbf{ItemRecommend}(q, CB, c, k, i_p, i_{p+1})
19. \(CB^' := \{i \in CB \mid \text{Satisfies}(i, c)\}\)
20. if\((i_p \neq \text{null}) \& \& (i_p == i_{p+1})\)
21. \(R \leftarrow \text{BoundedGreedySelection}(q, CB, k, b)\)
22. else
23. \(CB^" \leftarrow \text{sort } CB^' \text{ in by decreasing sim to } q\)
24. \(R \leftarrow \text{top } k \text{ items in } CB^"\)
25. return \(R\)

26. define \textbf{BoundedGreedySelection}(q, CB, k, b)
27. \(CB' := \text{bk items in } CB \text{ that are most similar to } q\)
28. \(R := \{\}\)
29. \(\text{for } j := 1 \text{ to } k\)
30. \(\text{Sort } CB' \text{ by Quality}(q, i, R) \text{ for each case } i \text{ in } CB'\)
31. \(R := R + \text{First}(CB') \text{ the first in } CB' \text{ is added}\)
32. \(CB' := CB' - \text{First}(CB')\)
33. \textbf{EndFor}\)
34. return \(R\)

... where \textbf{Quality}(q, i, R) = a \text{Sim}(q, i) + (1-a) \text{RelDiv}(i, R)

\textbf{Carrying the preference}

- ...means that the selected case is shown at the next recommendation cycle
- In theory carrying the preference may lead to inefficiencies because the carried preference takes up a valuable slot in each cycle, thus limiting recommendation coverage
- However, this is compensated for because carrying the preference helps protect against false-leads
  - If none of the \(k-1\) new cases are relevant then by reselecting the carried preference the user is at least maintaining the previous best recommendation rather than being forced to accept a lower quality recommendation.
Non Incremental Critique

- Note that the critiques given at a certain stage of the interaction are lost at the next step
- The system locally adapt to user critiques and then forget
- There are incremental approaches that maintain the critiques given in successive steps (see next presentation on MobyRek and [Reilly et al., 2005])
- Incremental approaches must cope with conflicts in the critiques given in the same session (e.g. maintaining only the most recent).

Evaluation by Simulation

- Measure the average number of cycles and average unique items that must be presented to a user in a typical recommendation session
- Using a leave-one-out methodology
  - each case (test item) of a data set is temporarily removed
  - it serves as base case for a set of queries constructed by taking random subsets of item features
  - the item that is most similar to the original base is the target (of the simulation)
  - during each recommendation cycle 3 cases are recommended to the user
  - each time the most similar recommendation to the target is chosen as the users preference, and this is the case that is critiqued (in line with a random feature of the base)
  - each query (test item) is satisfied when the target case is returned in a recommendation cycle.
Example

Example of simulated critique

- Assume that the price in the base case is 2500 (i.e. in the test case removed from the case base in the simulated session)

- The (simulated) user is critiquing a case with price 2000 – this is the case most similar to the target

- Then a random feature is extracted

- If the feature, for instance, is PRICE, then a critique of the form [Price, >, 2000] is generated.
Data Sets

<table>
<thead>
<tr>
<th>CASE ID</th>
<th>DISTILLERY</th>
<th>AGE</th>
<th>PROOF</th>
<th>SWEETNESS</th>
<th>PEATINESS</th>
<th>COLOR</th>
<th>NOSE</th>
<th>FLAVOR/PALATE</th>
<th>FINISH</th>
</tr>
</thead>
<tbody>
<tr>
<td>Case 500</td>
<td>The Glenlivet</td>
<td>13</td>
<td>40</td>
<td>7</td>
<td>4</td>
<td>gold</td>
<td>sweet</td>
<td>medium-peat</td>
<td>full-body</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>CASE ID</th>
<th>REGION</th>
<th>SEASON</th>
<th>TYPE</th>
<th>ACCOMMODATION</th>
<th>DURATION</th>
<th>TRANSPORT</th>
<th>NO. OF PEOPLE</th>
<th>PRICE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Case 1021</td>
<td>Italy</td>
<td>December</td>
<td>Skiing</td>
<td>2 star</td>
<td>7 days</td>
<td>plane</td>
<td>2</td>
<td>$ 2800</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>CASE ID</th>
<th>MANUFACTURER</th>
<th>TYPE</th>
<th>MONITOR SIZE</th>
<th>MEMORY</th>
<th>PROCESSOR</th>
<th>PROCESSOR (MHz)</th>
<th>PRICE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Case 500</td>
<td>DELL</td>
<td>Laptop</td>
<td>10”</td>
<td>512 MB</td>
<td>PENTIUM II</td>
<td>500</td>
<td>$ 3500</td>
</tr>
</tbody>
</table>

Results

- Whiskey data: Standard critiquing requires 13 cycles (on simple queries) and 38 unique case – with adaptive selection a reduction of 31% in recc. length and (51% in number of cases examined by the user)

- Similar reductions are observed even if the simulation is terminated when a case similar to the target is returned (not exactly the target!)

- Similar behavior is observed if we insert some noise in the similarity metric – simulating a user that is not able to evaluate exactly the best option (most similar to the target).

