Recommender Systems
Based on Navigation Path Features

Wolfgang Gaul, Lars Schmidt-Thieme
{Wolfgang.Gaul,Lars.Schmidt-Thieme}@wiwi.uni-karlsruhe.de
Institut für Entscheidungstheorie und Unternehmensforschung
University of Karlsruhe, Germany

Abstract:
Several kinds of features of user navigation paths (e.g., subsets of the resources as nodes of the paths covered, subsequences of the sequences used for path description, path fragments constructed via combination of subsequences and wildcards) can be employed to build recommender systems designed for tasks as different as site personalization, cross-/up-selling, and navigation assistance. A vocabulary to describe different kinds of recommender systems and generic quality measures for system evaluation are formulated. The construction of specific recommender systems, especially systems based on frequent path features, is explained. In addition to the attempt to provide a formal framework for navigation path based recommender systems results on the performance of different types of suchs systems are reported.

1 Introduction

A recommender system is software that collects and aggregates information about site visitors (e.g., buying histories, products of interest, hints concerning desired/desirable search dimensions or other FAQ) and their actual navigational and buying behavior and returns recommendations (e.g., based on customer demographics and/or past behavior of the actual visitor and/or user patterns of top sellers with fields of interest similar to those of the actual contact). These recommendations have to be created in such a way that they are valuable for browsers/customers/visitors as well as for site owners. Nowadays, recommender systems are installed in more and more commercial sites to assist consumers in better/faster accessing useful information but also site owners in converting browsers to buyers, in stimulating cross- and up-sales, and in establishing customer loyalty as part of the activities to improve electronic customer care. — Recommender systems have been studied extensively since Resnick et al. (1994) who used the label *collaborative filtering*. An overview about applications of recommender systems in e-commerce can be found in Schafer et al. (1999, 2000) and an analysis of some recommendation algorithms in Breese et al. (1998) and Sarwar et al. (2000).

Beside such known approaches to designing recommender systems (based on buying behavior) a new type of recommender system has emerged that aims at helping surfers in navigating the web. At each step in the navigation process recommendations based on the up-to-now known navigation history are given concerning pages to visit next. Recommender systems based on navigation paths thereby add to the static hyperdocument linking by expanding it to dynamically linked hyperdocuments.

Recommender systems based on navigation paths are useful in e-commerce contexts as they try to make buying more pleasant for potential customers. As major parts of an e-commerce site can consist of product catalogs and the presentation of individual products, linking between such information can also be performed by traditional recommender systems. The real strength of recommender systems based on navigation paths becomes clearer in more or less unstructured collections of information, as found in, e.g., news groups, message boards, web directories, search engines and the like — provided that usage information within those collections can be gathered and joined.

First roots of recommender systems for paths can be found in an adaptive hypertext system of Stotts and Furuta (1991) that requires a special document reader which can be advised to modify (attributes of) links already coded in the documents with respect to usage behavior. The idea of developing recommender systems based on standard HTTP-servers and the information in their logfiles dates back to Yan et al. (1996), where a very simple clustering algorithm is used and the construction of recommender systems is described as *dynamic hypertext linking*. Perkowitz and Etzioni (1998) build recommender systems based on co-occurrence frequencies between resources and connected components of the usage graph and call them *adaptive web sites*. Mobasher (2001) has investigated three different approaches to compute recommendations by using sessions described as sets of pages visited together (including visit times). His first approach is based on association rules for sets, for a second alternative session clusters (computed via the k-means algorithm) are needed, the third approach uses resource clusters (computed by means of ARHP (association rule hypergraph partitioning)). — From the point of view of web server design, the problem of computing recommendations resembles the prefetching problem (see e.g., Bestavros (1996)), i.e., the prediction of the pages requested next by the active users to speed up server operation and thereby lower waiting times; the prefetching problem is easier in the respect that pages close to the recommendation point are perfect choices for predicted requests (as they are expected to rest a shorter time in the cache). — Bodner and Chignell (1999) tackle the problem from the point of view of text retrieval: they exploit the reference texts of visited links and keep track of a list of relevant key words that is fed into a search engine; the results of the search are linked from the keywords found in the active document. Joachims et al. (1995) and Lieberman (1995) present WebWatcher and Letizia, two agents for web browsing that are capable of giving recommendations depending on the users' searching behavior so far; WebWatcher uses similarities in the link structure to identify related documents, Letizia gathers additional data about user behavior (as bookmarking and usage of documents on different servers not available on server-side). Fu et al. (2000) propose another agent that collects usage information of different users in a central repository and computes recommendations from sets of pages visited frequently together by means of association rules.

2 Prerequisites for recommender system evaluation

Let R be an arbitrary set (the set of resources of a website that correspond to the nodes of the link graph of this site). The set $R^* := \bigcup_{n\in\mathbb{N}} R^n$ of all tuples of R is called the set of sequences of R and serves as basis to model user paths.

A sequence $p = (p_1, ..., p_{|p|}) \in R^*$ describes a path as sequence of resources of R, its length is denoted as $|p|$. Sometimes, we replace the tuple-notation by just putting the corresponding resources one after the other, i.e., $p_1 p_2 \cdots p_{|p|}$.

From a mathematical point of view, a *recommender system based on navigation paths* is a map

$$r : R^* \longrightarrow \mathcal{P}(R) \tag{1}$$

(where $\mathcal{P}(R)$ denotes the powerset of R) and the set $r(p)$ is called *recommendation set* for $p \in R^*$.

Starting point for an evaluation of recommender systems is a (multi)set of paths $\mathcal{S}$. Each path $p \in \mathcal{S}$ can be split at position $i \in \{1, ..., |p| - 1\}$ in a *history* $h_i(p) := (p_1, ..., p_i)$ and a *future* $f_i(p) := (p_{i+1}, ..., p_{|p|})$. p_i is called *recommendation point*.

