Next-Stop Recommendation to Travelers According to Their Sequential Wandering Behaviors

Dirksen Liu, Maiga Chang
School of Computing and Information Systems, Athabasca University, Canada
dirksen@gmail.com, maiga@ms2.hinet.net

Abstract

This paper proposes a novel solution of route recommendation which makes recommendations of potential next-stops to the users based on their previous wandering behaviors. The proposed recommendation method matches the user's current wandering behavior with the set of popular route patterns. Sequential pattern mining methods are used to extract the popular route patterns from a large set of historical route database collected from previous users. A complete example is used to verify the effectiveness of the proposed solution.

Keywords: Sequential, Location-based service, Route pattern, Data mining, Mobile, Casual wandering.

1 Introduction

Planning a route between two any given locations is the main feature of many navigational applications: such as the ever-popular portable/vehicle GPS navigational devices. These applications take route planning as a classical problem of finding the shortest path between two vertexes on a graph. There are many shortest-path finding algorithms, such as Dijkstra's algorithm [4], and its variants A* search algorithm [7], D* search algorithm [13], etc. which can produce route plans efficiently. However, not every case of traveling involves only one destination. This research looks into a different kind of traveling behavior that involves multiple destinations -- casual wandering.

This research is motivated by the need to make recommendations for casual wanderers, who travels between a series of locations attracting to them. Examples of casual wandering behavior can be found in small touring activities such as museum going, gallery visiting, zoo exploring, or as big as an excursion in a national park, or a day trip in a city tour. Travelers in these cases are more concerned about visiting the right places -- the places they like. On the other hand, they care less about how to minimize the time or distance of traveling, although reasonable economy is expected (such as no wasteful traveling of zigzagging).

The basic strategy of making route recommendation is to extract popular route patterns from a large route database and recommend the route patterns whose prefixes are most similar to the active user's current route. The underlying assumption of this strategy is that those who agreed in the past tend to agree again in the future, so if the active user's route is similar to the prefix of a popular route pattern, then there's a good chance the active user will follow the rest of the popular route pattern.

The goal of this research is to (1) identify an approach to extract popular route patterns from a historical route database and (2) to devise a method of finding the top-N recommendations via popular route pattern most similar to the active user's route, effectively and efficiently.

2 Recommender System in Sequential Context

Making recommendations is also the main feature of a large variety of applications called recommender systems which are achieving widespread success in E-Commerce nowadays. Examples of such applications include recommending books, CDs and other products at Amazon.com [10], movies by MovieLens [11], pre-detect fraud report by accounting [8] and news at VERSIFI Technologies [3].

Recommender systems are usually classified into two categories, based on how recommendations are made [2]: content-based recommendations and collaborative recommendations. Content-based systems recommend items similar to those that a user liked in the past [9] while collaborative recommender systems (or collaborative filtering systems) try to predict the preference of items for a particular user based on the items previously rated by other users [5]. Typical examples include the book recommendation system from Amazon, the PHOAKS system that helps people find relevant information on the WWW [14] and the Jester system that recommends jokes [6].

The method being proposed to solve the route recommendation problem is inspired by the collaborative methods, in that it also tries to identify the previous users that share the same choice pattern with the active user, and then recommend the next choice made by those previous users to the active user. However, most of the current collaborative recommender systems do not consider the order of choices made by the users, while sequential order is a crucial factor in the route recommendation problem. This issue will be discussed in details in Section 3.

*Corresponding author: Maiga Chang; E-mail: maiga@ms2.hinet.net

There are a number of recommender systems that take the order of user's choices into account. Shani, Brafman and Heckerman (2005) view the recommendation process as a sequential decision problem and propose using Markov decision processes (a well-known stochastic technique for modeling sequential decisions) for generating recommendations [12]. Tseng and Lin (2006) use n-gram (another derivative from the Markov chain model) based sequential pattern mining techniques to mine the mobile user's web usage sequence, aligned with the location sequence where the user uses the web [15]. Although their objective is to predict the next user request and the next location, to reduce mobile web surfing latency, their work is strongly related the recommender system. After all, making recommendations is to predict what users like.

3 Scenario of Bar Tour Guide

The motivation of this research is to find a solution for the Bar Tour Guide application. The main objective of the application is to recommend bar tour routes to its users. A typical scenario of the Bar Tour Guide application usage is as follows. The user carries the Bar Tour Guide handheld device as setting out on a tour. The device constantly monitors the user's routing activities. The user will find the first bar by himself/herself (the application will not make recommendation until the user has made the first choice; more discussion later in the chapter). As soon as the user finishes the drinking and is stepping out of the first bar, the Bar Tour Guide will start the calculation, and presents a list of top-N next popular bar that are most likely favored by the user. The user may or may not take the recommendation, so the next time the user issues a new request for recommendation, the system will recalculate the recommendations based on the most up-to-date user route.

