ATHABASCA UNIVERSITY

Route Recommendation

Based On Behavior Analysis

By

Dirksen Liu

An essay submitted in partial fulfillment

Of the requirements for the degree of

MASTER OF SCIENCE in INFORMATION SYSTEMS

Athabasca, Alberta

December, 2008

DEDICATION

This work is dedicated to my wife AJ, and son Vincent whose sacrifices, which were realized by our loss of precious time together, were for me the most painful and humbling of all. It is also dedicated to my brother James, for his valuable advice and insights.

ABSTRACT

This essay presents a novel problem of route recommendation which guides the user through a series of locations. The recommendation is made by matching the user's current route 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 records from previous users. An application with real requirements – the Bar Tour Guide – is used to demonstrate the effectiveness of the proposed solution.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

CHAPTER I

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 (Dijkstra, 1959), and its variants A* (Hart, Nilsson, & Raphael, 1968) search algorithm, D* search algorithm (Stentz, 1994), 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.

1.1. MOTIVATION

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 touristic 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.

This research looks into a novel activity of casual wandering – *bar tour* (sometimes called a bar tour, pub crawl or pub-hopping). A bar tour is an touristic activity where one or more persons visit and drink in a series of bars, pubs, or clubs in one single night, usually within walking distance. It's very popular among tourists with a taste of nightlife, who would naturally have a great desire to savor the night life of the city they are visiting. It is another form of foraging travel. The objective is not just for drinks, but also for the experience of fun and relaxation when hanging out with friends. The drinking experience varies a great deal from bar to bar. In fact, due to competitions, operators of drinking establishment will go to a great length to create unique experiences, by controlling the factors such as the décor, the music, the environment, the atmosphere, etc. As a result, whether someone will find a bar agreeable highly depends on this person's taste and preference. However, to judge whether a bar fits one's taste, one has to get inside of the establishment. Being tourists in an unfamiliar territory, they have to visit one bar after another to find out the ones they like.

To help these tourists, this research presents a route recommendation application – the Bar Tour Guide. Similar to the popular GPS vehicle navigational devices, the Bar Tour Guide runs on a handheld device equipped with a GPS sensor. But instead of recommending route between two specific locations, the device will monitor the user's bar tour activity, then predict and recommend the

top-N bars next to visit that match the user's preference. As the user gets on the bar tour, and the system will make a series of recommendations, leading the user through a bar tour route.

1.2. GOAL AND CONTRIBUTIONS

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.

The essay will also discuss several implementation issues, and make a couple of contributions accordingly. They are:

1) **An algorithm to preprocess GPS signal stream into routing sequence:** as the mining process and the recommendation process all deals with routing sequence, not GPS signals.

2) **An incremental pattern matching algorithm:** as the number of patterns generated could be large, the matching of patterns and the active user route may take too long to process. An incremental algorithm makes the process more efficient.

1.3 CHAPTER DESCRIPTIONS

The rest of the essay is structured as follows: Chapter two provides the background information and discusses the related work. Chapter III describes the system architecture and work-flows of the Bar Tour Guide application, and analyzes the main problems behind its workings, and projects the major challenges in finding the solutions to these problems. The solutions themselves are presented in Chapter IV in details. Chapter five illustrates the whole process by applying the above solutions to an example. Finally Chapter six concludes the essay and outlines future research directions.

CHAPTER II

RELATED WORK

This chapter reviews the past studies by the three subjects, route planning for navigation, recommender systems, and specific recommender systems that use sequential model or patterns.

2.1. ROUTE PLANNING FOR NAVIGATION

All GPS navigational devices make route plan recommendation their major functionality. Classical route planning algorithms such as the Dijkstra algorithm only find paths with the shortest travel distance. New research, such as Nanayakkara, Srinivasan, Lai, German, Taylor, and Ong (2007), Kanoh and Hara (2008), has expanded the problem to find the optimal route dynamically that satisfies multiple objectives, such as travel time, and ease of driving, etc. Other researches, such as Rogers and Fiechter (1999), and Park, Bell, Kaparias and Bogenberger (2007) take drivers' preference into consideration of route selection in planning.

2.2. RECOMMENDER SYSTEMS

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 (Linden, Smith & York, 2003), movies by MovieLens (Miller, Albert, Lam, Konstan & Riedl, 2003), and news at VERSIFI Technologies (Billsus, Brunk, Evans, Gladish, and Pazzani, 2001).

Recommender systems are usually classified into two categories, based on how recommendations are made (Balabanovic & Shoham, 1997): content-based recommendations, and collaborative recommendations. Content-based systems recommend items similar to those that a user liked in the past (Lang, 1995), 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 (Goldberg, Nichols, Oki & Terry, 1992). Typical examples include the book recommendation system from Amazon, the PHOAKS system that helps people find relevant information on the WWW (Terveen, Hill,., Amento, B., McDonald, D. & Creter, 1997) and the Jester system that recommends jokes (Goldberg, Roeder, Gupta & Perkins, 2001).

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 3.2.

2.3. RECOMMENDER SYSTEM IN SEQUENTIAL CONTEXT

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. Tseng and Lin (2006) use n-gram (another derivative from the Markov chain model) based sequential pattern mining techniques to mind the mobile user's web usage sequence, aligned with the location sequence where the user uses the web. 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 they user likes.

These systems share the basic principle with this research, in that they all tap into the power of sequence to solve each respective problem. However, their definitions of sequential patterns are rather limited in power, which will be discussed in details in Section 3.2.

CHAPTER III

PROBLEM ANALYSIS

The motivation of this research is to find a solution for the Bar Tour Guide application. This chapter will start with a description the application and its requirements, and then identify two problems of research interests, and further dwell into various aspects of the two research problems.

3.1. BAR TOUR GUIDE

The main objective of this 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.

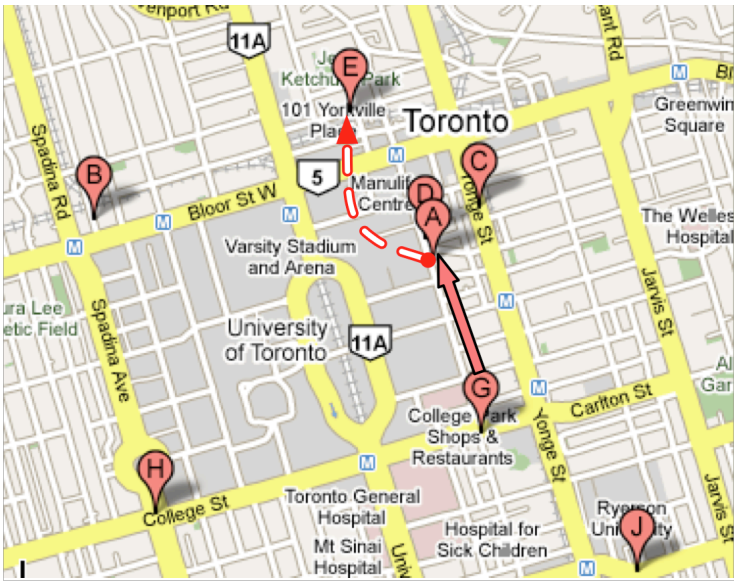


Figure 1. Bar touring in downtown Toronto

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 essay.