Now, a general definition for a *recommendation quality measure* can be given by

$$\begin{array}{rcl} q : R_1 \times R_2 \times R_3 & \longrightarrow & \mathbb{R}_0^+ \\ (h, f, r) & \mapsto & q(h, f, r) \end{array} \tag{2}$$

where R_1 describes the history space, R_2 the future space, and R_3 the space of (sets of) recommendations $r(h)$ derived from $h \in R_1$. Various choices of R_1, R_2, and R_3 are possible. We will restrict to $R_1 = R_2 = R^*$ and $R_3 = \mathcal{P}(R)$, in the following.

$q(h, f, r)$ measures the quality of recommendations (e.g., by choosing $h = h_i(p)$ and comparing $r = r(h_i(p))$ with $f = f_i(p)$ for a path p).

Simple examples of recommendation quality measures are

$$q(h, f, r) := |\{y \in r | y \text{ occurs in } f\}| \tag{3}$$

which is just the number of recommended resources that also occur in f, or

$$q(h, f, r) := \sum_{y \in r(h)} \bar{q}(h_{|h|}, f, y) \tag{4}$$

where $\bar{q} : R \times R^* \times R \to \mathbb{R}_0^+$ describes a measure that depends only on the recommendation point $h_{|h|}$ and evaluates the degree of conformity between f and a single recommendation $y \in r(h)$, i.e., the quality measure does not take into consideration any compound effects as, e.g., preference of resources concentrated in a particular region of the site over those scattered all over the whole site.

Recommendation quality can take into consideration the distance between history resources and recommended resources (measured with the help of the underlying site graph structure or, alternatively, defined as minimal number of resources between recommendation point and recommended resource in the actual future of a path), e.g., for $x, y \in R$ and $f \in R^*$ (e.g., with $x = p_i, y \in r(h_i(p))$, and $f = f_i(p)$ for a path $p \in R^*$) one can define

$$\bar{q}(x, f, y) := \begin{cases} u(\operatorname{dist}(x, y)) & \text{, if } y \neq x \\ & \quad \text{occurs in } f \\ 0 & \text{, otherwise} \end{cases} \tag{5}$$

where dist denotes an appropriate *distance function* and u measures the *utility* assigned to the distance between pairs of resources. The meaning is that resources in the direct neighborhood of a recommendation point are easier to find (and, thus, to recommend) than adequate resources far away. Examples for utility functions are

$$u : \mathbb{R}_0^+ \longrightarrow \mathbb{R}_0^+ \tag{6}$$

$$d \mapsto \begin{cases} 1 & \text{hit count} \\ d & \text{linear scale} \\ \log d + 1 & \text{log. scale} \\ (d - d_0 + 1)\delta_{[d_o, d_1]}(d) & \text{window effect} \end{cases}$$

$$\text{with } \delta_{[d_o, d_1]}(d) := \begin{cases} 1, & d \in [d_0, d_1], \\ 0, & \text{otherwise} \end{cases}$$

Up to now, recommendation quality measures as depicted in (3), (4) are restricted to a single navigation path but, of course, for a given recommender system r, a recommendation quality measure q, and an underlying (multi)set $\mathcal{S}$ of navigation paths, one can define, e.g.,

$$Q_r^{\text{raw}}(\mathcal{S}) := \sum_{p \in \mathcal{S}} \sum_{i=1}^{|p|-1} q(h_i(p), f_i(p), r(h_i(p)))$$

as *raw recommendation score for* r *relative to* $\mathcal{S}$. Let

$$Q_{\max}^{\text{raw}}(\mathcal{S}) := \max_r Q_r^{\text{raw}}(\mathcal{S})$$

be the (theoretically) *maximal recommendation score* (relative to a given quality measure q); see section 3 for a simple method to compute $Q_{\max}^{\text{raw}}(\mathcal{S})$ for a given test set $\mathcal{S}$. Then, one can define

$$Q_r(\mathcal{S}) := Q_r^{\text{raw}}(\mathcal{S}) / Q_{\max}^{\text{raw}}(\mathcal{S})$$

as *normalized recommendation score*, which is a useful characteristic number for the comparison of the performance of a recommender system on different test sets or of different recommender systems on the same (multi)set $\mathcal{S}$.

Now, the problem to find an optimal recommender system can be formalized as follows: given a quality measure q construct a recommender system r on the basis of information from a training set S^{train} of paths so that the raw recommendation score of r on a test set S^{test} of paths (not used for building the recommendation system) is maximal.

For the simple recommendation quality measure (3) that just counts the number of conformities between resources of r and f, apparently, the optimal recommender system is the system that simply recommends all resources for any given history: for sure, this recommendation set will hit all resources in the future and be of no interest whatsoever. Two kinds of modifications are possible to make the problem more interesting:

1. *Modify the recommendation quality measure.* For instance, one may think of counting the number of hits relative to the number of given recommendations. While optimal recommendations for the simple quality measure (3) consist of large recommendation sets, optimal recommendations for the relative number of hitting recommendations have very small recommendation sets: in almost all cases for each history only the one resource with highest follow-up probability is selected and all other resources with lesser but perhaps also high probabilities are discarded.

2. *Restrict the space of possible recommender systems by imposing additional constraints.* A restriction that always ever is sensible in practice is to allow only recommendation sets of a given maximal size (i.e., $|r(h)| \leq n$ for all $h \in R^*$ and a given $n \in \mathbb{N}$). This constraint forces a restriction to the best n recommendations; in practice, n will be a small number, say 3 up to 5, of recommendations that users may be willing to look at. — Thus, one may specialize the problem of finding an optimal recommender system to the construction of an optimal one among a predefined class of recommender systems (e.g., those with at most a given number of recommendations per history).

For paths in a (sparsely linked) graph the computation of recommendations with respect to a quality measure based on hits (disregarding distances of recommended resources) will — in most cases — still result in a set of resources directly linked to the recommendation point. Here, we use the idea of Mobasher (2001) to weight resources farther apart higher by choosing an appropriate quality measure depending on the distances of the recommended resources (another, simpler distance sensitive quality function can be found in Cooley et al. (1999)). Of course, utility functions — when used for modeling different problems — may depend on other parameters (as, e.g., explicit or implicit ratings) besides distance as well.