For example, Maz is bar touring in downtown Toronto (see Figure 1 for a map of some bars and pubs in that area, and Maz's route on the map). He first visits G, then A, and now he wonders which one to go next. So he pulls out his GPS iPhone, which is running the Bar Tour Guide program. The system scans a list of popular bar tour routes, and discovers that many previous bar tour goers, who drank at G then A, would pick E as the next hop. Therefore, E becomes the recommendation for Maz.

The problem as how to get from one stop to the next in the route on the street level can be addressed with conventional navigational tools, and thus is out of the scope of this paper.

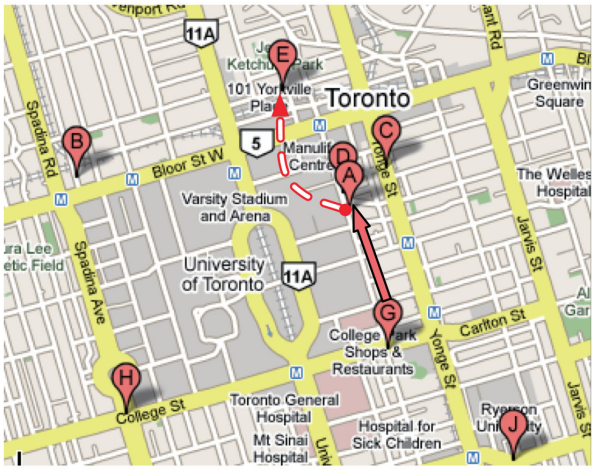

Figure 1 Bar Touring in Downtown Toronto

4 Issues Needed to Solve

First, let us define what a route is. Since we are mostly concerned about the choices made by the bar tour goers, so a route can be represented as an ordered list of locations. We are also only interested in a number of specific types of locations, referred to as significant locations (in this case, drinking establishments). In other words, if the user drops into a café, a flower shop, during his/her bar tour, those stops will not be recorded in his/her route.

Formally, significant locations Λ is a set of predefined geo positions of significant interest, such as the geo positions of all bars in a city in the case of bar touring. The member of Λ is denoted as a 3-tuple (*latitude*, *longitude*, *name*). A user route $route_{userID}$ is an ordered list of locations, denoted by the form $\langle \lambda_1 \lambda_2 \ldots \lambda_n \rangle$, where λ_j Ì Λ. A sequence with length k is referred to as a *k-sequence*.

The objective of any recommendation problem is to predict what the user likes. The route recommendation problem, presented by the Bar Tour Guide application, is unique from other conventional recommender system, in that it needs to predict not only what the user likes, but also in what order.

Many recommender systems, especially those that follow the collaborative approach, exploit the observation that like-minded people behave similarly, and thus often make similar decisions again. The basic strategy of the solution being proposed follows the same line of thinking. It takes the following three steps:

1. Look up recurring behavior patterns as what the previous users have chosen to visit, and in what order.
2. Compare the current active user's behavior against the patterns discovered in step one, identify those patterns that matches the current user. Obviously the previous users exhibited such behavior patterns share common preference profiles with the active user, hence

what locations they visited next can be presented as recommendations as those locations would be very likely preferred by the active user as well.

3. There is often more than one recommendation. To avoid bombarding the active user with too many options, the system should rank each recommendation according to some certain scheme, to help the user to decide which one to take.

Given a database Θ of routes, and an active user route $route_{userID}$, and according to the three-step strategy, the route recommendation problem can be divided into three research issues:

1. how to extract popular route patterns from the database Θ;
2. how to generate recommendations from the popular route patterns; and,
3. how to rank the relevancy of the recommendations to the active user given his/her route, $route_{userID}$?

5 Sequential Routing Patterns

Most current recommender systems consider patterns as sets of commonly chosen items. The order of the items in a set is deemed to be irrelevant. Non-sequential pattern based recommender systems are very popular in product recommendation, such as books, music, movies, etc. In these cases, the involved effort to acquire such products makes little influence on the user's decision making and thus often ignored.