3.2. PROBLEM FORMULATION

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 $<\lambda_1 \lambda_2 \ldots \lambda_n>$, where $\lambda_j \subset \Lambda$. 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 are 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 problem of route recommendation can be divided into three research sub-problems:

1) How to extract popular route patterns from the database Θ;

2) How to generate recommendations from the popular route patterns;

3) How to rank the relevancy of the recommendations to the active user given his/her route $route_{userID}$.

3.3. DEFINING ROUTE PATTERNS

Pivotal to the aforementioned strategy, route pattern provides the means to find compatible preference profiles. Let us find out the correct definition of route

patterns by examining its various characteristics and comparing to other existing pattern definitions.

3.3.1. SUB-SEQUENCE

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. Applying to the case of route recommendation, the strategy would be turned into:

Find patterns as sets of locations that are commonly chosen by previous users

If the current active user has also chosen some locations in one of the patterns, then the rest of the locations in that pattern can be presented as recommendations.

Recommendations made by this approach are simple preference-oriented. In order words, they calculate predictions purely based on the users' preference profile. The cost of traveling to each location is not taken into consideration. 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. 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 traversing 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 (*B C D E*), 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 do not make sense to either given their context.

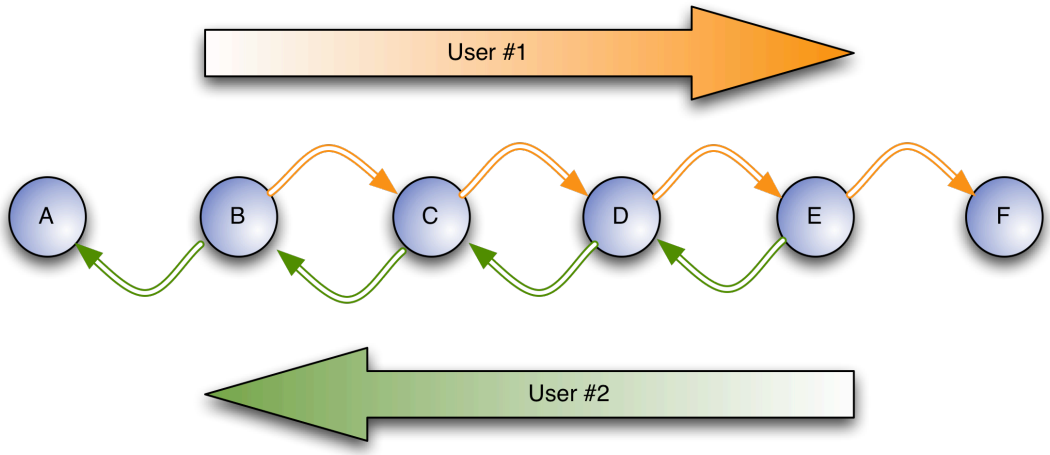


Figure 2. Preference-oriented methods vs. sequential methods

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

3.3.2. SEQUENCE FRAGMENT VS. SEQUENTIAL PATTERN

The conclusion in the last section begs a question: 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-sequence of their users' action sequences in their respective problems. Their methods are both based on the application of Markov chain models, more specifically the n-gram model, which are 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 $S_i' = <s_{i1}' \ s_{i2}' \ ... \ s_{in}'>$ is said to be a *sub-sequence* of an original sequence $S_i = <s_{i1} \ s_{i2} \ ... \ s_{im}>$, where $n \leq m$, if there exists a strictly increasing sequence of indices, namely $j_1 < j_2 < ... < j_n$, such that $s_{i1}' = s_{ij_1}$, $s_{i2}' = s_{ij_2}$, …, $s_{in}' = s_{ij_n}$. An n-gram model predicts s_{in} based on $<s_{i1} \ s_{i2} \ ... \ s_{in}>$.

By the above definition, 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, <*A B C*> and <*A D C*> 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 *B* and *C*, and in the same order. That indicates there is some connection between them. The two users who generate these two routes share something in common. In this case, <*A C*> 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, user 1 visits *D, E, F, G*, consecutively. User 2 starts from *H*, and then *D, E, F, I* consecutively. Obviously their routes have a common sequence fragment <*D E F*>, the difference between two definitions of sub-sequence. 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*.

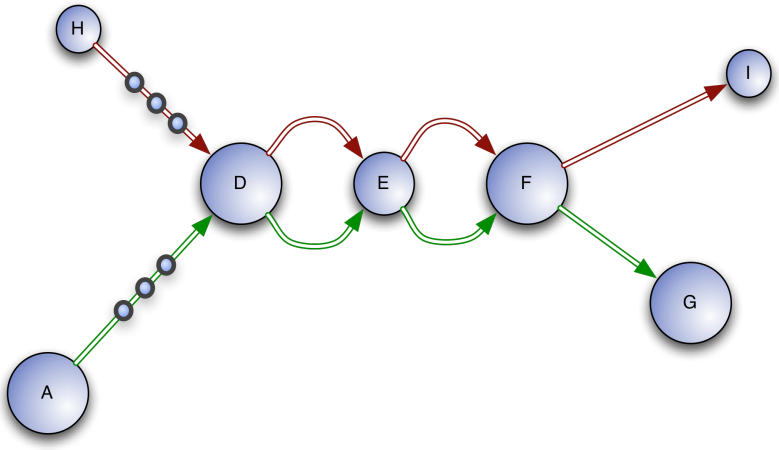


Figure 3. Sequence fragments vs. sequential patterns

Now the concept of sub-sequences can be more thoroughly defined as: a sequence is a sub-sequence of another, if all of its elements appear in the other sequence, and appear in the same order. Formally, a sequence $S_i' = \langle s_{i1}' \; s_{i2}' \; \ldots \; s_{in}' \rangle$ is said to be a *sub-sequence* of another $S_i = \langle s_{i1} \; s_{i2} \; \ldots \; s_{in} \rangle$ (S_i is also said to contain, or to be the *super-sequence* of S_i'), where $n \leq m$, if there exist integers $j_1 < j_2 < \ldots < j_n$ such that $s_{i1}' = s_{ij_1}$, $s_{i2}' = s_{ij_2}$, …, $s_{in}' = s_{ij_n}$.

3.3.3. ROUTE PATTERNS

In the last section, we reach the conclusion that elements in a route pattern

can occur non-consecutively in the original sequences. The relaxation of the consecutiveness constraint gives rise to a new issue. Suppose *A* and *Z* are two locations so far apart that no users who are currently visiting *A*, would make *Z* as their next destination. Here is a route $route_{userID}$=<*A …. Z*> that starts from *A*, passes a number of other locations, and finally ends at *Z*. By definition, <*A Z*> is a sub-sequence of $route_{userID}$, 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 *Z*, 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

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

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 appears 2 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.

With route patterns clearly defined, the first step in the strategy becomes a problem of sequential pattern mining problem. This is a well-developed and fruitful research area, with many efficient algorithms to choose from. We will come back to the discussion of sequential pattern mining methods in Chapter IV.

3.4. RECOMMENDATION RULES

This section focuses on the second step of the strategy. In this step, we need to connect the active user with previous users through route pattern matching, and then we need to work out a method to generate route recommendations from the matching route patterns.