3 Different types of recommender systems

As the preceding discussion has shown, a variety of optimality criteria for recommender systems can be designed on the basis of appropriate choices of q and optional restrictions for r. Here, we start with some obvious possibilities to typecast recommender systems.

As normally the number of collected navigation paths is very large compared to the number of resources of the underlying site, we may break down the global problem of finding optimal recommender systems for a whole site into a set of smaller subproblems of constructing optimal systems for each single resource. We split R^* for $x \in R$ into spaces $R^*_x := \{p \in R^* \mid p_{|p|} = x\}$ that consist only of sequences with x at the last position and call

$$r_x : R^*_x \longrightarrow \mathcal{P}(R) \tag{7}$$

a *local recommender system at resource* x in contrast to the *global* version described by (1). Accordingly, the training set $\mathcal{S}^{\text{train}}$ for the global system is transformed into training sets $\mathcal{S}^{\text{train}}_x \subseteq R^*_x \times R^*$ for the local systems that consist of all navigation paths p split at x (as recommendation point, if p contains x); in the case that a resource x appears k times in a path $p \in \mathcal{S}^{\text{train}}$, then $\mathcal{S}^{\text{train}}_x$ contains k replications of p split at each occurrence of recommendation point x. — Once that optimal local systems for all $x \in R$ have been found, they can be pieced together to a global system $r : R^* \to \mathcal{P}(R)$ by delegating the recommendation task to the appropriate local model, i.e., $r(h) := r_{h_{|h|}}(h)$, as there is no dependency of the recommendations given at one recommendation point upon those given at another recommendation point.

We further distinguish between *static* and *dynamic recommender systems*: static recommender systems do not take into account the former navigation histories of users and provide a static set of recommendations for all visitors, while dynamic recommender systems may depend on the histories and provide different recommendation sets for users with different histories. Dynamic systems may be build by first partition the histories of the training set S^{train} and, then, compute a static system for each class.

Training sets for static (local) recommender systems can be described as (multi)sets of futures $F_x \subseteq R^*$, extracted from $\mathcal{S}_x^{\text{train}}$ via $F_x := \{f \in R^* \mid \exists h \in R_x^* : (h, f) \in \mathcal{S}_x^{\text{train}}\}$. A simple recommender system just counts frequencies of resources $y \in R$ in the future paths via

$$\text{freq}(y) := |\{f \in F_x \mid y \text{ occurs in } f\}|$$

and recommends the n most frequent ones. Up to now, no utility functions have been taken into consideration. To do so, one has to sum up the utility values for all resources in the future paths, e.g., for the distance sensitive utility functions within (6) one computes the weighted frequencies

$$\text{wfreq}(y) := \sum_{f \in F_x} \bar{q}(x, f, y)$$

with $\bar{q}$ as given in (5) and, again, selects the n highest valued follow-up resources. Note that the computation of the weighted frequencies depends on the recommendation point x, but the recommendation set itself does not, thus, a static recommender system is generated. By construction this is the optimal system among all static recommender systems at x.

Dynamic (local) recommender systems make use of a history partition $\mathcal{C} = \{C_1, \ldots, C_m\}$ of (a superset of) all histories $h \in R^*$ in the test set (where $m \in \mathbb{N}$ is the number of classes). The test set $\mathcal{S}^{\text{test}}$ can be partitioned into test sets $\mathcal{S}^{\text{test}}|_C := \{(h, f) \in \mathcal{S}^{\text{test}} \mid h \in C\}$ for each class $C \in \mathcal{C}$ and a static recommender system can be build for each such class.

While the use of ordinary partitions is straightforward, fuzzy partitions need additional information about predicted utility values for recommendations given by the static recommender systems for each class. Now, let $\mathcal{C} = \{w_1, \ldots, w_m\}$ be a fuzzy partition of the histories, i.e., all $w \in \mathcal{C}$ are functions $w : R^* \to [0, 1]$ with $\sum_{w \in \mathcal{C}} w(h) = 1$ for all $h \in R^*$. $w(h)$ is called *weight of h in class w*. The static recommender systems for each class w have to provide a predicted utility value for each recommendation, i.e., they are maps

$$r_w : R^* \to [0, 1]^R$$

with recommendation set

$$\text{rec}(h) := \{(y, v) \in R \times [0, 1] \mid v = r_w(h)(y) > 0\}$$

To yield recommendations for a given history h a dynamic recommender system using fuzzy partitions, first, computes the weights of h for all classes w, second, computes for all classes w with $w(h) > 0$ the (extended) recommendation set $r_w(h)$, third, adjusts the predicted utility values by the weight $w(h)$ for the class the recommendations stem from, and, then, chooses the n recommendations with highest (adjusted) predicted utility.

A trivial example for a dynamic recommender system is the one build upon the singleton partition $\mathcal{C} = \{\{h\} \mid \exists f \in R^* : (h, f) \in \mathcal{S}^{\text{train}}\}$, that we call *recommender system based on finest history partition.* As each difference between histories results in different classes, this recommender system extremely suffers from overfitting and therefore performs very poorly on test sets. But beside the fact that the recommender system based on finest history partition is a trivial example for a dynamic system, it can be useful for the computation of an upper bound for the raw recommendation score (that can be achieved by any recommender system on the underlying test set), if it is trained by the test set (!) itself, i.e., the raw recommendation score of the recommender system based on finest history partition for a test set $\mathcal{S}^{\text{test}}$ is the *theoretically maximal raw recommendation score* $Q_{\max}^{\text{raw}}(\mathcal{S}^{\text{test}})$.