The case is different for route recommendation because there is a cost of traveling incurred on the users moving from one location to the next. This cost will add up as the users travel through a number of locations, and thus the total traveling cost is determined by the visiting order of these locations. As such, users are no longer making decisions solely based on what they like, but also the cost involved in traveling as well. More precisely, the thinking process now becomes:

1. What do I like to visit next?
2. Is it too far? Do I like it so much that I'm willing to travel that far?
3. Is it in my general travel direction? (to keep the total cost down, a common strategy is to move in a constant direction)

Without considering the sequential order, the pattern will not be able to capture the decision making process behind the user's behavior, and thus will failed to find the real like-minded users and present recommendations that's out of context.

Figure 2 illustrates a typical case where the simple preference-oriented recommender systems would have failed. Two users have been traversing the same set of locations $\langle B\ C\ D\ E\rangle$, but in opposite directions. The simple preference-oriented recommender systems will consider the two users as being like-minded, and will recommend choice of the next destination of each user to the other (i.e., A to User #1, and F to User #2), which does not make sense to either given their context.

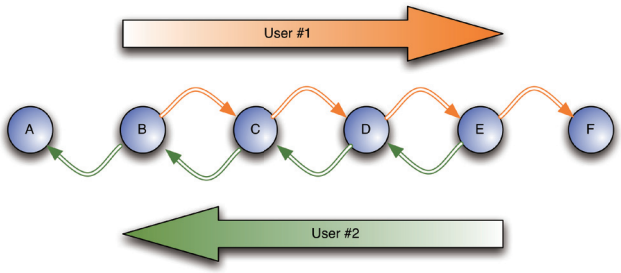

Figure 2 Preference-Oriented vs. Sequential Method

In summary, route patterns must be sequential. Given routes are sequences, route patterns should be common sub-sequences of the original routes.

What is sub-sequence? Shani, Brafman and Heckerman (2005) and Tseng and Lin (2006) both seek to extract correlation between users by examining the sub-sequences of the users' action sequences in specific problems [12] [15]. Their methods are both based on Markov chain models, more specifically the n-gram model, which is a type of probabilistic model for predicting the next item in a sequence. An n-gram is a sub-sequence of n items from a given sequence. Formally, a sequence $S_i' = \langle s_{i1}'\ s_{i2}'\ \ldots\ s_{in}'\rangle$ is said to be a *sub-sequence* of an original sequence $S_i = \langle s_{i1}\ s_{i2}\ \ldots\ s_{im}\rangle$, where $n \le m$, if there exists a strictly increasing sequence of indices, namely $j_1 < j_2 < \ldots < j_n$, such that $s_{i1}' = s_{ij1}$, $s_{i2}' = s_{ij2}$, …, $s_{in}' = s_{ij_n}$.

Sub-sequence is an exact fragment of the original sequence. However, focusing only on local fragments, we would loose sight of patterns that span non-consecutively across the original sequences. For example, $\langle A\ B\ C\rangle$ and $\langle A\ D\ C\rangle$ are routes that do not share any consecutive sub-sequences and thus will be dismissed by any Markov chain model as not having any connections between them. And yet that's not true, as they both contain A and C, and in the same order. That indicates there is some connection between them. The two users who generated these two routes share something in common. In this case, $\langle A\ C\rangle$ is a common pattern occurred in both routes.

Sequential patterns that occur non-consecutively in original sequences are better media to capture the common features of those sequences. Figure 3 demonstrates the power of non-consecutive sequential patterns. It depicts two user routes. User 1 starts from A, and after visiting an arbitrary number of locations, D, E, and F, User 1 arrives at G. User 2 starts from H, and then D, E, F, I consecutively. Obviously their routes have a common sequence fragment $\langle D\ E\ F\rangle$. In the example, there are two

routes sharing a common fragment ⟨*D E F*⟩. Since equal number of users visit ⟨*D E F*⟩, but end up in different destination, the Markov chain model cannot determine the probability difference between the two routes. If we expand our visual scope to the entire bodies of both routes, we can then discover the correlations between *H* and *I*, *A* and *G*.

