Recommendation in CHIP

This document describes CHIP project research and activities that relate to recommender systems. Motivations for this page include:

- setting up clear goals for separate papers
- establishing points of collaboration with others, such as Henriette, Petko and other Telematica Instituut projects
- determining requirements and research issues for the user model
- forming the schedule and plan for Lloyd's involvement in CHIP

Table of Contents

- 1. Abstract
- 2. Theories and Hypotheses
  - 2.1. Algorithm Accuracy
  - 2.2. Dialog Strategies
  - 2.3. Interaction
  - 2.4. Other Hypotheses
- 3. Architecture and Requirements
  - 3.1. System Types
  - 3.2. Interface
  - 3.3. Recommendation Processing
  - 3.4. Dialog Strategy
- 4. Methodology
  - 4.1. Recommendation
  - 4.2. Confidence
- 5. Algorithms
  - 5.1. Laplace Smoothing Algorithm
  - 5.2. Statistics-based Algorithm
    - 5.2.1. Description
    - 5.2.2. Psuedocode
    - 5.2.3. Code online
  - 5.3. Flagged PageRank Algorithm
    - 5.3.1. Description
    - 5.3.2. Psuedocode
    - 5.3.3. Example
- 6. Simulator
  - 6.1. Introduction
  - 6.2. Current Status
  - 6.3. Latest Statistics
  - 6.4. Instructions
  - 6.5. Future Work
- 7. Plan
1. Abstract

This work explores applying recommender system technologies to personalize interaction with electronic museum collections. Resulting systems provide more engaging interaction and help users learn their art history more quickly. The goal for the targeted user group, which consists of visitors to museum and museum website, is making their visits more engaging by having them reflect their unique personal interests and tastes. Semantic Web technologies apply by providing representation and processing of metadata of artworks in the museum collection, including art history concepts related to the artworks. Recommender system technologies apply by enabling system to record and estimate varying user interest in the different artworks and related concepts. This work combines these technologies into Semantic Web interfaces to museum collections that adapt based on user interests. It can apply as well to interaction with other domains with Semantic Web representations of concepts for which uses can have varying levels of interest.

2. Theories and Hypotheses

The main theory is that semantics help recommendation. More specifically, the theory is that applying Semantic Web technologies to recommendation systems and interfaces using them helps users in legitimate tasks in ways beyond what other recommender techniques provide alone. The three main subtheories, discussing in the following subsections, are that semantic processing ...

1. improves recommendation accuracy
2. accelerates getting adequate ratings
3. improves interaction in other ways

This section’s subsections present their hypotheses in approximate order of provability.

2.1. Algorithm Accuracy

One contribution is the specification of the concept of semantic-based recommendation, which is a form of content-based recommendation that uses Semantic Web technologies to derive object features from which to find patterns corresponding with user interest. The following hypotheses all regard the accuracy of different algorithms for predicting interest. Their quality is measured by comparing them with each other and with more general and well established algorithms such as social filtering.

Semantic Applying Semantic Web technology improves recommendation accuracy. More specifically: semantic-based algorithms for interest prediction compare well with established content-based and social filtering algorithms. This is a general hypothesis that relates to the other algorithm hypotheses. To test this, a semantic algorithm will be compared with non-semantic benchmarks. If it performs less well, making hybrids of semantic recommendation with other forms may perform better than the other forms alone.
Tag Processing flat tags annotating objects can effectively predict user interest in those objects. In terms of semantics, flat tags are either literals or URIs for which any properties are ignored.

Concept Processing user ratings of concepts helps predict interest in objects the link to. Corresponding input interfaces must let users rate concepts. One can test this by comparing accuracy with the same data without the concept ratings.

Typed property Certain types of object properties contribute more to recommendation accuracy than others. One can test this by comparing accuracy with tags from triples having different predicates. "Hot properties".

Inherited tags Processing inherited tags helps recommendation. Tags are inherited through a subsumptive hierarchy.

Network Processing a connected network of object annotations contributes well to recommendation accuracy. Here we use the flagged ant algorithm and its various variants.

Ontology Some sets of concepts predict interest better than others. These sets can be determined by: class, external ontology, subtree. Running the ARIA website structure through this comparison may shed interesting light on how different ways and approaches for "ontologies" as compared to general Web structure affects recommendation.