The computation of $Q_{\max}^{\text{raw}}$ requires the building of a huge amount of static recommender systems (one for each history), that either may consume a considerable large amount of memory or forces several iterations over the test set database, i.e., can not be done efficiently. If runtime is an issue, a more pessimistic upper bound can be computed by choosing the best recommendation for each very history in the test set in a single loop over the test set database. Note that this may result in different recommendation sets for the very same history and, thus, may never be achieved by a real recommender system. But for test sets with many different histories and a decent quality function this bound is close enough to $Q_{\max}^{\text{raw}}$. As in our experiments in section 6 runtime has not been considered, we use exact values for $Q_{\max}^{\text{raw}}$. — Please note that $Q_{\max}^{\text{raw}}$ is used to compute normalized recommendation scores. As the main purpose of normalized scores is the comparison of recommender systems on different data sets or of different loci of a localized system, its main applications are in research oriented contexts. For tracking recommender system performance in operational contexts raw scores may be used.

Obviously, both, localization (i.e., the determination of local systems) and the usage of history partitions can be viewed as application of a clustering technique to the histories of paths in $\mathcal{S}$ that — based on an adequate similarity criterion for navigation paths — reduces the global problem to the handling of subproblems described by more homogeneous sub(multi)sets $\mathcal{S}|_C$, where C denotes the class under consideration of the resulting (possibly fuzzy or overlapping) classification. Of course, one can combine these possibilities and apply history partitions to $\mathcal{S}_x$, resulting in $\mathcal{S}_x|_C$, or split $\mathcal{S}|_C$ into subsets of navigation paths with same recommendation point x, resulting in $(\mathcal{S}|_C)_x$. Note, that while localization uses very intuitive classes that do not have to be computed, the hard part of history partitions is the computation of the partition itself. Therefore, one first applies localization and afterwards computes history partitions for each local system. An overview of the architecture of such a complex system is given in figure 1.

4 Path features

Paths that users have taken on a site belong to the most valuable information that can be gained. But paths as sequences of resources of different lengths are complex objects which are not that easy to compare and to use in data mining algorithms. Thus, one is interested in determining sets of simpler features for path description (*feature extraction*).

A *substructure space* of R^* is defined as pair $(\mathcal{A}, \preceq)$ of a set $\mathcal{A}$ and a relation $\preceq$ on $\mathcal{A} \times R^*$ where $a \in \mathcal{A}$ is called *substructure of* $p \in R^*$ if $a \preceq p$.

$$\delta_a : \quad R^* \to \{0,1\} \qquad p \mapsto \begin{cases} 1, & \text{if } a \preceq p \\ 0, & \text{otherwise} \end{cases}$$

is called *indicator function of substructure* a. Examples of substructures are:

1. *sets* $(\mathcal{P}(R), \subseteq)$ of resources, where a set of resources $a \in \mathcal{P}(R)$ is defined to be a substructure of a path $p \in R^*$ if all resources $x \in a$ occur in path p,

2. *sequences* $(R^*, \leq_{ct})$, where a sequence $a \in R^*$ is defined to be a substructure of a path p if it is a contiguous subsequence, i.e., it exists $i_0 \in \{0, \ldots, |p| - |a|\}$ with $a_i = p_{i_0+i}$ for all $i = 1, \ldots, |a|$,

3. *generalized sequences* $((R \cup \{*\})^*, \leq_{gen})$, i.e., sequences consisting of elements of R and an additional symbol $*$ used as wildcard, where a generalized sequence $a \in (R \cup \{*\})^*$ is defined to be a substructure of a path p if it is a generalization of a (contiguous) subsequence of p and *generalization* means that arbitrary parts of the sequence may be replaced by wildcards (see Gaul and Schmidt-Thieme (2000) for an exact definition),

4. *simple generalized sequences* $(R^*, \leq_{nct})$, where a (simple generalized) sequence $a \in R^*$ is defined to be a substructure of a path p if it is a noncontiguous subsequence with the following meaning: It exists $j : \{1, \ldots, |a|\} \to \{1, \ldots, |p|\}$ strictly increasing with $a_i = p_{j(i)}$ for all $i = 1, \ldots, |a|$, i.e., in the context of generalized sequences, if $a_1 * a_2 * \cdots * a_{|a|} \leq_{gen} p$.

The space of simple generalized sequences can be viewed as a subspace of the space of generalized sequences where a wildcard is interspersed between each two resources. Notice that in practical applications only generalized sequences without a wildcard at the first and/or last position (i.e., $a \in (R \cup \{*\})^*$ with $a_1, a_{|a|} \in R$) are of interest. These sequences are called *path fragments*.

For any substructure space $\mathcal{A}$ the symbol $\emptyset$ describes the *empty substructure* (i.e., the empty set or the empty sequence, respectively) and $|a|$ the *substructure complexity* of $a \in \mathcal{A}$ defined as cardinality (for sets) or length (for sequences).

Now, we can define a *path feature* to be a pair (Φ, φ), where Φ is an arbitrary set called *feature space* and $\varphi : R^* \to \Phi$ the *feature map* mapping paths to features. For a path $p \in R^*$ we call $\varphi(p)$ the *φ-feature of* p.

Trivial examples for path features are its length $(\varphi : R^* \to \mathbb{N}, p \mapsto |p|)$ and its entry point $(\varphi : R^* \to R, p \mapsto p_1)$. More interesting features are obtainable via substructures.