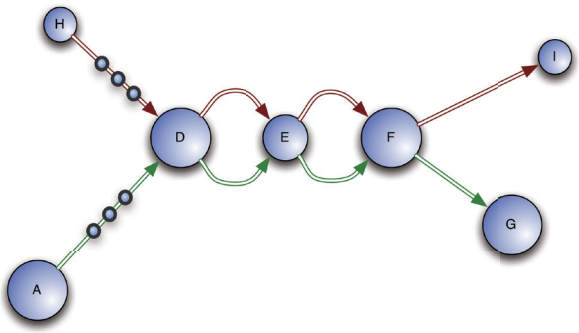

Figure 3 Sequence Fragments vs. Sequential Patterns

The relaxation of the consecutiveness constraint gives rise to a new issue. Suppose *A* and *G* are two locations so far apart that no users who are currently visiting *A*, would make *G* as their next destination. Here User 1's route $route_1$ = ⟨*A* …. *G*⟩ starts from *A*, passes a number of other locations and finally ends at *G* as Figure 3 shows. By definition, ⟨*A G*⟩ is a sub-sequence of $route_1$, and thus has a chance of becoming a route pattern. This is in contrast to the fact that no one would travel directly from *A* to *G*, and thus not a faithful reflection of the reality. As such, a constraint on distance between adjacent elements is needed to ensure the route pattern itself is a possible route. The upper limit of the distance between adjacent elements of a pattern is a user-defined parameter, referred to as *maximal_distance*.

Summarizing the above discussion, a routing pattern can be defined as a recurring sub-sequence with significant frequency in the routing sequence database. The significance of a routing pattern is measured by its support. Given a database $\Theta = \{S_1, S_2, ..., S_N\}$ that contains *N* sequences, the support of a routing pattern is defined as

$$sup(pattern) = \frac{count\left(pattern \subset S_i and 1 \le i \le N\right)}{N} \quad (1)$$

A routing pattern is said to be *popular*, or *frequent*, if its frequency of occurrences in the database is no less than a certain user-defined minimum support (or referred to as the *min_sup*). For example, in a database which has the following three routes: ⟨*A B C D*⟩, ⟨*E B C F*⟩, ⟨*H I J*⟩, with minimum support of 50%, the route pattern ⟨*B*⟩, ⟨*C*⟩, ⟨*B C*⟩ are all popular since they appear two times out of 3. A routing pattern is said to be *maximal*, if it is not a sub-sequence of any other popular routing patterns. In the above example, ⟨*B C*⟩ are maximal, while ⟨*B*⟩, ⟨*C*⟩ are not.

6 Recommendation Rules

After we find the patterns, we need to connect the active user with previous users through route pattern matching and generate route recommendations from the matching route patterns.

The solution to the first task is straightforward. Since a number of previous users are said to be like-minded if their routes contain the same route pattern, then in the same line of thought, an active user whose route also contains this pattern will have a strong connection with these users as well. When a route pattern is found contained in an active user route, it is said to be a match to the active user route.

With a number of previous user found to be like-minded to the active user, the next step is to find out suitable next-stops for recommendations among those visited by this group of previous users, while not yet visited by the active user. More importantly, the suitableness of a next-stop follows these criteria:

1. The next-stop should be visited by a significant number (over *min_sup*) of users in the previous user group identified by the pattern; if not, it is probably not worth to recommend.
2. The next-stop should occur after the pattern in the previous user route, otherwise the system will recommend the active user to go backward.
3. The next-stop should not be too far (less than *maximal_distance*) from the last stop of the pattern, since the active user would not take it if it is too far from where he/she is at right now.

If such a next-stop can be found, then appending this new stop to the pattern will produce a new pattern, as it satisfies all requirements of route patterns. It is a popular sub-sequence occurs in a significant number of existing routes and the distance between every adjacent stop does not exceed *maximal_distance*. Being a route pattern, it will be discovered by the sequential pattern mining process. Therefore we can exploit the route patterns to generate recommendations by converting them into recommendation rules. For a popular route pattern ⟨s_{i1} s_{i2} … s_{im}⟩, a recommendation rule can be generated as follows:

$$rule = <s_{i1}\ s_{i2} \ldots s_{i(m-1)}> \rightarrow s_{im} \quad (2)$$

The antecedent ⟨s_{i1} s_{i2} … $s_{i(m-1)}$⟩ is also termed as the left hand side of the rule denoted as *LHS*(*rule*), is to be used as the pattern to match with the active user's route; and the consequent s_{im} is termed as the right hand side of the rule and is denoted as *RHS*(*rule*), is the next stop the system is going to recommend to the active user.

Given an active user route $route_{userID} = \langle s_{j1}\ s_{j2}\ \ldots\ s_{jn}\rangle$, the $rule = \langle s_{i1}\ s_{i2}\ \ldots\ s_{i(m-1)}\rangle \rightarrow s_{im}$ is said to match or applicable to $route_{userID}$ if $origin(rule) \subseteq route_{userID}$ and $s_{i(m-1)} = s_{jn}$, where *origin* is a function that returns the original route pattern for any given *rule*. The additional constraint $s_{i(m-1)} = s_{jn}$ is necessary to avoid making out-of-context recommendations. For example, *A*, *B*, *C* and *D* are four consecutive stops on a street, with *A* on one end and *D* on the other. A user has visited $\langle A\ C\ D\rangle$. Without the additional constraint, his route $\langle A\ C\ D\rangle$ matches the rule $\langle A\rangle \rightarrow \langle B\rangle$, therefore the system will suggest *B* as the next-stop, which makes no sense to the user, as the user has clearly passed *B*.