The solution to the first issue 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 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 locations 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 location follows these criteria:

1. The location 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 location should occur after the pattern in the previous user route, otherwise the system will recommend the active user to go backward.
3. The location should not be too far (less than *maximal_distance*) from the last location of the pattern, since the active user would not take it if it were.

If such a location can be found, then appending this new location to the pattern will produce a new pattern, as it satisfy all requirements of route patterns, that it is a popular sub-sequence occurs in a significant number of existing routes, and the distance between every adjacent elements of which 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:

$rule = \langle s_{i1}\ s_{i2}\ \ldots\ s_{i(m-1)} \rangle \rightarrow s_{im}$

By the definition of *rule*, the antecedent $\langle s_{i1}\ s_{i2}\ \ldots\ s_{i(m-1)} \rangle$ is also termed as the left hand side – *LHS(rule*), which is to be used as the pattern to match active user route with; and the consequent s_{im} as the right hand side – *RHS(rule*), which is the recommendation to present to the user.

Given an active user route $route_{userID}=\langle s_{i1}\ s_{i2}\ \ldots\ s_{in} \rangle$, the *rule* is said to match or applicable to $route_{userID}$ if $origin(rule) \subseteq route_{userID}$ (*origin* is a function that returns the original route pattern for any given *rule*), and $s_{i(m-1)}=s_{in}$. The additional constraint $s_{i(m-1)}=s_{in}$ is necessary to avoid making out-of-context recommendations. For example, *A, B, C, D* are four locations on a street, with *A* on one end, and *D* on the other. A user has visited *A, C, D*. Without the additional constraint, his route <*A C D*> matches the rule <*A*>→<*B*>, therefore the system will suggest *B* as the next location, 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, and thus the system will be unable to make any recommendation at all.

This is well known issue called *cold start*, common among recommender systems. A simple solution is to recommend the locations 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.

3.5. RECOMMENDABILITY

For an active user, there could be a number of matching route patterns, leading to multiple recommendations. This is most common at the beginning of the 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 recommendation, which is in turn determined by how strong a connection exists between the rule that generates this recommendation and the active user.

To measure this connection, let us review how the connection comes into being. The connection is established between a rule and the active user, when this rule's LHS is contained in the active user's route. As such, the length of the LHS is a good indicator how similar the rule is with the active user route – the longer the LHS, the more traits of the rule occurs in the active user route, the stronger the connection, and therefore the more like-minded the active users is to the group of previous users that generate this rule. As such, given a function *length* which returns any sequence length, the recommendability of a rule can be

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

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

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 location, all the rules available for matching will all have their LHS being one-element long. To tell which rule is more recommendable than the other when they have the same LHS length, we can turn to the support (*sup*) of the route pattern behind each rule. 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* (Agrawal & Srikant 1995), and is defined as:

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

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 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 the work of Tseng and Lin (2006), which is defined as:

$strength(rule_I) = sup(rule_i) \times conf(rule_i)$

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

$$recommendability(rule_i) = bias\_similarity \times simularity(rule_i) + (1 - bias\_similarity) \times strength(rule_i)$$

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

CHAPTER VI

IMPLEMENTATION

Chapter III has identified the research problems, and provided general principles of solving the problems. Guided by these principles, this chapter will turn to the implementation of those solutions. An architectural design of the Bar Tour Guide system is presented at the beginning, as the backdrop for the later implementations. The rest of the chapter will evolve around the three major components revealed in the design.

4.1. SYSTEM ARCHITECTURE

4.1.1 HARDWARE COMPONENTS

The system has two major hardware components – the GPS capable handheld device, and a central server. The handheld device tracks the user's bar tour route, and preprocesses the GPS signal stream into a sequence of visited bars. It is also responsible to produce recommendations as which bar to visit next. The function of the central server is to collect bar tour routes from all handheld devices, and run sequential pattern mining methods to extract popular patterns. Figure 4 illustrates the hardware components:

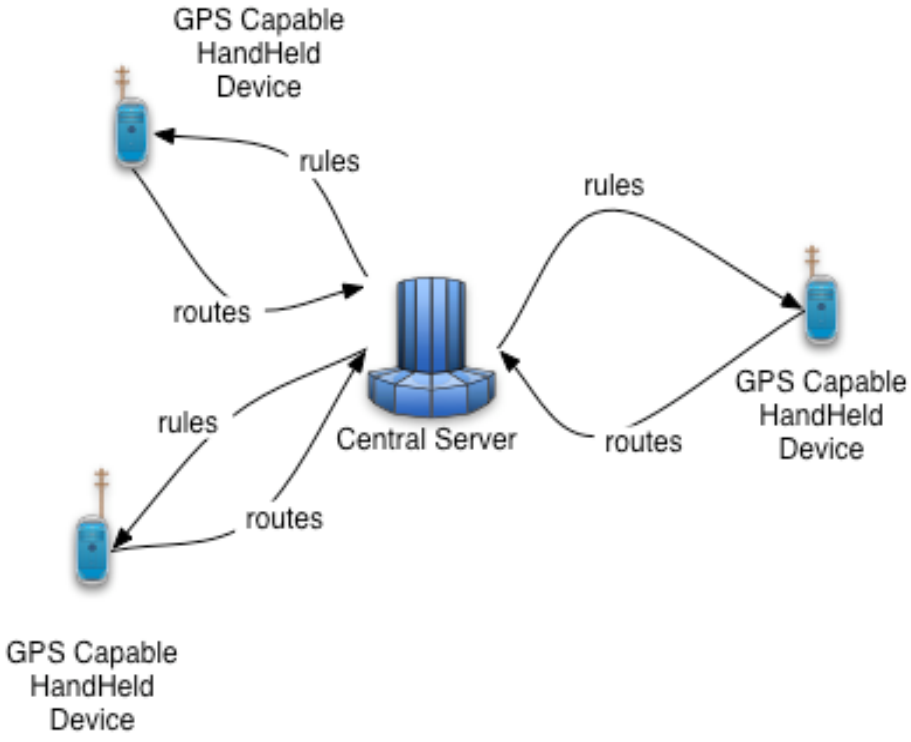


Figure 4. Hardware components

3.2.2 LOGICAL DESIGN

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 5:

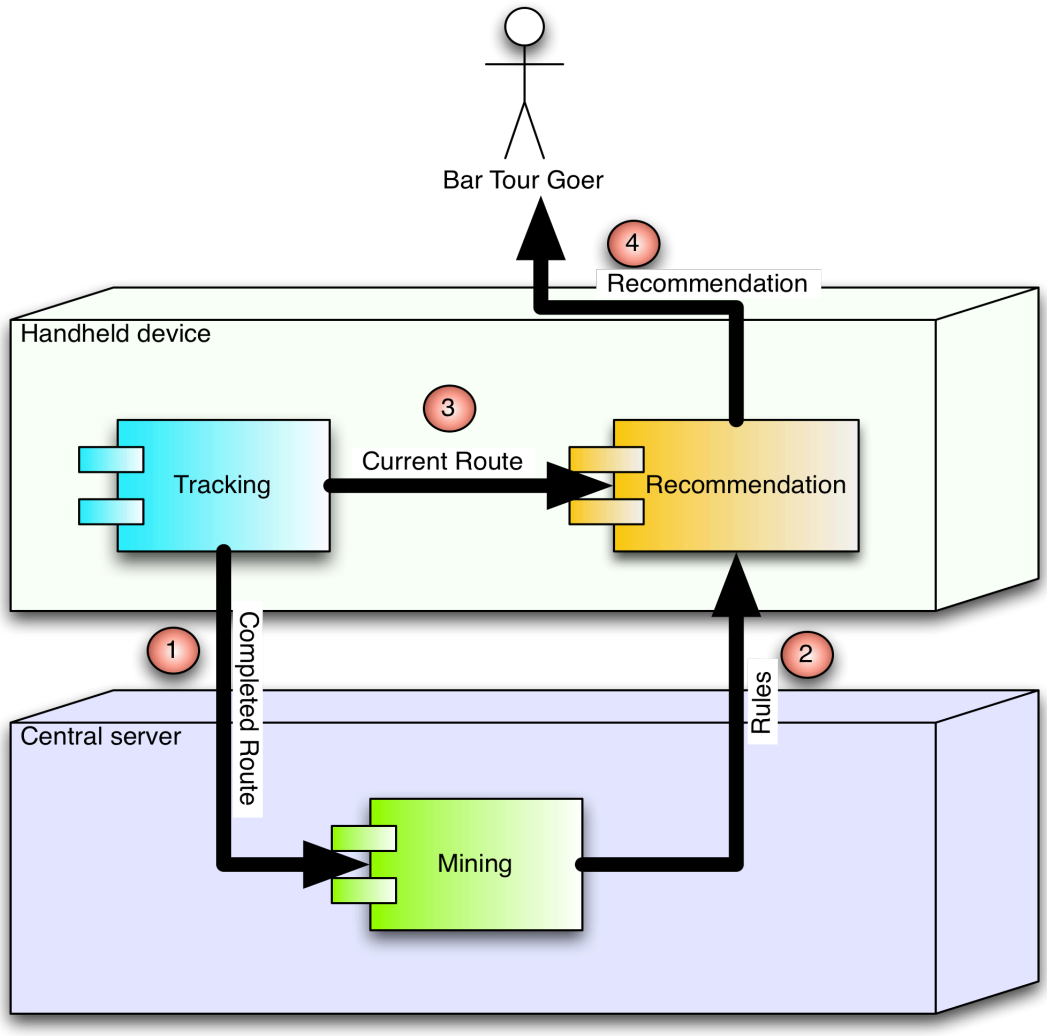


Figure 5. System architecture

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.

4.1.3 SYSTEM OPERATION FLOW

Joining them together, we can have the whole picture of the system operation flow (see Figure 6). When a user starts a new bar tour, his/her handheld device will download the latest set of recommendation rules from the mining module. Meanwhile, the tracking module starts recording a series of geo positions from the GPS sensor, and preprocesses them into routing data. When the user requests for recommendation, the recommendation module will take a snapshot of the current routing data from the tracking module, and produce recommendation with the help of the recommendation rules. At the end of the tour, the tracking module will submit the complete bar tour route to the mining module. As more application users finish their bar tours, more routes are feeding into the mining module. Then mining module will find a time of low traffic, and start mining for new sets of popular patterns, and update the set of recommendation rules, which are waiting for the next round of bar tours.

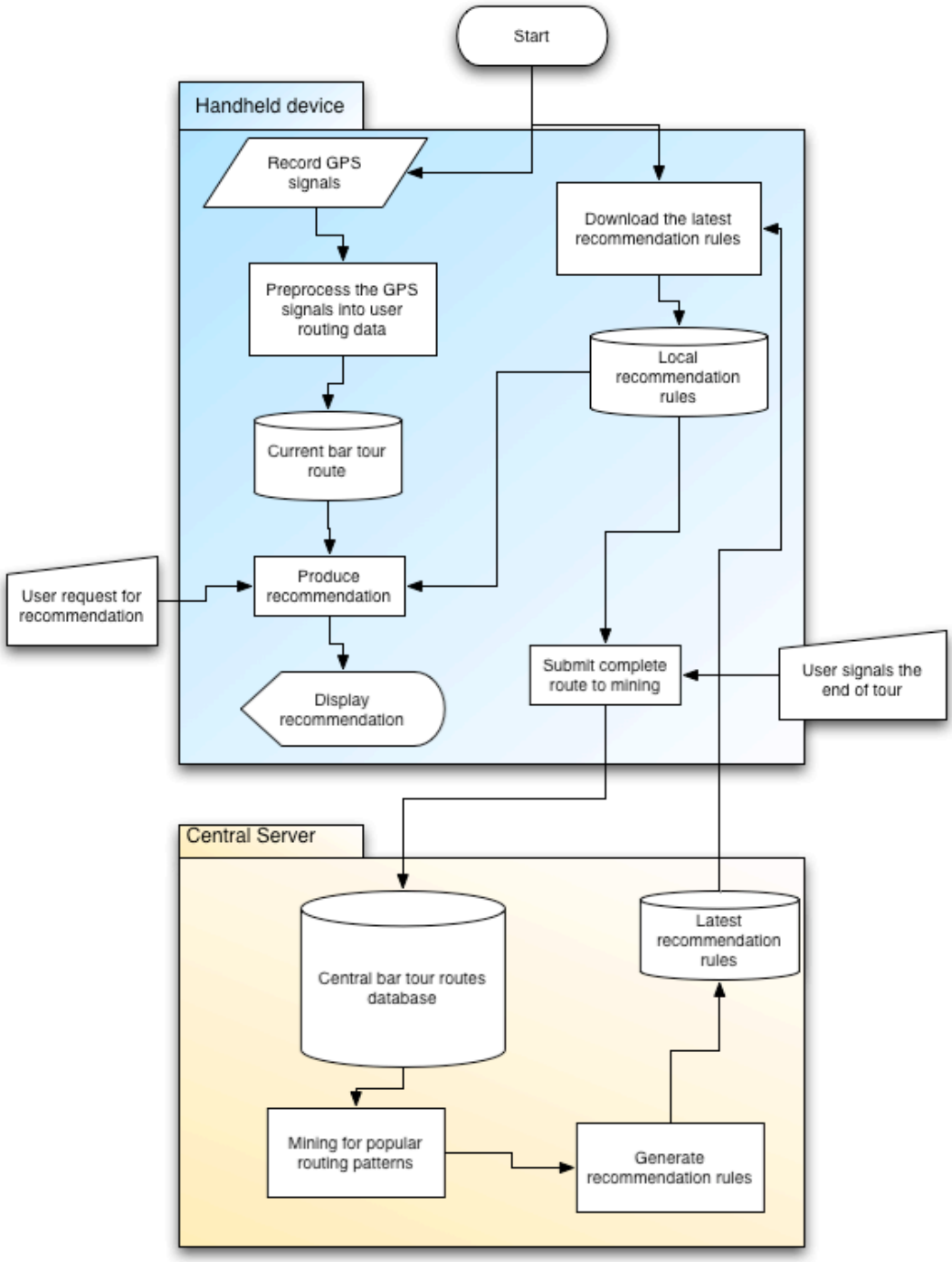


Figure 6. System operation flow

4.1.4. INITIAL OPERATION CONDITION

The above operation assumes the pre-existence of a large bar tour routes database, which is crucial in the generation of recommendation rules. The issue then becomes how to acquire a large database bar tour routes in the first place? The solution is to divide the launch of the application in two phases. In phase 1, the objective is to collect as many bar tour routes as possible, and build up a large routing database over time. Therefore many volunteer bar tour goers are encouraged to carry the Bar Tour Guide handheld devices, while they are having their night time tours. During this phase, only the tracking module is turned on, while the recommendation module is disabled temporarily. At the end of the tours, the routes are submitted to the central server. When the size of the central database hits a critical mass, the central server will start mining popular patterns over the database and generates recommendation rules. Once the rules are forming, the recommendation service will be turned on, and the application will be in full swing.