From an arbitrary substructure space $(\mathcal{A}, \preceq)$

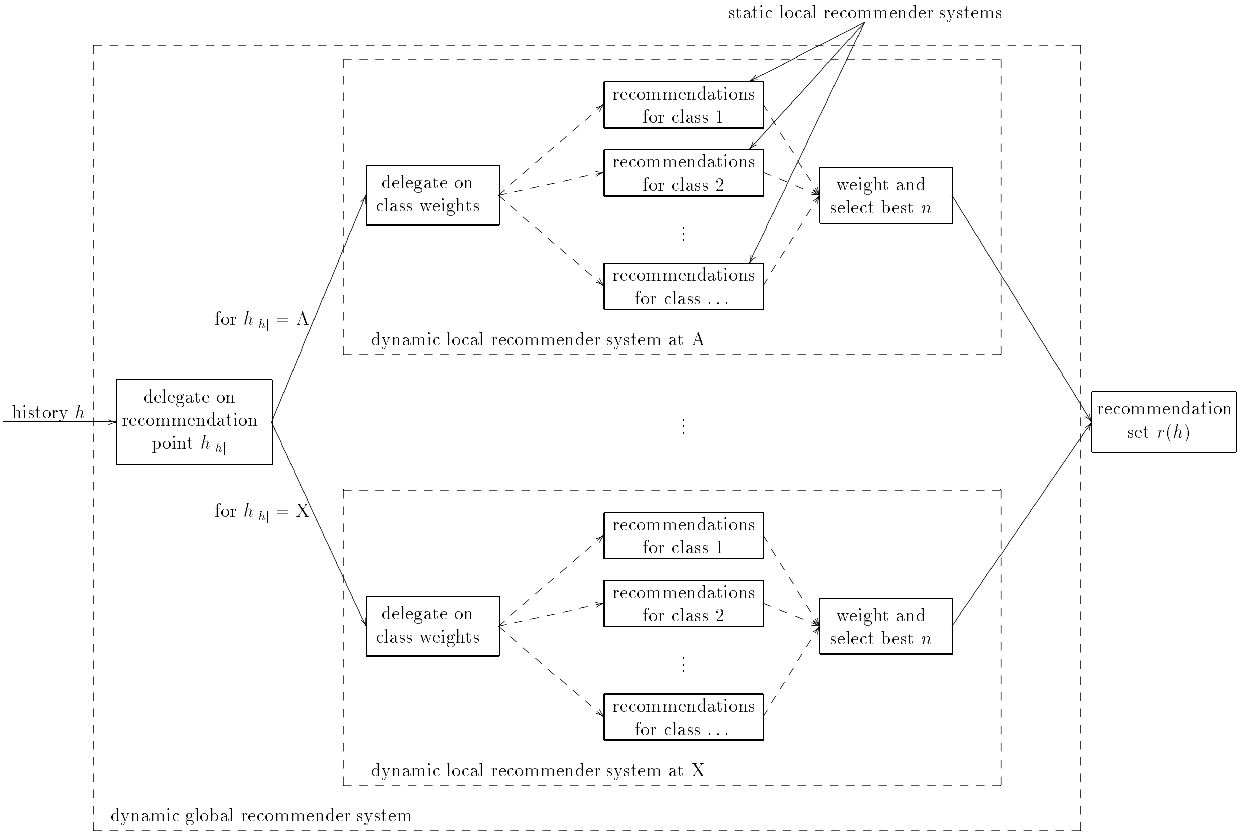


Figure 1: Architecture of a dynamic global recommender system

we derive its associated path feature

$$\begin{array}{rrl}\varphi: & R^* \to & \{0,1\}^{\mathcal{A}} \\ & p \mapsto & \left(\begin{array}{rrl}\delta: & \mathcal{A} \to & \{0,1\} \\ & a \mapsto & \delta_a(p)\end{array}\right)\end{array}$$

i.e., a feature space that — for every path p — contains a binary vector indicating whether an element $a \in \mathcal{A}$ is a substructure of p or not.

Feature spaces based on substructures turn out to have the disadvantage of high dimensionality: the feature space build from subsets has dimension $2^{|R|}$, the one build from finite sequences (if subsequences are restricted to length n) has dimension $\sum_{i=1}^{n} |R|^i$. Therefore — given an underlying (multi)set of navigation paths $\mathcal{S}$ that has to be analyzed — one is looking for interesting subsets of substructures that result into a smaller number of dimensions but still carries as much information as possible for a description of the objects of (multi)set $\mathcal{S}$ (*feature selection*). We call a dimension *sparse with respect to* $\mathcal{S}$ if the corresponding entry in the binary vector is zero for almost all paths of (multi)set $\mathcal{S}$. In applications, one often can drop a vast number of sparse dimensions and restrict to those dimensions for which the percentage of non-zero entries in the binary vectors exceeds a lower bound.

Dependent on this bound called minsup frequent substructures of the paths of $\mathcal{S}$ can be determined beforehand. For a substructure $a \in \mathcal{A}$ one defines its relative frequency

$$\sup_{\mathcal{S}}(a) := \frac{|\{p \in \mathcal{S} \mid a \preceq p\}|}{|\mathcal{S}|}$$

as *support of* a *in* $\mathcal{S}$. The task to compute all frequent substructures, i.e., the set $\Phi_{(\mathcal{S},\mathrm{minsup})} := \{a \in \mathcal{A} \mid \sup_{\mathcal{S}}(a) \geq \mathrm{minsup}\}$ of all substructures with at least a given minimum support $\mathrm{minsup} \in \mathbb{R}_0^+$, is well known and accomplished by the Apriori algorithm for sets (Agrawal and Srikant (1994)), sequences (Agrawal and Srikant (1995)) and generalized sequences (Gaul and Schmidt-Thieme (2000)), respectively. Building the feature space from the frequent substructures $\Phi_{(\mathcal{S},\text{minsup})}$ only instead of using all substructures of $\mathcal{A}$ can reduce the dimensionality dramatically (depending on the minimum support and the structure of $\mathcal{S}$, of course). We call this feature space *path features based on frequent substructures* in general, and, in particular, *path features based on frequent subsets, subsequences, (simple) generalized sequences or path fragments*, etc.