As sequential mining methods will return all popular patterns, popular *1-sequences* (sequences of length 1) will be among the discovered pattern as well. However, the above definition excludes popular *1-sequence* from recommendation rules generation. If the only one item in the 1-sequence is used as recommendation, then there leaves nothing for the system to infer connections between the active user and the popular *1-sequences*. The system will be unable to make any recommendation at all.

This is a well-known issue called *cold start*, common among recommender systems. A simple solution is to recommend the stops in those popular *1-sequences* which are within the *maximal_distance* radius to the active user's current location. If none can be found, then the system should refrain from making any recommendations.

For an active user, there could be a number of matching route patterns, leading to multiple recommendations. This is most common when the user starts his/her tour. To avoid bombarding the user with too many options, the system should rank the recommendations to help user to choose. In other words, the purpose of the ranking is to estimate how well the active user would like each of the recommendations, which is in turn determined by how strong a connection exists between the rule that generates this recommendation and the active user.

The connection is established between a rule (take $rule_A$ as an example here) and the active user, when $LHS(rule_A)$ is contained in the active user's route pattern. As such, the length of $LHS(rule_A)$ is a good indicator to represent how similar $rule_A$ is with the active user route pattern, i.e., the longer the $LHS(rule_A)$ is, the more traits of $rule_A$ occurs in the active user route pattern, the stronger the connection is therefore the more like-minded the active users is to the group of previous users that generate $rule_A$. We design a function *length* which returns any sequence's length, i.e., *length* ($LHS(rule_A)$).

The recommendability of a rule can be measured by the similarity between the rule's *LHS* and the active user route pattern. Given a $rule_i$ and $route_{userID}$, the similarity can be defined as:

$$simularity(rule_i, route_{userID}) = \frac{length(LHS(rule_i))}{length(route_{userID})}\ x\ 100\% \quad (3)$$

However, *LHS*-length-based *simularity* may not be enough to rank the rules. For example, at the beginning of a tour, when the active user has only visited one place, all the rules available for matching will all have their *LHS* being one-element long. To tell which rule is more recommendable than the others when they have the same *LHS*-length, we can turn to the support of the route pattern behind each rule (cf. Equation 1). However, the support is only a measurement of how popular this rule is among the previous users. It tells nothing about the connection with the current active user. As such, a concept of *confidence* is borrowed from the idea of *association rules* [1], and is defined as:

$$conf(rule_i) = \frac{sup(< s_{i1}\ s_{i2} \ldots s_{im} >)}{sup(< s_{i1}\ s_{i2} \ldots s_{i(m-1)} >)}\ x\ 100\% \quad (4)$$

By definition, the confidence predicts the probability that an active user will take the recommendation of a matching rule. Obviously the confidence is a more relevant measurement over the support. However, a confidence-only approach will likely produce unpopular result of high confidence. To balance the measurement, another concept of *strength* is borrowed from [15] and is defined as:

$$strength(rule_i) = sup(rule_i) * conf(rule_i) \quad (5)$$

Combining the *strength* and *similarity*, the general recommendability of a $rule_i$, can be defined as:

$$\begin{aligned} recommendability(rule_i, route_{userID}) = \\ bias\_similarity * simularity(rule_i) + \\ (1 - bias\_similarity) * strength(rule_i), \end{aligned} \quad (6)$$

where the parameter *bias_similarity* allows the user to give bias toward similarity or strength.

7 System Architecture and Example

The operation flow of the system has three phases -- tracking, mining and recommendation. Accordingly, the system is logically divided into three modules, as illustrated in the following Figure 4.