[McGinty & Smith, 2003]
Compound Critiques

- Compound critiques operate over multiple features
- Example1: "I want a sportier car", in a car recommender, which operates over a number of different car features; engine size, acceleration and price are all increased
- Example2: "higher performance PC", in a PC recommender simultaneously increase processor speed, RAM, hard-disk capacity and price features
- They carry considerable explanatory power (if suggested by the system) because they help the user to understand common feature interactions
  - in the PC example above the user can easily understand that improved CPU and memory comes at a price.

[Reilly et al., 2004]
Static vs Dynamic Critiques

- Compound critiques may be **hard-coded** by the system designer so that the user is presented with a fixed set of compound critiques in each recommendation cycle
  - these compound critiques may, or may not, be relevant depending on the cases that remain at a given point in time

- **Dynamic** approach to critiquing in which compound critiques are generated on-the-fly, during each recommendation cycle, by mining commonly occurring patterns of feature differences that exist in the remaining cases
  - Ex: see the snapshot shown before

- Dynamic critiques are very closely related to query tightening.

[Reilly et al. 2004]

Critiques and Browsing

- Users cannot be relied upon to provide consistent feedback over the course of a recommendation session

- Many users are unlikely to have a clear understanding of their requirements at the beginning of a recommendation session

- Many users rely on the recommender as a means to educate themselves about the features of a product space

- *Users may select apparently conflicting critiques during a session as they explore different areas of the product space in order to build up a clearer picture of what is available*
Incremental Critiquing

- Current recommender systems that employ critiquing tend to focus on the current critique and the current case, without considering the critiques that have been applied in the past
  - The user can be confused by the fact that previous critiques are discarded (if she use an incremental approach for product filtering)
- But if all the critiques are taken into account we can fall into the “no results” situation quite easily

Incremental Critiquing

- The idea is to maintain a critique-based user model which is made up of those critiques that have been chosen by the user so far
- The critiques that a user has applied so far provide a representation of their evolving requirements
- The problem is how to deal with inconsistent set of critiques
The algorithm

| CB: casebase, U: user-model, q: query, t: chosen critique, r: recommended items |
|---|---|
| 1. define Incremental-Critiquing(q, CB) |
| 2. U = {} |
| 3. t = {} |
| 4. repeat |
| 5. r ← ItemRecommend(q, CB, t, U) |
| 6. t ← UserReview(r, CB) |
| 7. q ← QueryRevise(q, r) |
| 8. U ← UpdateModel(U, t, r) |
| 9. until UserAccepts(r) |
| 10. define UserReview(r, CB) |
| 11. t ← user critique for some feature e r |
| 12. CB ← CB - r |
| 13. return r |
| 14. define QueryRevise(q, r) |
| 15. q ← r |
| 16. return q |
| 17. define UpdateModel(U, t, r) |
| 18. if IsCompound(t) then |
| 19. t-set ← UnitCritiques(t) |
| 20. else |
| 21. t-set ← {t} |
| 22. endif |
| 23. for each t ∈ t-set |
| 24. do |
| 25. U ← U - contradict(U, t, r) |
| 26. U ← U - refine(U, t, r) |
| 27. U ← U + {t, r} |
| 28. EndFor |
| 29. return U |
| 30. define ItemRecommend(q, CB, t, U) |
| 31. CB' ← |i ∈ CB | satisfies(i, t)| |
| 32. CB'' ← sort CB' by decreasing Quality |
| 33. r ← top item in CB'' |
| 34. return r |

[Reilly et al., 2005]

Updating the User Model: Contradict and Refine

- When the system updates the user model (i.e., the collection of critiques made by the user) exploits the following procedures:
  - Compound critiques are split into the elementary critiques
  - Contradict(U, t, r) are the critiques in the accumulated set of critiques (U = user model) that are in contradiction with t
  - THESE ARE REMOVED: hence the idea is that it keeps as many as possible of the old critiques.
  - Refine(U, t, r) substitute the critiques in U that are refined by the new critique t
Integrating Multiple Critiques

- Instead of ordering the products \((c')\) by the similarity with the case recommended at the previous interaction (the selected case by the user – since only one case is shown) ...
- The system try to satisfy as many critiques as possible and to suggest a case similar to that (previously) recommended

\[
Compatibility(c', U) = \frac{\sum_{i} \text{satisfies}(U_i, c')}{|U|}
\]

\[
Quality(c', c, U) = Compatibility(c', U) \times \text{Similarity}(c', c)
\]

- satisfies\((U_i, c')\) is 1 if the critique \(U_i\) is satisfied by \(c'\) (0 otherwise)
- The similarity is decremented by the percentage of the old critiques that are satisfied.

Evaluation: Algorithms

- STD is the standard approach (only the last critique is used)
- IC is the incremental approach
- IC-ENABLE is an approach where after the user model has been revised the system try to remove the critiques in contradiction with the current case (hence compatibility with the current case is always maximal = 1)
Evaluation: procedure

- Each case is temporarily removed
  - It is used to generate queries (taking random subset of the features)
  - It is used to select the case most similar – this is considered the target – the simulation stops when the target is reached
- In each recommendation cycle the simulated user picks a critique that is compatible with the target case (?! The simulated user is assumed to search for the target ?!)