5 Recommender systems based on frequent substructures of navigation histories

Frequent substructures of the histories at a resource x can be used to build fuzzy partitions for dynamic local recommender systems. For a local training set $\mathcal{S}_x^{\text{train}} \subseteq R_x^* \times R^*$ at a recommendation point $x \in R$ let $H_x := \{h \in R_x^* \,|\, \exists f \in R^* : (h, f) \in \mathcal{S}_x^{\text{train}}\}$ be the (multi)set of corresponding histories. Let $\Phi_x := \Phi_{(H_x,\text{minsup})}$ denote the set of frequent substructures of H_x. For each frequent substructure $a \in \Phi_x$ we build a class represented by the weight function $w_a : R_x^* \to [0, 1]$. Note, that the empty substructure $\emptyset$ occurs in every history by definition and, thus, is frequent in any case; it serves as class for histories without any frequent (non-empty) substructures. For $h \in R_x^*$ let $\Phi(h)$ denote the set of frequent substructures occurring in h (including $\emptyset$). The class weight functions w_a are then defined as

$$w_a(h) := \begin{cases} \frac{\nu(a)}{\sum_{b\in\Phi(h)} \nu(b)} & , \quad \text{for } a \in \Phi(h) \\ 0 & , \quad \text{otherwise} \end{cases}$$

where $\nu : \Phi_x \to \mathbb{R}_0^+$ is a function that measures how specific or interesting a frequent substructure is. ν might be set constantly to 1 (not making any differences between different substructures and, thus, weighting them all equally), it might consider the frequency of a substructure and be set to the inverse of the support $\frac{1}{\sup_{H_x}(a)}$, or it might consider structural information of a substructure and be set to the complexity $|a|$ of a frequent substructure. We will use the second variant in our experiment in section 6.

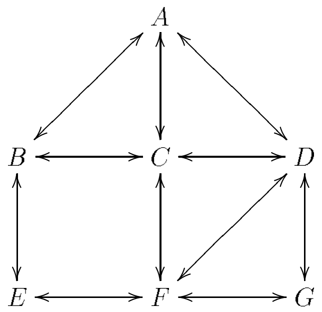
Alternatively, one can assign histories only to classes representing maximal substructures. Let $\Phi^{\max}(h)$ be the set of maximal frequent substructures of h (especially $\{\emptyset\}$, if there are no frequent substructures at all) and compute w_a with $\Phi^{\max}(h)$ instead of $\Phi(h)$.

Partitions based on frequent structures tend to become rather large for small minimum support values. A pruning step can decrease the number of interesting frequent substructures before the partition is formed and static recommender systems for each class are build. For any frequent substructure $a \in \Phi_x$, all more specific substructures $b \in \Phi_x$ (e.g., supersets, supersequences, etc.), that have the same support, can be removed: as identical support values will lead to the same class weights and as both frequent substructures a and b have the same training sets, i.e., lead to the same static recommender system, they would create the very same recommendations, and, consequently, the system for b is superfluous. The reader acquainted with the notion of closed subsets in the theory of set based association rules will see the analogy between this pruning step and the search just for closed subsets (see, e.g., Zaki and Hsiao (1999)); but note, that while closed subsets are the most specific ones among all subsets with same support, here — by contrast — we keep the most general ones among all substructures with same support.

6 Examples and experiments

After the theoretical outline we present a series of small examples to illustrate the capabilities and shortcomings of the different kinds of recommender systems based on navigation paths before an experimental evaluation is described with the intention to provide a feeling for the advantages obtainable by application of path features.

Figure 2 shows the link graph of a small site of seven resources $R = \{\text{A,B,C,D,E,F,G}\}$ and three examples of navigation paths that are all analyzed at recommendation point C. The paths



a) site graph

utility	u_1:	1	2	3	4
	u_2:	1	1.7	2.1	2.4
path	history	future			
p_I:	AC	D	F	G	
p_{II}:	ABC	F	E	B	
p_{III}:	AC	F	D	G	D
p_{IV}:	ABC	D	G	F	E

b) paths of example 1

utility	u_1:	1	2	3	4
	u_2:	1	1.7	2.1	2.4
path	history	future			
p_V:	BAC	D	F	G	
p_{VI}:	ABC	F	E	B	
p_{VII}:	BAC	F	D	G	D
p_{VIII}:	ABC	D	G	F	E

c) paths of example 2

utility	u_1:	1	2	3	4
	u_2:	1	1.7	2.1	2.4
path	history	future			
p_{IX}:	BAC	D	F	G	
p_X:	ADFEBC	F	E	B	
p_{XI}:	BAC	F	D	G	D
p_{XII}:	ABC	D	G	F	E

d) paths of example 3

Figure 2: Site graph and sample paths of examples 1, 2, and 3

differ with respect to their histories while their futures remain unchanged in the different examples. For simplicity, at most a single recommendation ($n = 1$) is provided for comparisons.

We start with example 1 and assume that the underlying system is static and that $\mathcal{S} = \{p_I, p_{II}, p_{III}, p_{IV}\}$ is the (multi)set of navigation paths under consideration. Without using utility functions one just counts occurrences of resources following C: F is the only resource occurring in all four futures and, thus, a recommender system based on mere frequencies would recommend F. Now, we add a utility distance $u_1(d) := d$ or $u_2(d) := \ln d + 1$, respectively; the utility values for the resources following C are given in the upper two lines of the tables. Using utility sums, the recommender system based on weighted frequencies now computes the u_1-utility sum 7 for F and the u_1-utility sum 8 for G or the u_2-utility sum 5.8 for F and the u_2-utility sum 5.9 for G and — in both cases — would recommend G instead of F. If a resource occurs more than once in the future path (as D in path p_{III}) only the first occurrence (shortest empirical distance) is counted.

The recommender system based on finest history partition recovers two history classes {A} and {AB}. For class {A} the recommendation G with u_1-utility sum 6 (or u_2-utility sum 4.2) is computed, for class {AB} recommendation E, also with u_1-utility sum 6 (or u_2-utility sum 4.1), is found, resulting in a theoretically maximal possible recommendation score of 12 for u_1 (or 8.3 for u_2). Thus, the normalized recommendation score of the recommender system based on mere frequencies is $7/12 = 0.58$ for u_1 (or $5.8/8.3 = 0.70$ for u_2) while for the system based on weighted frequencies it is $8/12 = 0.66$ for u_1-utility summation (or $5.9/8.3 = 0.71$ for u_2-utility summation). This part of our small example was designed to show that the incorporation of utility distances (other utility functions are thinkable) leads to the recommendation of resource G farther apart from recommendation point C instead of resource F directly linked to C.