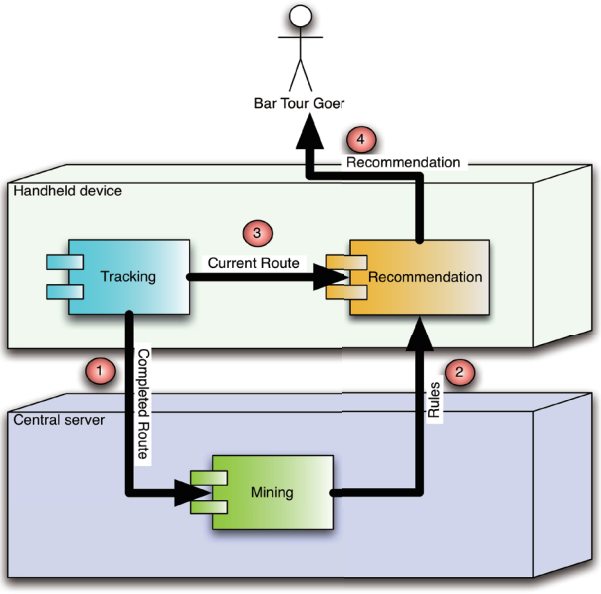

Figure 4 System Architecture

The task for the tracking module is to monitor and record the user's behavior data and preprocess them into routing data, which is its output. The input of the mining module is completed routes. Its task is to collect such completed routes from all handheld devices and mine them for popular patterns. These patterns will be used to generate recommendation rules. The rules are then the output of the mining module. At the same time these rules are one of the inputs to the recommendation module. The other input is the current routing data at the moment of user's request. The task of this module is to produce recommendation by running the current user routing data through a set of recommendation rules.

We use a complete example here to show how the three phases work with sample data. Figure 5 marks a number of sample tour routes on the map to help us demonstrating the mining process for popular route patterns. Table 1 lists the route database.

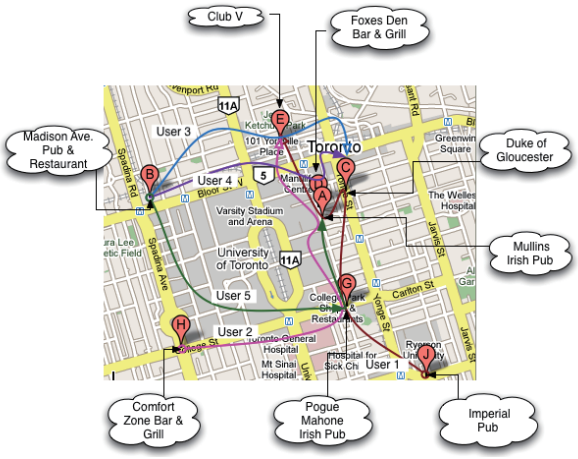

Figure 5 Sample Tour Routes

Table 1 Routes Database

User Id	Routes
1	$\langle J\ G\ C\ A\ E \rangle$
2	$\langle H\ G\ A\ D\ E \rangle$
3	$\langle B\ E\ A\ C \rangle$
4	$\langle B\ D\ A\ C \rangle$
5	$\langle B\ G\ A \rangle$

Suppose the minimal support is 40%, i.e., any route pattern has to have at least 2 occurrences in the above database to be counted as popular and set *maximal_distance* to 1.5km. Following the algorithm, we first look for popular *1-sequence* patterns as Table 2 lists, i.e., route patterns of 1-stop long that have over minimal support.

Table 2 Popular *1-sequence* Patterns

1-sequence Patterns	Support
A	100%
B	60%
C	60%
D	40%
E	60%
G	60%

Then we generate candidate *2-sequence* patterns by self-joining the *1-sequence* patterns and eliminate those patterns whose support values are less than the minimal support. Following the same process, we can find the only popular *3-sequence* pattern. At this point, we have found all popular route patterns in the database as Table 3 lists.

Table 3 Popular Route Patterns

1-sequence Patterns	*2-sequence* Patterns	*3-sequence* Patterns
A	*A C*	
	A E	
B	*B A*	*B A C*
	B C	
G	*G A*	*G A E*
	G E	

By definition, all *1-sequence* patterns cannot be turned into recommendation rules and therefore discarded. The rest of the patterns can generate the set of recommendation rules as Table 4 lists.

Table 4 Recommendation Rules

Rule ID	Recommendation Rules
1	$\langle G\ A\rangle \rightarrow E$
2	$\langle G\rangle \rightarrow A$
3	$\langle G\rangle \rightarrow E$
4	$\langle A\rangle \rightarrow E$
5	$\langle B\rangle \rightarrow C$
6	$\langle A\rangle \rightarrow C$
7	$\langle B\ A\rangle \rightarrow C$

With the set of recommendation rules, we are ready to make recommendations. Suppose Maz, has started his tour today. So far he has visited a number of places: $\langle J\ G\ C\rangle$, and he is just leaving from A. He's having a good time, quite satisfied with the recommendations. So he pulls out his built-in GPS mobile phone, and checks what can be the next interesting stop.