2.2. Dialog Strategies

Assuming users prefer dialogs, an algorithm can chose the object or concept whose rating will most benefit the accuracy of the overall user model. How well does strategy for semantic compare with social or content-based? The theory here is that semantic-based algorithms enable strategies for selecting objects for users to rate in order to most efficiently reach the state in which the system can predict user interest across the domain with adequate accuracy, and these compare well with those of content-based and social filtering algorithms.

Hypothesis Users prefer dialog strategy

- users asked if dialog fast enough
- controls: random, fixed expert, fixed calculated for general user

2.3. Interaction

Superficially, semantic recommendation predicts interest levels for objects in a given domain, as content-based and social filtering recommendation do, to generate lists of objects to recommend. However, semantic recommendation enables also adapted user interaction in ways beyond what content-based and social filtering recommendation do because the network of concepts it defines enhances both explanations of user interest and the user’s means of exploring the knowledge domain. The following hypotheses
regard the effectiveness of user interface components. One tests each feature by comparing the satisfaction of users of tours that have personalized version of each feature with tours that have fixed version of the feature and that lack the feature altogether. The later shows if the feature is desired in general. The first shows that personalizing that feature is beneficial to the user.

Recommended Tour Users prefer tours with recommended content over those with fixed content.

- users asked how much they liked it (in various ways)
- tour interface minimized, approved by Rijks, in line with HVNR
- controls: expert made, most popular recommendations, random
- if multiple algorithms, see if users notice “most accurate” as better
  - semantic vs: social, basic content-based (tags)

Explanations Semantic-based algorithms provide explanations that users accept and trust and that help users in navigating the knowledge repository.

Document Structure Semantic-based algorithms can helpfully adjust sorting and clustering of navigation interfaces that the system generates for the user, thus generating document structure in such generated output as personalized tours.

Group Users prefer personalized groups (in rooms, or replace rooms for web tour)

Sub-topic Users prefer tours with sub-topics to those that do not. In addition, user prefer personalized sub-topics to fixed sub-topics.

- users asked about subtopics
- fixed or personalized selection of artworks?
- control subtopics: random, expert, calculated most popular
- semantics required to make this possible

Recurring theme Users prefer tours with recurring themes to those that do not. In addition, user prefer personalized recurring themes to fixed recurring themes.

2.4. Other Hypotheses

- works cross-domain
  - see Schlomo's work
  - movies
    - NetFlix: we can get data (very large)
    - MovieLens: TI has data
    - Ontology?: I have read that someone made one from IMDB. Can we map a movie rating set to this through titles and IMDB IDs?
    - Dutch broadcast: Duine ratings plus CHOICE semantics?
overlap: recommend Beeldenstorm broadcast from CHIP ratings?

- navigation logs can assist recommendation (with Petko and/or Vera)
  - datamining
  - determining similar users in order to infer preferences
- cross domain with same users
  - motivation: start in Rijks with your ratings from: Amazon, E-Bay, MovieLens, NetFlix, etc.
  - primary requirement is means of identifying same users over different domains
  - domain here can be any way of modeling users, such as navigation patterns
  - example
    - user A has no movie ratings, only books
    - user A has similar move ratings to user B
    - user B likes certain books
    - recommend these books to user A
    - replace movies/books with any domain or model such as navigation patterns
- recommendation accuracy benefits from Semantic Web
  - role of each SW construct
    - RDF & RDFS
    - OWL types: all, but especially
      - sameAs: both URIs should be equally recommended
      - reflexive: direction of relation should not affect recommendation
      - transitive: all items on the chain should behave as if directly linked
    - SKOS: broader (hierarchy), Concept, other
      - abstract
    - hybrid semantic with other techniques
      - motivation: social may be more precise, but semantic is more useful for interaction

3. Architecture and Requirements

This section presents our architecture for processing recommendations within a Semantic Web framework. It also presents requirements that various components of this architecture have from algorithms and techniques for processing semantic recommendation.

3.1. System Types

- transparency and control
  - activities: show, explain, provide control
  - confidence
3.2. Interface

This subsection describes the recommendation system interface that this architecture’s techniques apply to.

Rating The system presents objects and concepts to the user. The user can rate each of these. Our implementation presents one to five stars, with three being neutral, less than three negative interest, and more than three positive interest. However the user enters ratings, and whatever the range he sees, on the processing level, ratings vary from -1 (negative interest or appreciation) to +1, with 0 being neutral.