Now we apply the recommender system based on frequent subsets with minsup $= 0.5$ to example 1. The two frequent subsets {A} with support 1 and {A,B} with support 0.5 are found and lead to a fuzzy partition with 3 classes. For the static recommender system based on weighted frequencies that takes into account all histories containing {A} all four paths are used and consequently G is computed as best recommendation (the same as for histories containing $\emptyset$), for the same system that, now, takes into account all histories containing {A,B} only the paths p_{II} and p_{IV} are used, so that E is computed as best recommendation. Thus, the recommender system based on frequent subsets achieves a recommendation score of 1.0.

Next, we take example 2 which is a slight modification of example 1: paths p_V and p_{VII} now start with BA (instead of A as p_I and p_{III} did). Consequently, the recommender system based on frequent subsets recovers only one class of histories {A,B} and, thus, will achieve the recommendation score of 0.70 only. The recommender system based on frequent subsequences still can distinguish between subsequences AB and BA of the histories and achieves a recommendation score of 1.0 again.

If we, now, look at example 3 in which only the history of path p_X was changed compared to path p_{VI} of example 2 by inserting a small deviation DFE between A and BC in the history, the extraction of frequent contiguous subsequences would result in BA only and a distinction between the two classes found in example 2 would not be possible. But the recommender system based on simple generalized subsequences (or path fragments) is able to separate the two frequent simple generalized sequences $A \star B$ and BA in the histories of example 3 and can, thus, give better recommendations.

Finally, our findings have been checked on larger (multi)sets. Table 1 shows the result of an experimental evaluation of some of the different recommender systems based on navigation paths as described earlier. We modeled four different classes of users navigating a small site of 20 resources (A-T) with several crossings. On the basis of an abstract description of user classes (percentage of total users, templates of navigation behavior, distributions of variations etc.) we created a database of 10.000 navigation paths. 90% of the paths were used as training set $\mathcal{S}^{\text{train}}$ to build the models, the remaining 10% of the paths as test set $\mathcal{S}^{\text{test}}$ to evaluate the quality

resources		A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T
weights		0.10	0.15	0.18	0.07	0.06	0.05	0.05	0.07	0.08	0.05	0.05	0.05	0.05	0.05	0.04	0.05	0.05	0.05	0.06	0.05
	global	local																			
static recommender systems:																					
freq	0.41	0.21	0.31	0.57	0.42	0.36	0.58	0.16	0.67	0.58	0.51	0.14	0.18	0.52	0.57	0.32	0.50	0.15	0.53	0.17	0.50
wfreq	0.64	0.88	0.69	0.57	0.66	0.73	0.58	0.58	0.67	0.64	0.65	0.67	0.61	0.52	0.60	0.52	0.56	0.55	0.61	0.67	0.54
dynamic recommender systems based on frequent substructures:																					
sets	0.65	0.87	0.71	0.55	0.66	0.73	0.58	0.61	0.67	0.64	0.64	0.69	0.55	0.58	0.69	0.62	0.56	0.64	0.64	0.69	0.56
seq	0.69	0.88	0.71	0.58	0.68	0.78	0.73	0.71	0.67	0.74	0.66	0.71	0.63	0.65	0.74	0.59	0.68	0.67	0.62	0.71	0.63
sgseq	0.76	0.96	0.77	0.63	0.68	0.77	0.80	0.82	0.73	0.78	0.76	0.76	0.73	0.68	0.82	0.70	0.73	0.79	0.79	0.83	0.73
frag	0.77	0.96	0.77	0.66	0.67	0.78	0.80	0.83	0.73	0.79	0.80	0.77	0.75	0.69	0.81	0.73	0.74	0.80	0.79	0.86	0.73

Table 1: Experimental evaluation of different recommender systems on a small site with 20 resources (A–T). The row *weights* describes the percentage of occurrences of each resource in the path data. The column *global* gives the global recommendation quality for each recommender system. Furthermore, for the *local* parts of each recommender system with resource A up to resource T as recommendation point the quality is given. The following recommender systems (r.s.) are evaluated: r.s. based on frequencies of resources following the (actual) recommendation point (*freq*), r.s. based on weighted frequencies (*wfreq*), r.s. based on frequent subsets (*sets*), r.s. based on frequent subsequences (*seq*), r.s. based on frequent simple generalized subsequences (*sgseq*), and r.s. based on frequent path fragments (*frag*); all recommender systems based on frequent substructures used a minimum support of 0.2.

of the models and compute the recommendation scores. We used the utility function u_1 and allowed only a single recommendation at each resource ($n = 1$). As expected, the use of simple frequencies (model *freq*) resulted in a low global recommendation score of 0.41, for the local versions the quality dropped below 0.15 at some of the resources. Using this as baseline, the model based on weighted frequencies (*wfreq*) that takes into consideration the special form of the utility function achieves an overall improvement in global quality of over 50% (global recommendation score 0.64). As we put strong sequential effects in the navigation patterns of the different user classes, dynamic recommendations based on frequent sets (*sets*) do not result in a better global score (0.65), but using sequences (*seq*) or even better simple generalized sequences (*sgseq*) or path fragments (*frag*) further improvements of the global recommendation quality were possible (by another 32% with respect to the baseline score (see the global recommendation scores of 0.69, 0.76, and 0.77, respectively)). In all cases, the frequent substructures for the dynamic models have been extracted with a minimum support of 0.2 (the smallest expected user segment size for the data set).