4.2. PREPROCESSING

The major function of the preprocessing module is to convert the GPS signal stream into a route, a sequence of locations. A GPS position recorded by a device is denoted as a 3-tuple *(latitude, longitude, t)*, in which *latitude* is the latitude value, *longitude* the longitude value. The third item *t* is the timestamp where the pair of *latitude, longitude* is sampled. A GPS signal stream is a series

of GPS position records sampled in various time intervals. On the other hand, a route is a sequence of significant locations. As defined above, a location is denoted as a 3-tuple (*latitude, longitude, name*), thus the conversion process is to find the significant spot (i.e. visited by the user) in the GPS signal stream, and map that spot to location names.

4.2.1. CHALLENGES

There are two major challenges in the preprocessing stage:

Whether the user has passed any of the significant locations: although both the GPS signal stream and the set of significant locations contain geo positions – the (*latitude, longitude*) tuples, it's unlikely to find the exact (*latitude, longitude*) of a significant location in any given GPS signal stream, for two reasons. First the target location is an area, not just a geo-spot. Second, commercial GPS devices have an accuracy of 1-10 meters (Snively, J., 2009). As such, the user may be standing on the exact spot pinpointed by the (*latitude, longitude*) of a significant location, but his/her GPS device will still give a reading off that spot.

Whether the user stays in that location, or just passes by: as a sub-issue, the system also needs to tell whether the user stays in that location long enough, to warrant a conclusion that the user does like this location.

4.2.2 SOLUTIONS

For challenge one, a simple solution would be to check if any GPS record from an active user's GPS signal stream falls within a certain distance of π to the geo-position of the center of any significant location. However, since different bar covers various area sizes, a fixed π is not going to work for all cases. A solution can be derived from the observation that most bars usually have only one main entrance, which any patron of the establishment will have to pass through. So by using the geo-position of the bar main entrance to geo representation of the establishment, a fixed value of π can be found to accommodate most cases.

Now that we know a user has been to a bar, but did he enter the bar or just passes by? Considering most bars are indoor establishments, the second challenge can be reduced to the problem of detecting indoor events from the GPS signal stream. Marmasse, N. and Schmandt, C. (2000) found a solution to this problem, by using loss of GPS signals to detect buildings (indoor event) in their work of the *comMotion* system. Their approach exploits a weakness of the GPS system. GPS devices rely on satellite signals tell where its current position is. With the satellite signals being blocked, such as the case of getting inside a building, the device will no longer work, resulting a gap in the GPS signal stream. By looking for a large time gap between two consecutive GPS records, one can assume the user has entered a building. Borrowing their idea, the preprocessing module will compare the timestamp of each pair of adjacent GPS records, and

calculate the interval. If the interval is found to be larger than a pre-defined t_{gap}, the GPS record before the gap will be tested for if matching any significant location. If the test result is positive, then a bar name will be resolved from the matched significant location, and the name will be appended to the user's routing sequence. Figure 7 shows how the discovery works.

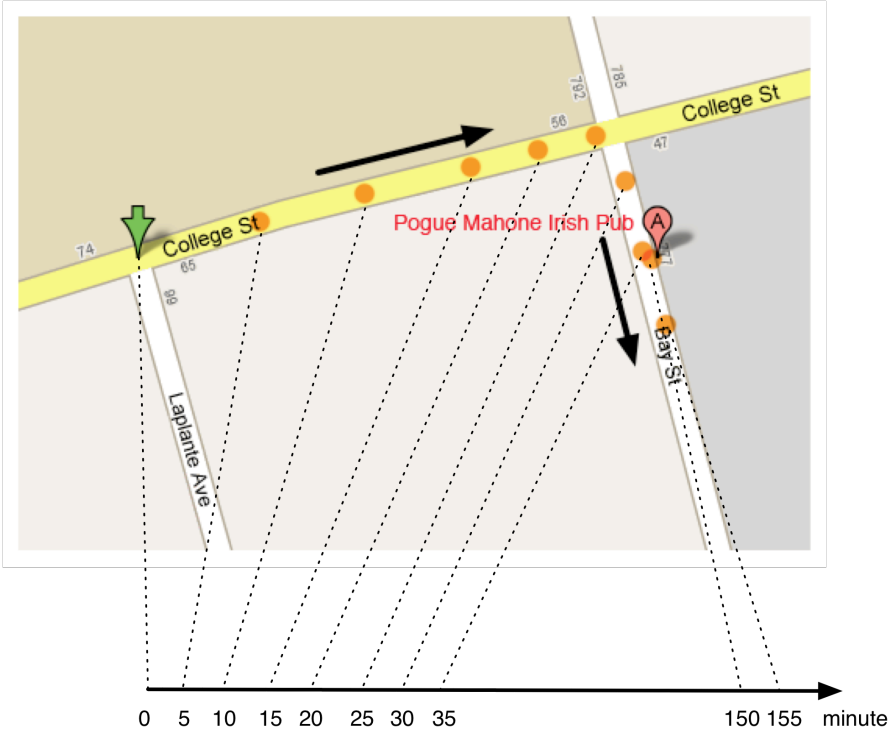


Figure 7. Discovering significant locations in the GPS signals stream

We can also exploit the time gap to gauge how much the user is interested in the establishment. Since the time gap indicates how long the user stays in the building, we can set t_{gap} high enough, so that user has to stay at least that long in a particular bar before this bar is being registered in user routing sequence. This

way, we can filter out the cases where the user pops into a bar, but then leave shortly after realizing he/she does not like it.

Here's an algorithm to convert GPS stream data into a user behavior pattern sequence:

```
Algorithm GPSToRoutingSequence
Begin
   Global user_route
   Global last_gps_record
   read current_gps_record from the GPS sensor
   if current_gps_record['time'] - last_gps_record['time'] >= t:
      current_location = is_a_bar(last_gps_record)
      if current_location is not a bar:
         exit
      else:
         append current_location to user_route
End
```

The algorithm will run incrementally, at each time interval when the device samples the GPS sensor. Both *last_gps_record* and *user_route* are declared global. The former remembers the last GPS position, which might be a bar entrance; the latter keeps track of the bar visited by the user.

The function *is_a_bar* will find the bar entrance nearest to the current GPS record. However, it is time consuming to check every user position against every bar's location, as the number of bars in a city can be huge. This problem can be converted to the problem of collision detection, a famous problem well researched and addressed in robotics, as a user wandering in a city on a bar tour is just like a robot bumping into fixed objects. There are a wealth of algorithms addressing this problem, such as Lin (1993). The time constraint does not allow further investigation of this issue, and it has to be left for future research efforts.