Recommending A recommender system processes user ratings on the rated objects to calculate what other objects are likely to interest the user, and then recommends them. As this calculation most often depends on user ratings, the user typically enters a number of ratings as an initial rating phase before entering the recommendation phase.

Unobtrusiveness The users, rating is a means to a goal rather than a desired activity in and of itself, and thus is obtrusive. Thus the system should minimize the amount of rating it asks of the user. It should thus be unobtrusive as possible while still getting enough ratings to calculate satisfying recommendations. The following subsections identify two important components of unobtrusiveness: completion and efficiency.

Completion The system should be able to determine when the rating phase is complete so it can move the interaction to the recommendation phase. This occurs when the system has enough ratings from the user to provide adequate recommendations. This concept of completion is fuzzy, meaning that systems will likely have to depend on estimations and probabilities rather than absolute certainties. There are two aspect to completion: arrival at and anticipation of completion. Arrival at completion is recognizing when the user has entered enough ratings, meaning the rating phase is complete and the user can expect satisfactory results from entering the recommendation phase. Anticipation of completion means having some indication of how many more ratings are still needed and communicating this to the user. Giving the user an ongoing indication of how many ratings remain can help encourage patience and continued cooperation from the user. It is likely that calculation anticipation is more complex than calculating arrival.
Efficiency The *efficiency* rating interface is how few ratings it requires from the user in order to complete the rating phase. A system can control efficiency by selecting those objects whose ratings are most valuable to reaching completion and asking the user to rate them. In addition, other aspects of the interface may allow the user to encounter or seek other objects and rate these as well.

### 3.3. Recommendation Processing

This subsection shows the types of processing that can occur in this semantic recommendation architecture.

Recommendation One way or another, the system decides which unrated objects to recommend. This means that for each node, the system has a means of determining whether to recommend it or not. A system recommends an object if it determines the user is likely to like it. It cannot know for sure if the user would like it in the absence of an explicit rating from the user. Systems typically calculate this likelihood from other information. This information is typically ratings of other objects or ratings from other users of this and other objects.

Recommendation has several mathematic assumptions that apply to most, if not all, recommender systems. One way or another, most systems treat this likelihood as a probabilistic measure: a probability of 0% to 100% that the user will like an object. This requires, of course, a definition of when a user "likes" an object. In the pilot version of our demo, a user likes an object if he or she gives it four or five stars. It also requires a threshold for the probability: that is, what probability of the user liking an object is required to recommend it. A typical threshold from statistics is 95%. Often, but not always, the probability that a user will like an object is broken up into what rating the system expects, the predicted interest, and how likely it is that the user will give this rating, the confidence.

Predicted Interest For each component of the knowledge base that the user has not rated, the system uses ratings of other objects to calculate a prediction of how the user would rate this one. The system can then recommend to the user objects it figures the user would rate highly.

Confidence While the system can estimate the interesting the user would have in an object, its confidence in this estimate can vary. Given a high predicted interest, the system must also have high confidence in this estimation before recommending the object to the user.

Distribution The calculates its predicted interest for each unrated node from the users ratings on other related nodes. The values of these related rated nodes will have a distribution. If they tend to be the same, they have a low distribution. If they vary widely, then they have a high distribution.

### 3.4. Dialog Strategy
This subsection explains how semantic recommendation systems can make user interaction more efficient.

Information Gain Each explicit rating has an impact on calculating the predicted interest, confidence and distribution of related unrated nodes. A new rating helps by increasing the confidence of the calculations for other nodes. This increases with the number of nodes improved and by how much each node gains confidence. One can calculate the information gain from rating a given node by the total increase in confidence it gives to all nodes it effects.

Overall Confidence It is helpful to the user if the system can determine when it has enough ratings and is ready to take the user to a next phase of the interaction, such as receiving recommendations or starting a personalized tour. We calculate overall confidence as the average confidence of the nodes in the network. If we can determine a threshold of overall confidence that indicates readiness to progress to a next interaction phase, then the system can know when to inform the user.

Rating Request Selection At any given point, the object whose rating will most improve overall confidence is the one whose rating have the highest information gain. If the system can determine this object, then it can ask the user to rate it in order to minimize the number of ratings the user needs to reach the threshold of overall confidence.