The local quality scores show that not in all cases (i.e., at all recommendation points) the same ranking as for the global recommendation quality values can be observed. This is due to the fact that at some recommendation points sequential effects were not strong enough in the data so that local recommender systems based on path substructures could not take advantage of some of the path features, and/or due to the choice of the same minimum support for all local recommender systems, that is responsible for small overfitting effects in some of the local systems.

Of course, findings depend on the structure of the user segments. If only few sequential effects are in the data, the more complex models can not show their strengths and give similar results than the other models. In another experiment with 10.000 users in 10 segments (of size 10% each) on a small site with 100 resources, where crossings of navigation paths occurred by chance only, the global scores of table 2 were yielded (we omit the values of the local systems).

freq	wfreq	sets	seq	sgseq	frag
0.27	0.37	0.40	0.40	0.40	0.41

Table 2: Experimental evaluation of different recommender systems on a small site with 100 resources. and only few sequential effects in the data

7 Outlook

A framework for recommender systems based on navigation paths has been presented and the influence of different path features on recommendation quality considerations was theoretically discussed and empirically demonstrated. We developed a generic method to measure the quality of recommender systems in terms of a the (normalized) recommendation score, so that different systems can easily be compared, and gave examples for recommender systems based on navigation paths that made use of frequent substructures in the path histories.

Future work should address questions as pruning based on substructure partitions, automatically finding optimal support values, as well as comparing our results to those of history partitions obtained by approaches different from frequent substructures. Beside theoretical work on mathematical modeling of recommender systems an empirical evaluation of how they are used (and liked) by site visitors is one of the urgent questions in the field.

References

Agrawal, R. & Srikant, R. (1994). Fast Algorithms for Mining Association Rules. In Bocca, J.B., Jarke, M., & Zaniolo, C. (Eds.), *Proceedings of the 20th International Conference on Very Large Data Bases (VLDB'94),* September 12-15, 1994 (pp. 487–499), Santiago de Chile, Morgan Kaufmann, Chile.

Agrawal, R. & Srikant, R. (1995). Mining Sequential Patterns. In Yu, P.S. & Chen, A.L.P. (Eds.), *Proceedings of the Eleventh International Conference on Data Engineering,* March 6-10, 1995 Taipei, Taiwan, IEEE Computer Society, pp. 3–14.

Bestavros, A (1996): Speculative Data Dissemination and Service to Reduce Server Load, Network Traffic and Service Time. In *Proceedings IEEE Conference on Data Engineering (ICDE'96)*, pp. 180–189.

Bodner, R.C. & Chignell, M.H. (1999). ClickIR: Text Retrieval Using a Dynamic Hypertext Interface. In *Proceedings of the Seventh Text Retrieval Conference (TREC-7),* Gaithersburg, Maryland.

Breese, J.S., Heckerman, D. & Kadie, C. (1998). Empirical Analysis of Predictive Algorithms for Collaborative Filtering. In *Proceedings of the Fourtheenth Conference on Uncertainty in Artificial Intelligence,* Madison, WI, July, 1998.

Cooley, R., Mobasher, B., and Srivastava, J. (1999): Data Preparation for Mining World Wide Web Browsing Patterns. In *Knowledge and Information Systems 1/1 (1999),* pp. 5–32.

Fu, X., Budzik, J.,& Hammond, K.J. (2000). Mining Navigation History for Recommendation. In *Proceedings of the 2000 International Conference on Intelligent User Interfaces,* New Orleans, LA, January 2000, pp. 106–112.

Gaul, W. & Schmidt-Thieme, L. (2000). Mining web navigation path fragments. Proceedings of the WEBKDD'2000 workshop, Boston, 2000.

Joachims, T., Mitchell, T., Freitag, D.,& Armstrong, R. (1995). WebWatcher: Machine Learning and Hypertext. In Morik, K.,& Herrmann, J. (Eds.), *GI Fachgruppentreffen Maschinelles Lernen,* University of Dortmund, August 1995.

Lieberman, H. (1995). Letizia: An Agent That Assists Web Browsing. In *1995 International Joint Conference on Artificial Intelligence,* Montreal, CA, 1995.

Mobasher, B. (2001). Mining Web Usage Data for Automatic Site Personalization. To appear in Gaul, W.,& Ritter, G. (Eds.), *Classification, Automation, and New Media,* Springer.

Perkowitz, M. & Etzioni, O. (1998). Adaptive Web Sites, Automatically Synthesizing Web Pages. In *Proceedings of the Fifteenth National Conference on Artificial Intelligence,* Madison, WI.

Resnick, P., Iacovou, N., Suchak, M., Bergstrom, P.,& Riedl, J. (1994). GroupLens: An Open Architecture for Collaborative Filtering of Netnews. In *Proceedings of the Conference on Computer Supported Cooperative Work,* Chapel Hill NC, 1994, pp. 175–186.

Sarwar, B., Karypis, G., Konstan, J.A.,& Riedl, J. (2000). Analysis of Recommendation Algorithms for E-Commerce. In *ACM Conference on Electronic Commerce (EC-00).*

Schafer, J.B., Konstan, J.A.,& Riedl, J. (1999). Recommender Systems in E-Commerce. In *ACM Conference on Electronic Commerce (EC-99)*, pp. 158–166.

Schafer, J.B., Konstan, J.A.,& Riedl, J. (2000). Electronic Commerce Recommender Applications. *Journal of Data Mining and Knowledge Discovery 5/1,* pp. 115–152.

Stotts, P.D. & Furuta, R. (1991). Dynamic Adaptation of Hypertext Structure. In *Third ACM Conference on Hypertext Proceedings,* Association of Computing Machinery.

Yan, T.W., Jacobsen, M., Garcia-Molina, H.,& Dayal, U. (1996). From User Access Patterns to Dynamic Hypertext Linking. In *Fifth International World Wide Web Conference,* May 6–10, 1996, Paris, France.

Zaki, M. & Hsiao, C.-J. (1999). CHARM: An Efficient Algorithm for Closed Association Rule Mining. RPI Tech. Report. 99-10.