4.3. MINING

4.3.1. SEQUENTIAL PATTERN MING PROBLEM

As pointed out in Chapter III, the extraction of route patterns can be transformed into the sequential pattern mining problem. The transformation is required to resolve two differences.

The first difference is that, in the sequential pattern mining problem, the sequence element is a set of items. While in route pattern mining, the element is a single item – a location. The problem of sequential pattern mining was first identified by Agrawal & Srikant (1995), as an effort to satisfy the need from the retailing business to discover customer purchase patterns. In that setting, the element of the sequence is purchase transaction, and each of the transaction may contain several purchased items. Thus they define transaction as a set of items, denoted as ($i_1, i_2, \ldots i_n$), and sequence as an order list of item-sets. Accordingly the definition of sub-sequence is changed as follows:

A sequence <$s_{i1}' s_{i2}' \ldots s_{in}'$> is a sub-sequence of another <$s_{i1}\, s_{i2} \ldots s_{in}$> ($s_{ii}'$ and s_{ii} are both sets of items) if there exist integers $j_1<j_2<\ldots<j_n$ such that $s_1' \subseteq s_{j_1}, s_2' \subseteq s_{j_2}, \ldots s_n' \subseteq s_{j_n}$. For example, *<(ab)(d)>* is a sub-sequence of a sequence S_1=*<(ab)(cde)(e)>*, since (*ab*) is a subset of (*ab*), and (*d*) a subset of (*cde*), and they occur in the same order as in S_1.

This difference can be easily resolved. The bar tour goers are shoppers

shopping for drinking experience. They are just like super-store shoppers. The only difference is that super-store shoppers can shop multiple products in one transaction, while bar tour goers can only shop one bar at a time. Therefore visiting a bar is in fact a special case of purchase transaction, in that there is always only one item in any transaction. As such, the bar tour route can be seen as a sequence of 1-item transaction. Thus changing the route denotation from $<s_{i1}\ s_{i2}\ ...\ s_{in}>$ to $<(\ s_{i1})\ (s_{i2})\ ...\ (s_{in})>$, the routes will match the definition of sequence in sequential pattern mining, and the routes database can then be processed by any sequential pattern mining method.

The second difference is related to the *maximal_distance* constraint on the route patterns. Not surprisingly, none of the sequential pattern mining methods so far proposed in the literature handles this constraint, as they are more concerned about what happened in a sequence event, and when it happened, rather than where it happened. Therefore, we can re-use any of the existing sequential pattern mining method to extract popular route patterns, but not without minor extension to handle the *maximal_distance* constraint.

4.3.2. BRIEF SURVEY OF EXISTING MINING ALGORITHMS

Sequential pattern mining is computation and I/O intensive because such mining may generate and/or test a combinatorially explosive number of intermediate subsequences. The quest for efficient and scalable methods has been one of the driving forces behind the research on sequential pattern mining.

There are two major research directions: (1) efficient methods for mining the *full* set of popular sequential patterns, and (2) efficient methods for mining only the *set of closed* popular sequential patterns. Briefly, a sequential *pattern* is *closed* if there exists no sequential *pattern'* where *pattern'* is a proper super-sequence of *pattern*, and *pattern'* has the same frequency support as *pattern*. By mining only closed sequential patterns, the search space can be shrunk considerably.

However, it is desirable to obtain a full set of popular patterns for the working of the Bar Tour Guide application. When the user is on a bar tour, the user may request for recommendation at every stop of the route. Thus the system will be asked for recommendation for every possible route, i.e. every possible routing sequence, including non-closed routing sequences. Therefore, the set of only closed patterns will not be able to provide enough recommendation rules to cover as many cases as possible.

The major approaches for mining the full set of sequential patterns either directly or indirectly explore the Apriori property (Agrawal, Imielinksi, & Srikant, 1994). The property states the fact that every nonempty sub-sequence of a popular sequential pattern is a popular sequential pattern, that is, it must satisfy the minimum support. The Apriori property is anti-monotonic (or downward-closed) in that, if a sequence is not frequent, all of its super-sequences must not be frequent. Many sequential pattern mining methods rely on this property to prune the search space, and make the discovery of sequential patterns more

efficient.

Within the methods that discover the full set of frequent sequential patterns, there are four types of approaches: the candidate generate-and-test approach, the vertical format-based methods, the pattern-growth approach, and the incremental mining methods. The following gives a brief review of these approaches.

Candidate generate-and-test approach: This type of methods follow a common pattern of first growing a number of candidates from frequent *1-sequence*, and then test each of them against the minimum support. The process is then repeated to find frequent *n-sequence*, where *n>1*. GSP (Generalized Sequential Patterns), a sequential pattern mining method that was developed by Srinkant and Agrawal in 1996, is the most famous algorithm of this type of methods. Although GSP benefits from the Apriori pruning, it still generates a huge number of candidate sequences, requiring multiple scans over the database. Some of these candidates may not appear in the database at all, resulting in wasted time. A more detailed introduction can be found in the next section.

Vertical format-based methods: GSP uses the horizontal data format, where the data are represented as <*sequence_ID: sequence_event*>. As the name suggests, vertical format-based methods adopts the vertical data format, where the data becomes a set of tuples of the form <*sequence_event: (sequence_ID, event_ID)*>. That is, for a given event, the system records the

sequence identifier and the corresponding event identifier for which the event occurs. The *event_ID* of the *i*th event in a sequence is *i*. The benefit of using vertical data format is that, the system only requires one scan to find all frequent 1-sequences, and then builds patterns (i.e. frequent sequences) of longer length by simple joining. SPADE (Zaki, 1998) is the most successful algorithm of this approach. The name stands for Sequential Pattern Discovery using Equivalent classes. It is a great improvement over GSP as it reduces the number of scans over the sequence database. However, it shares the similarity with GSP in that, it still has to generate large sets of candidates, thus it suffers most of the same difficulties as occur in GSP.

Pattern-growth method: Pattern-growth is a method of frequent-pattern mining that does not require candidate generation. The general idea of this approach follows the divide-and-conquer framework. It partitions the sequence database in a number of smaller databases, called projected databases. The sequences in each of these databases share a common prefix pattern. The algorithm grows the prefix patterns, which it concatenates with suffix patterns in the corresponding projected databases to find frequent patterns, avoiding candidate generation. PrefixSpan (Pei, Han, Mortazavi-Asl, Wang, Pinto, Chen, Dayal and Hsu, 2001) is a typical example of this type of methods. According to Pei et al., PrefixSpan outperforms GPS and SPADE in most cases.

Incremental method: The above approaches for sequential pattern mining assume that the mining is performed in a static sequence database. However,

many real life sequence databases, such as the routes database for the Bar Tour Guide, grow incrementally. It is undesirable to mine sequential patterns from scratch each time when a small set of sequences grow, or when some new sequences are added into the database. Incremental algorithms were developed for sequential pattern mining so that mining can be adapted to incremental database updates. IncSpan (Cheng, Yan & Han, 2004) is a typical efficient algorithm for incremental mining of sequential patterns. It solves the challenge by buffering semi-frequent patterns. Semi frequent patterns are “almost frequent” patterns. They are likely to become frequent in the growing database. By keeping the additional information of semi frequent patterns, they system can quickly grow semi frequent patterns into frequent patterns when new sequences arrive.