4. Methodology

This subsection describes methodologies for measuring how well the techniques and algorithms this work explores apply to the recommendation framework this work assumes. Measurement of accuracy for applying different techniques to various framework components makes of the core of this methodology.

One "validation" measure we have for ontologies is how well they predict user ratings of interest in items. As this is intended for recommendation, one can argue that it would be rather fuzzy for measuring general correctness of a mapping. However, given sufficient user ratings for a collection of artworks, one can automatically "measure" any overlaying ontology, even if the mappings are automatically generated. Theoretically, the inherent concept maps that users effectively have correspond to what their interests and tastes are. Thus, theoretically, if annotations predict interest well, then they correspond with such a concept map.

4.1. Recommendation

The recommender system research community has established means of measuring the accuracy of recommendation algorithms. The primary means is the Leave-N-Out technique. Given a set of real user ratings as a ground truth, Leave-N-Out and other technique measure recommendation accuracy by comparing the recommendations an algorithm generations with the corresponding actual ratings from the rating set. Here,
established information retrieval measurements, particularly precision and recall, indicate the accuracy.

4.2. Confidence

As with formulas for recommendation, formulas for calculating confidence can be imprecise approximations. However, confidence calculation is an important part of forming dialog strategies for recommender systems. Therefore, it is important to have confidence formulas and to ensure and maximize their accuracy. Being able to compare different formulas for confidence helps approach this goal by enabling developers to progressively increase the accuracy of this important function.

The foundation for measuring confidence calculation this work proposes here is measuring how well the confidence calculated for a set of predicted interests correlates with difference between these predicted interests and their ground truth values.

5. Algorithms

5.1. Laplace Smoothing Algorithm

The demo's initial means of determining recommendations comes from calculating the confidence that a concept should be recommended.

Topics The initial formula for confidence for recommending a topic is:

\[
\frac{(\# \text{ linked artworks rated positively} + 1)}{((\# \text{ linked artworks shown user}) + 2)}
\]

Artworks The initial formula for confidence for recommending an artwork is

\[
\sum(P(i)W(i)/N),
\]

where

- \( i \) indicates all rated or recommended properties of an artwork
- \( N \) is the total number of properties the artwork has, not just rated and recommended
- \( P \) is one rated or recommended property of the artwork
- \( W \) is the weight a property has, which is
  - \( 100\% \) when the property is rated, and
  - when the property is recommended, the confidence level of the recommendation

5.2. Statistics-based Algorithm

This section describes a potential next step for calculating recommendations that is based on standard polling statistics. Here, we recommend a concept when its predicted interest has a statistically significant chance of falling in the confidence interval.
Predicted interest The mean (average) of the explicit interests of directly linked nodes.

Statistical significance Achieved when enough directly linked nodes have explicit interests. Given that all directly linked nodes are considered the statistical population, and those with explicit interests are the sample, we apply the standard calculation for a statistically significant sample size for that population size. See the wiki entry.

Confidence interval This is the range of values we want statistically significant certainty the predicted interest will fall in. For recommendations, this is, of course, at the top of the scale. The lower a predicted interest we accept, the larger the confidence interval and, thus, the smaller the required sample size.

5.2.1. Description

Here is a simple model to start with for propagating predicted interest from neighboring explicit interest through a network such as the Rijksmuseum data. Here, each node without an explicit interest gets both a predicted interest and its certainty. These calculations use basic probability and statistics to, simply put, do a poll of objects with a property to see how favored they are to the user.

Mathematically, we treat the neighbors of each node as a population, among which the explicitly rated ones are the sample set. The average of the neighboring ratings becomes the predicted interested. We use the standard statistical formula for calculating the certainty from this sample and population size.

Prorogation here only traversing a single edge (triple). That is, predicted interest is calculated using explicit interest of only directly linked nodes. However, we can infer additional direct properties by querying for certain chains. For example, if a node sits in a hierarchy, such as "Amsterdam", then all nodes linked to it are also then linked to its groups, such as "North Holland", "The Netherlands" and "Europe". All these nodes then become directly linked properties to all nodes linked to Amsterdam.