Table 5 Array of Recommendation Rules

Stop \ Rule	#1	#2	#3	#4	#5	#6	#7
A	-2			-1		-1	-2
B					-1		1
G	1	-1	-1				

In order to make the next-stop recommendation to Maz, our system first transforms the recommendation rules into Table 5.

Maz's route before reaching A was $\langle J\ G\ C\rangle$, the system has been matching his route against the *recommendation_rules* on every location he visited. Without going into too much detail, Table 6 lists the values of the *cursor* positions at this moment. The *cursor* value is counting from 1, for example, G in $\langle G\ C\ A\rangle$ makes the *cursor* value be 1 and C makes the *cursor* value be 2. However, please note the *cursor* value will be a negative number if the place locates at the end of a sequence, e.g., A in $\langle G\ C\ A\rangle$.

Table 6 Array of *cursor* Position

Rule ID	$LHS(rule_i)$	*Cursor* Value of $\langle J\ G\ C\rangle$ in $rule_i$
1	$\langle \boldsymbol{G}\ A\rangle$	1
2	$\langle \boldsymbol{G}\rangle$	1
3	$\langle \boldsymbol{G}\rangle$	1
4	$\langle A\rangle$	0
5	$\langle B\rangle$	0
6	$\langle A\rangle$	0
7	$\langle B\ A\rangle$	0

As Maz is leaving A, the system consults the *recommend-ation_rules* table (i.e., Table 4), and obtains the list of rules containing A, i.e., rule #1, #4, #6 and #7. By joining Table 4 and Table 6 on Rule ID, we get Table 7.

Table 7 A Joined Table to See Which Rules Can Be Ignored

Rule ID	*Cursor* Value	Position of *A*	|Position of *A*|
1	1	-2	2
2	1		
3	1		
4	0	-1	1
5	0		
6	0	-1	1
7	0	-2	2

Since we are looking for rules whose *LHSs* end with A; and rule #2, #3 and #5 can be removed due to the fact that none of them contains A. The *cursor* value of A in rule #1 is -2, the absolute value of which is greater than the *cursor* value for $\langle J\ G\ C\rangle$ by exactly one. So rule #1 completely matches Maz's route. The same goes with rule #4 and #6. Therefore rule #1, #4 and #6 are both eligible to make recommendations to Maz.

Now it's time to measure the recommendability of these eligible rules. Suppose Maz has set the system to recommend only the top one choice, and set *bias_similarity* to 0.7. Table 8 lists the similarity, the strength, and the recommendability of all eligible rules.

Table 8 The Similarity, Strength and Recommendability of the Eligible Rules

Rule ID	similarity	strength	recommendability
1	0.5	0.27	0.43
4	0.25	0.6	0.35
6	0.25	0.16	0.22

As a result, $\langle G\ A\rangle \rightarrow E$ (recommendability = 0.43) is more recommendable than $\langle A\rangle \rightarrow E$ (recommendability = 0.35) and then $\langle A\rangle \rightarrow C$ (recommendability = 0.22). We have E, E and C as recommendation candidates, which can be further reduced to E and C. As Maz only wants the top one choice in the candidate list, the system chooses to recommend Maz E as his next stop since E has much higher recommendability value.

8 Conclusions

This research studies a novel problem -- route recommendation based on behavior patterns. Based on

the observation that user's routing behavior is a sequential decision making process, the concept of sequential patterns is being used to define behavior patterns. As a result, sequential pattern mining methods are used to extract popular behavior patterns, which are then turned into a set of recommendation rules. Given an active user, the system will first find out which rules are applicable to the user, sort the rules according to a ranking scheme and finally present the top-n highest-ranking rules' *RHS* as the recommendations to the user.

We have already done the simulation test to verify the effectiveness of the proposed method. In the next step, we plan to build the application on the mobile phone and do the pilot experiment. The proposed recommendation method can be used in other application domains, e.g., sightseeing, museum visiting, shopping and cultural and historical spots learning, as long as the behavior data is a sequential one.

In the experiment, we plan to not only use questionnaire to verify if the recommendation fits the users' requirements, but also to analyze the recommendations and the user's followed behavior to see if the user appreciates the recommended next-stops. We always think that a recommendation system should only provide users suggestions rather than force them to comply.

At last, the proposed method currently only uses time gap and minimal range in distance to deal with noisy data. In the future research, we would consider to use Wavelet and other DSP (Digital Signal Processing) methodologies to filter the noise out.

References

[1] Rakesh Agrawal and Ramakrishnan Srikant, *Mining Sequential Patterns*, *Proc. of the Eleventh International Conference on Data Engineering*, Taipei, Taiwan, March, 1995, pp.3-14.