Results: Interaction Cycles

(a) Unit
(b) Compound
(c) Travel Cycles
(d) Travel Cycles
Results

- Query length is the number of features specified in the initial query.

Comparison of Feedback Strategies

<table>
<thead>
<tr>
<th>Feedback</th>
<th>Cost</th>
<th>Ambiguity</th>
<th>Expertise</th>
<th>Interface</th>
</tr>
</thead>
<tbody>
<tr>
<td>Value Elicitation</td>
<td>***</td>
<td>*</td>
<td>***</td>
<td>***</td>
</tr>
<tr>
<td>Ratings</td>
<td>**</td>
<td>***</td>
<td>**</td>
<td>*</td>
</tr>
<tr>
<td>Critique</td>
<td>**</td>
<td>**</td>
<td>**</td>
<td>*</td>
</tr>
<tr>
<td>Preference</td>
<td>*</td>
<td>***</td>
<td>*</td>
<td>*</td>
</tr>
</tbody>
</table>

Key: * Low  ** Moderate  *** High
Pu and Kumar Requirements

<table>
<thead>
<tr>
<th>Requirement</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>R1: Incremental effort of elicitation</td>
<td>The interface should allow users to make an incremental rather than a one-shot effort in constructing their preferences due to the highly adaptive nature of decision process and user’s lack of initial motivation in stating them.</td>
</tr>
<tr>
<td>R2: Any order</td>
<td>The interface should not impose a rigid order for preference elicitation.</td>
</tr>
<tr>
<td>R3: Any preference</td>
<td>The interface should let users state preferences under relevant contexts.</td>
</tr>
<tr>
<td>R4: Preference conflict resolution</td>
<td>The decision search tool should solve preference conflicts by showing partially satisfied results with compromises.</td>
</tr>
<tr>
<td>R5: Tradeoff analysis</td>
<td>In addition to search, the system and the interface should help users perform decision tradeoff analysis, such as “I like this apartment, but can I have something cheaper?” or “I like this apartment, but can I find something bigger?”</td>
</tr>
<tr>
<td>R6: Domain knowledge</td>
<td>The system and the interface should reveal domain knowledge whenever possible.</td>
</tr>
</tbody>
</table>

[Pu and Kumar, 2004]

User based evaluations

- [Pu and Kumar, 2004] compare a ranked list approach (the results can be sorted according to a feature) with a critique-based approach
- They found that the critique-based approach is most efficient (the task can be solved in less steps)
- [Pu and Chen, 2005] show that critiquing can improve the choice made without critiquing in 57% of the cases.
Ranked List Interface

![Ranked List Interface Diagram]

SmartClient – critique-based

![SmartClient – critique-based Diagram]
User tasks [Pu and Kumar, 2004]

1. Find your most preferred apartment.
2. Can you find something closer? You can compromise on one and only one attribute.
3. Can you find something bigger than what you found for question #1? You can compromise on one and only one attribute.
4. Find something which is roughly 100 francs less than the answer to question #1. You can compromise on up to two attributes, but not more.
5. Find an apartment which is 5 square meters bigger than the answer to question #1. You can compromise on up to two attributes but not more.

Time to complete the task

![Average Time](chart.png)
Choice Improvement [Pu & Chen, 2005]

- The user first select a product – without using the critiquing functionality: choice 1
- Then the user can use the critiquing and can eventually revise the selection: choice 2
  - If choice2 /= choice1 then user has improved the accuracy of choice
- Then the user is instructed to browse the full list of products and is asked if choice1 or choice2 or another one is the best (target choice)
Results

- Only 18% of the users found the target choice without using the critiquing
- 75% of the user found the target using the critiquing
- Users where unsure about the fact that choice1 was the target (certainty level 2.8 – on a scale -5,+5)
- Users where more sure about the goodness of the choice after critiquing (certainty level = 3.6)
- *Is this the truth (i.e. better accuracy with critiquing) or the system persuaded better the user after the "critiquing" process?*

Conclusions

- Conversational systems have a better chance to support the user decision process
- There are many approaches to conversational systems (logical query, similarity-based, comparison, critiques)
- There is no best approach: it depends on the user, product and context
- Conversational approaches are really hybrid (not only in the sense of mixing content-based and collaborative-based ideas)
- Evaluation of conversational system is expensive – it needs real user interaction
- Simulating human-computer interaction could be useful.
Questions

- Why constraint relaxation in IQM is useful? Are there other methods to approach that problem? What are the advantages and disadvantages?
- How it works the feature (questions) selection method proposed in IQM? How this could be improved?
- What is the definition of state and utility in IQM? Can this idea be generalized (more states variable – better utility)?
- What are the advantages of preference-based approaches with respect to those based on query (rewriting)?
- In comparison-based approaches what is the difference between preference-based and critiquing-based?
- Examine the simulation procedures for testing the proposed approaches. Are they reasonable? What kind of improvements can be applied?