5.2.2. Psuedocode

for each node
  if no explInst
    total = 0
    count = 0
  for each neighbor
    if neighbor explicit interest
      count++
      total = total + neighbor explicit interest
    if count = 0 predInst = empty
  else
    predInst = total/count
    certain = {certainty(size(sample),size(population)}

5.2.3. Code online
The `confidenceIntervalStudent` function in the Java library for the class Tally might be what we need for the demo's confidence "filter". It calculates the confidence interval of a sample, which is the "rate of error", or the portion of the total value range, with the average in the middle, that we are 95% certain that the true value falls in. I've "guesstimated" that 33% is enough for us. Thus nothing is recommended until we have an average above a certain level, say 66%, and the confidence level returned by this calculation is 33%. We can, of course, play with these levels as time goes on to improve how well the demo runs.

5.3. Flagged PageRank Algorithm

This subsection describes an algorithm proposed by Rogier at the March 12th 2007 Algorithms meeting.

5.3.1. Description

Here we propose the Flagged PageRank algorithm as a variant of PageRank for processing user ratings into semantic-derived recommendations. To apply it, consider a given Semantic Web repository as a node-edge graph. Following the typical PageRank analogy, toss a bunch of ants on nodes throughout the graph and let them wander along the links until we see that their distribution stabilizes. Here, "stabilized" means that the number of ants on each node at any moment remains the same even as they all continue wandering between nodes. PageRank then considers nodes with more ants as considered better connected. For search Web documents, the stereotypical application of PageRank, "better connected" means a search return is more relevant.

The Flagged PageRank algorithm gives each ant a flag holding the rating of the last explicitly rated node it passed through. Its starts by tossing the ants with initially empty flags on the graph. When the graph stabilizes, it looks at the flags on the ants in each non-rated node to calculate the user's predicted interest in that node.

Another aspect of Flagged PageRank is that each time a flagged ant passes through an unrated node, there is a fixed chance that it losses its flag. This introduces the "ripple" effect, in which the impact on predicted interested decreases the further away from a rated node a flagged ant wanders. Therefore, each node on a stabilized graph will have some number of unflagged ants as well as ants for each of the five flag values.

Given this distribution of the flags with various values, including empty, on an unrated node, we apply statistics on these flags to calculate predicted interest. The average of the flagged values becomes the predicted value for the user's interest in that node. The total number of ants on a node, along with the portion of those ants that are unflagged, indicates confidence in this prediction. The distribution of these flagged values is also important, as a wide distribution of values indicates that that node lacks correlation with predicting user interest. A wide distribution indicates lack of confidence in the predicted interest. It also indicates high confidence that that node lacks interest correlation if the node has many flags and few of them are empty. Finally, a node with a large number of
flag of which many are empty indicates a node that is well-connected among unrated nodes. Such nodes indicate high information gain from new user ratings. The system can select such nodes in its rating dialog with the user as part of making the dialog more efficient.