[2] Marko Balabanović and Yoav Shoham, *Fab: Content-Based, Collaborative Recommendation*, *Communications of the ACM*, Vol.40, 1997, pp.66-72.

[3] Daniel Billsus, Clifford A. Brunk, Craig Evans, Brian Gladish and Michael Pazzani, *Adaptive Interfaces for Ubiquitous Web Access*, *Communications of the ACM*, Vol.45, 2002, pp.34-38.

[4] Edsger Wybe Dijkstra, *A Note on Two Problems in Connexion with Graphs*, *Numerische Mathematik*, Vol.1, 1959, pp.269-271.

[5] David Goldberg, David Nichols, Brian M. Oki and Douglas Terry, *Using Collaborative Filtering to Weave an Information Tapestry*, *Communications of the ACM*, Vol.35, 1992, pp.61-70.

[6] Ken Goldberg, Theresa Roeder, Dhruv Gupta and Chris Perkins, *Eigentaste: A Constant Time Collaborative Filtering Algorithm*, *Information Retrieval*, Vol.4, 2001, pp.133-151.

[7] Peter E. Hart, Nils J. Nilsson and Bertram Raphael, *A Formal Basis for the Heuristic Determination of Minimum Cost Paths*, *IEEE Transactions on Systems Science and Cybernetics*, Vol.4, 1968, pp.100-107.

[8] Yeog Kim, Sang Jin Lee and Jong In Lim, *Fraud Detection for Information Reliability from the Internet in Forensic Accounting*, *Journal of Internet Technology*, Vol.11, 2010, pp.323-331.

[9] Ken Lang, *Newsweeder: Learning to Filter Netnews*, *Proc. of the Twelfth International Machine Learning Conference*, Tahoe City, CA, July, 1995, pp.331-339.

[10] Greg Linden, Brent Smith and Jeremy York, *Amazon.com Recommendations: Item-to-Item Collaborative Filtering*, *IEEE Internet Computing*, Vol.7, 2003, pp.76-80.

[11] Bradley N. Miller, Istvan Albert, Shyong K. Lam, Joseph A. Konstan and John Riedl, *MovieLens Unplugged: Experiences with an Occasionally Connected Recommender System*, *Proc. of the Eighth International Conference on Intelligent User Interfaces*, Miami, FL, January, 2003, pp.263-266.

[12] Guy Shani, David Heckerman and Ronen I. Brafman, *An MDP-Based Recommender System*, *Journal of Machine Learning Research*, Vol.6, 2005, pp.1265-1295.

[13] Anthony Stentz, *Optimal and Efficient Path Planning for Partially-Known Environments*, *Proc. of the 1994 IEEE International Conference on Robotics and Automation*, San Diego, CA, May, 1994, pp.3310-3317.

[14] Loren Terveen, Will Hill, Brian Amento, David McDonald and Josh Creter, *PHOAKS: A System for Sharing Recommendations*, *Communications of the ACM*, Vol.40, 1997, pp.59-62.

[15] Vincent S. Tseng and Kawuu W. Lin, *Efficient Mining and Prediction of User Behavior Patterns in Mobile Web Systems*, *Information and Software Technology*, Vol.48, 2006, pp.357-369.

Biographies

Dirksen Liu (Decheng Liu) received the BSc degree in Computer Science and BA degree in English from South China University of Tech. in 1997, and the MSc degree in Information System from Athabasca University in 2009. He is currently a software developer working for InfoBright.com, a column-oriented DB producer. His interests include data mining, column-oriented DB and software engineering methodologies.

Maiga Chang is Assistant Professor in the School of Computing Information and Systems, Athabasca University (AU), Athabasca, Alberta, Canada. His researches mainly focus on mobile learning and ubiquitous learning, museum E-learning, game-based learning, educational robots, learning behavior analysis, data mining, intelligent agent technology, computational intelligence in E-learning and mobile healthcare. He is the local chair of IEEE DIGITEL 2008, general co-chair of Edutainment 2009, and program co-chair of Edutainment 2011. He has participated in 124 international conferences/workshops as a Program Committee Member and has (co-)authored more than 126 book chapters, journal and international conference papers. In September 2004, he received the 2004 Young Researcher Award in Advanced Learning Technologies from the IEEE Technical Committee on Learning Technology (IEEE TCLT). He is a valued IEEE member for fourteen years and also a member of ACM, AAAI, INNS, and Phi Tau Phi Scholastic Honor Society.