5.3.2. Psuedocode

Below is pseudocode for PageRank in general. This forms the basis for upcoming adaptation for Flagged PageRank and definition of is variants.

```java
compute_statrateddistrib(graph, eps, q, q') {  
    // Initialize variables
    Nodes nodes = graph.nodes();
    int Nnodes = nodes.size();
    boolean converged = true;
    // Initialize nodes
    foreach(Node node in nodes)
        node.oldnorm = 1.0/Nnodes;
    For ( i=0; i++; i==Nstar-1 )
        node.oldp[i] = 1.0/(nodes.size() * Nstar);
    // Main body
    do{
        foreach(Node node in nodes) {
            If(node.rated) {
                // update node like in the unrated case but concentrated in the rating value
                continue; // next node
            }
            For ( i=0; i++; i==Nstar-1 )
                node.p[i] = ((1-q) /Nnodes) * avarage_ratingdistr[node, user][i];
            foreach ( Link link in node.inlinks() ) {
                startnode = link.startnode();
                For( i=0; i++; i==Nstar-1 ){
                    Node.p[i] = qprime* avarage_ratingdistr(node, user)[i];
                    node.p[i] += (1-qprime)* startnode.oldp[i]*q*weight(link);
                }
            }
            normdiff = 0.0;
            node.norm = 0.0;  // = sum_i node.p[i]
            For( i=0; i++; i==Nstar-1 ){
                node.norm += node.p[i]  // = abs(p[i])
                normdiff += abs(node.p[i] - node.oldp[i]);
            }
            converged &&= normdiff < eps* node.oldnorm;
        }
        foreach(Node node in nodes)
            node.oldp = node.p;
            Node.oldnorm = node.norm;
    } while(!converged);
```
Below is pseudocode for Flagged PageRank.

```plaintext
// initialize by spreading ants around nodes
node = nodes.first()
for each ant {
    ant.node = node          ;
    node     = nodes.first() ;
}

// move ants until distribution stabilizes
while nodes.change() > eps { // while not stable
    // move all ants one step
    for each ant {
        // move ant to a linked node
        ant.node = ant.node.linkSelect();
        // flag ant with current rating
        if   ant.node.rating       then ant.flag = ant.node.rating
        // chance of null flag grows with each unrated node passed thru
        else if ( random() < ripple ) then ant.flag = null;
    }
}

// predict ratings
for each node {
    node.predict = average (flagged ants on node) ;
    node.distrib = distribution (flagged ants on node) ;
    sample = count (flagged ants on node) ;
    population = count ( ants on node ) + 1 ;
    node.confid  = pollingConfidence ( sample, population );
}

5.3.3. Example

6. Simulator

6.1. Introduction

This program performs analysis of particular recommender algorithms. The UvA user study rating set is its test set. Analysis involves comparing each explicit user rating for a work with an algorithm's calculation of that rating from the other ratings. This program's SVN is https://svn.win.tue.nl/viewcvs/CHIP/programs/simulator/.

6.2. Current Status

- algorithm is average of neighboring ratings
- simulator takes 20 minutes to run

Previous failed attempts to support hypotheses (required specialized RDF)
• accounting for network distance two has no clear improvement
• different relation types have small effect

6.3. Latest Statistics

The statistics text below is directly copied in from the "analysis" worksheet. We give some starter algorithms for accuracy as benchmarks for the sake of comparison.

<table>
<thead>
<tr>
<th>Algorithm</th>
<th>Average Accuracy</th>
<th>Precision</th>
<th>Recall</th>
</tr>
</thead>
<tbody>
<tr>
<td>random</td>
<td>69.98%</td>
<td>17.95%</td>
<td>33.97%</td>
</tr>
<tr>
<td>average rated neighbors (works)</td>
<td>79.61%</td>
<td>69.65%</td>
<td>26.85%</td>
</tr>
</tbody>
</table>

6.4. Instructions

1. Clear the local Sesame repository "user" and then add the following RDF files into it
   - userRDFs/UvA_HCSusers.rdf
   - standard CHIP.rdf
2. Make analysis.txt (takes 20 minutes)
3. Open analysis.xls in Excel
4. Click worksheet "data" and then "Refresh All" from the "External Data" toolbar
5. Analysis is on tab "analysis"

6.5. Future Work

1. Clean up Excel
   1. Check and clearly state meaning of all columns
   2. As link-able HTML on members website
2. Add benchmarks
   1. Re-introduce random
   2. User's average rating all works
   3. Work's average rating all users
   4. LaPlace smoothing of current demo
   5. Social filtering from Duine (requires getting Duine running)
   6. Others from Duine?
3. Extend work on typed triples
   1. Achieve statistical significance
   2. Add types: predicate type (such as subject vs material)
4. Achieve statistical significance for varying distance ("ripple effect")
5. Try clustering "ghost tags" from ratings and compare with existing tag sets and their hierarchies

7. Plan
Below is my current plan for tasks in CHIP recommender research, in (approximate) order:

- make table of system types and features
- measure accuracy of current demo algorithm on Henriette's data (current activity)
- set up controls/benchmarks/nulmetingen
  - neutral, average and random (done)
  - demo's LaPlace smoothing
  - social filtering (Duine)
  - Duine's best
- remove "Semantic Web features" from data and compare accuracy
  - remove taxonomies above properties (done)
  - research review: content-based recommendation use of Semantic Web or equivalent features
  - features to consider and compare: any structure, typed properties, hierarchy
- make statistical algorithm and repeat above
- make dialog strategy emulator to measure efficiency
  - apply to various algorithms and data structures
  - test different strategies
- make algorithm for measuring significant of predicates
  - that is, which property types more strongly reflect interest, such as style
  - uses standard deviation for each value of a predicate
  - evaluate by measuring effect on accuracy and efficiency
- make algorithm for generating hierarchy
  - personalize based on: node interest, predicate significance
  - apply to: recommender, facet generation, tour