Since the Bar Tour Guide users are both consumers of the recommendation service, and producers of historical routes for mining, the system’s database are growing incrementally. As such, the incremental method would be the perfect tool for mining route patterns.

4.3.3. GSP TO MINE ROUTE PATTERNS

In this section, GSP is used to demonstrate the process to extract popular route patterns from the route database. GSP is chosen here to facilitate the discussion for its simplicity and clarity. When in production, more efficient algorithm should be used.

As discussed above, we need to transform the routes into sequences of 1-item transactions, so that they can be fed into GSP for mining. The algorithm is outlined as follows. In the first scan of the database, it finds all 1-event sequence with minimum support. Each subsequent pass starts with a seed set of sequential patterns – the set of sequential patterns found in the previous pass. This seed set is used to generate new potentially frequent patterns, called candidate sequences. Each candidate sequence contains one more event than the seed sequential pattern from which it was generated. Recall that the number of instances of events in a sequence is the length of the sequence. So, all of the candidate sequences in a given pass will have the same length. A sequence with length k is referred to as a *k*-sequence. Let $Candidate_k$ denote the set of candidate *k*-sequences. A pass over the database finds the support for each candidate *k*-sequence. The candidates in $Candidate_k$ with at least minimum support produce $Frequent_k$, the set of all frequent *k*-sequences. This set then becomes the seed set for the next pass, *k*+1. The algorithm terminates when no new sequential pattern is found in a pass, or no candidate sequence can be generated.

To account for the *maximal_distance* constraint, the GSP algorithm needs to be adjusted slightly. Obviously the candidate generation is where the adjustment is needed, that is, when generating $Candidate_{k+1}$ from $Candidate_k$, a check is required to make sure the new sequences in $Candidate_{k+1}$ conform to the *maximal_distance* constraint. The following pseudo code illustrates the GSP

algorithm with the *maximal_distance* adjustment:

```
   Algorithm GSP-with-maximal-distance-constraint
   Begin
      Scan database once, find Frequent1
      Let Frequent1=[si|si⊂Θ, length(si)=1, sup(si)≥min_sup]
      Let k=1
      While Frequentk is not empty do
         Generate Candidatek+1 from Frequentk
         filter out the sequences that's not qualified for the
maximal_distance
         constraint
         If Candidatek+1 is not empty
            Let Frequentk+1=[si|si⊂ Candidatek k, sup(si)≥min_sup]
            If Frequentk+1 is not empty
               Filter out the sequences that's not qualified for the
maximal_distance
               constraint from Frequentk+1
         Let k=k+1
   End
```

4.4. RECOMMENDATION

The functioning of the recommendation module follows the scheme of making recommendations discussed in Chapter III. That is, the system will scan the recommendation rules, comparing each rule against the active user route, and identify a number of rules that match the active user route (i.e. the rule's *LHS* is contained in the active user route). Then it will sort the matched rules by their recommendability, and present the top-n results (the rule's *RHS*) to the users as recommendations. The scheme is very simple and straightforward, but the implementation of the scheme, particularly the rule matching step, has a serious scalability issue.

Recall that the recommendation rules are generated from all popular route patterns that can be extracted from the route database. The number of

recommendation rules can be huge, since 1) the number of bars in a city such as Toronto can be over one thousand, and their possible combinations resulting in routes are exponential; 2) a route pattern's every possible sub-sequences are patterns as well (according to the Apriori property), and each of them will be presented as recommendation rule. Scanning such a large number of rules can be a lengthy process. If the system cannot produce recommendation within seconds of the user request, it will be considered un-usable by most users.

There are two types of scanning, as illustrate in Figure 8. The first one is the scanning across each rule's *LHS* to check if they match the active user route. The second one is repeating the above process on every rule vertically.

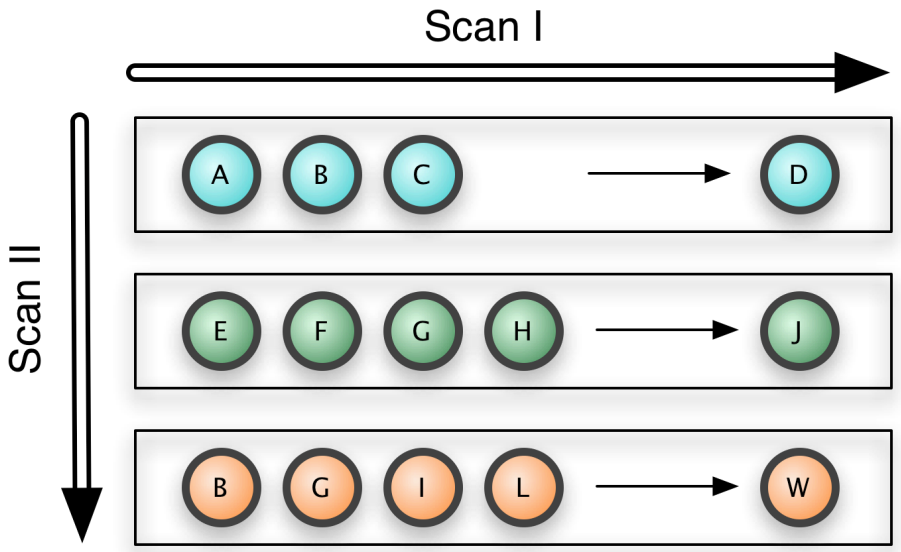


Figure 8. Two types of scanning

To speed up the first type of scanning, an incremental technique can be

used. It is based on the observation that, 1) the active user route is growing incrementally, with one bar location at a time, and 2) when the beginning part of a rule's *LHS* matches the beginning part of an active user route, there is a chance they will match completely in the future as the user progresses on the tour. With these observations, the system uses an array *cursors* to remember the matching progress of each rule. The index of *cursors* is the rule IDs, and its values are the latest matched positions in the *LHS* of the corresponding occurred rule. The use of cursors helps the system to avoid a lot of repetitive scanning. The following Figure 9 illustrates this process:

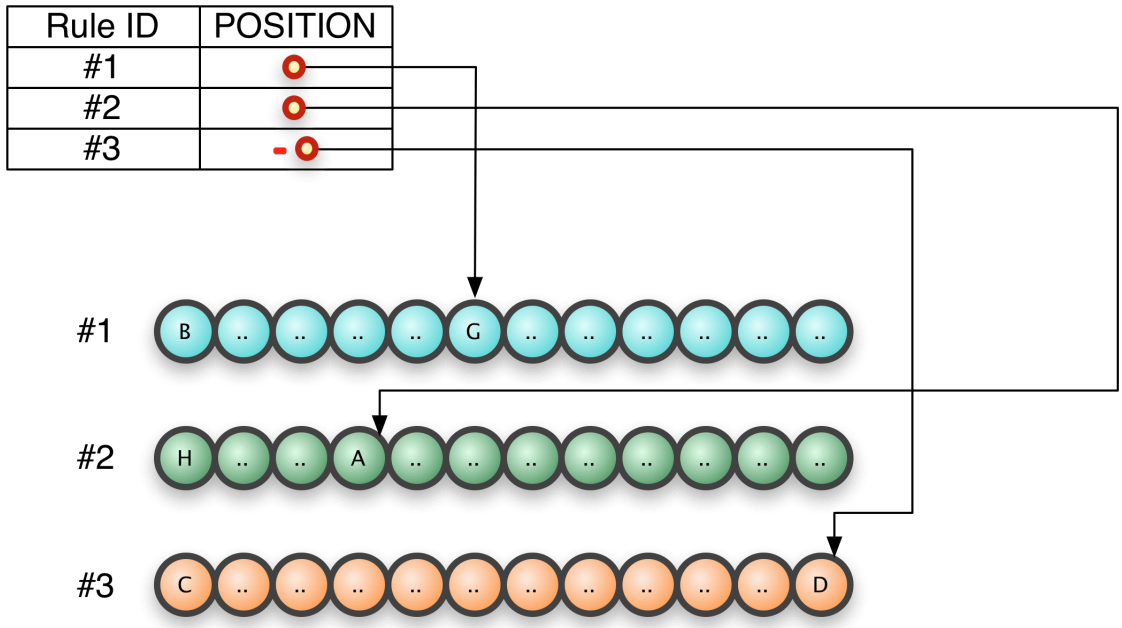
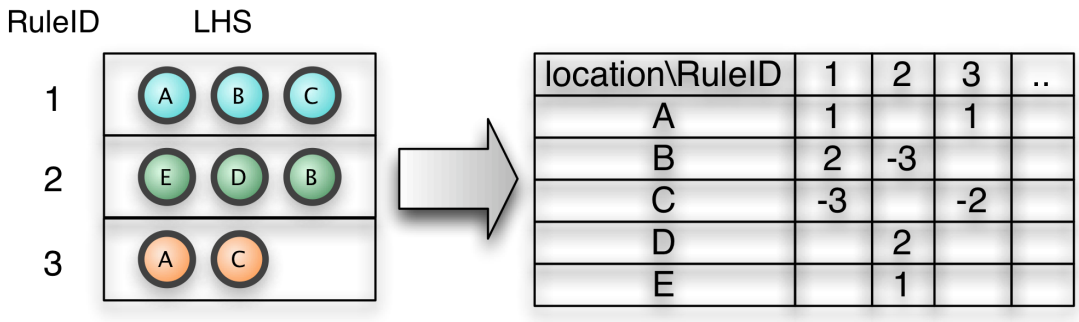


Figure 9. The array of *cursors*

The second type of scanning can be eliminated entirely, by the introduction of a new data structure of representing the recommendation rules. The new

technique takes inspiration from the vertical data format, and uses a two dimensional array *recommendation_rules* to store the recommendation rules. As illustrated in Figure 10, the first dimension of the array is the location, while the second dimension is the rule ID. Given a location, and a rule ID, the value they point to is the position where the location occurs in the corresponding rule's *LHS*. The negative value (such as the case of (*B*, 2), (*C*, 1), (*C*, 3) in Figure 10) indicates the end of the corresponding *LHS*.



location\RuleID	1	2	3	..
A	1		1	
B	2	-3		
C	-3		-2	
D		2		
E		1		

Figure 10. The array of *recommendation_rules*

With the help of *cursors* and *recommendation_rules*, the rule matching process works as follows (also shown in Figure 11): when the active user route has grown by one new location *C*, the array *recommendation_rules* is consulted to obtain a list of rules that contains this new location, and a list of positions the location occurs in these rules. And then the system compares the list of positions (in absolute value) with the positions in the *cursors* by the same rule IDs. If the position of the new location in a rule is greater than the corresponding position in the *cursors* by one, then a match progress is made (such as rule #3 and #4), and

the position held in *cursors* will be updated. If the position of the new location is also negative (rule #4), then the corresponding rule is found as a complete match, and will be appended to the recommendation results.

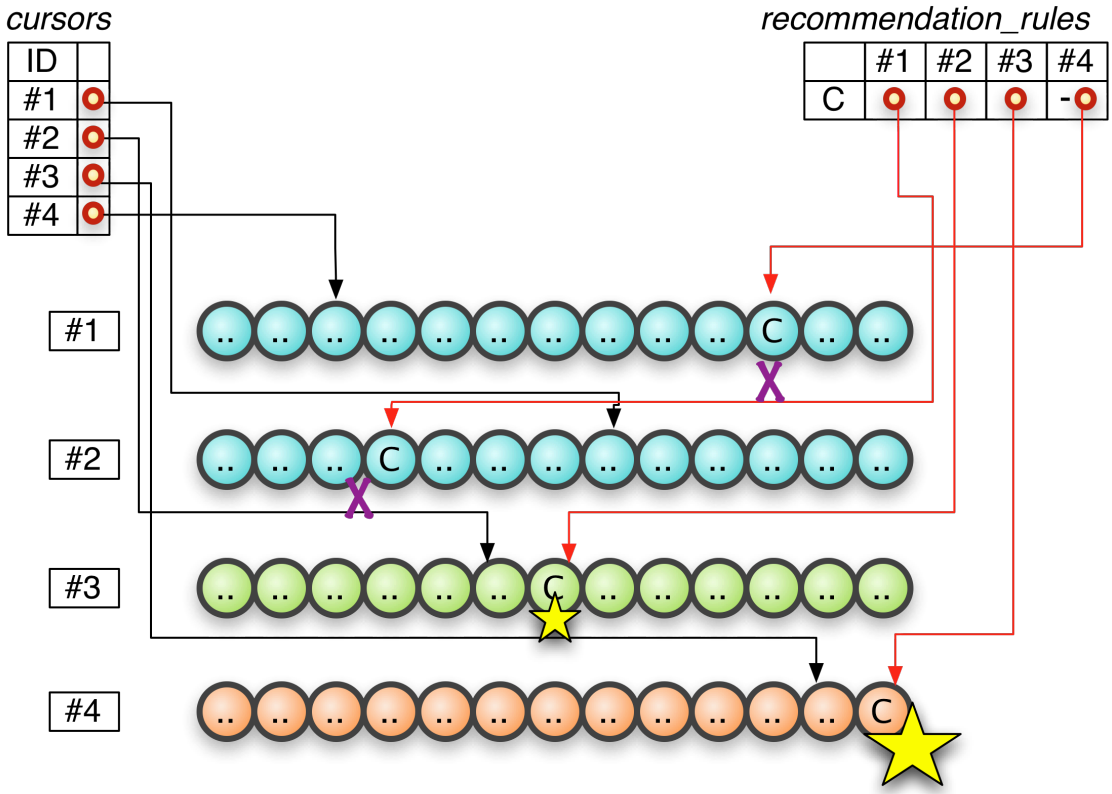


Figure 11. The incremental rule matching process

The following gives an incremental algorithm in pseudo code. The *recommendation_rules*, *cursors* and *recommendations* are global variables.

```
Algorithm FindRecommendation
Begin
   Read the latest location of the active user route -> location_ID
   recommendation_rules[location_ID] -> matched_rules
   For rule_ID in matched_rules:
      If absolute(recommendation_rules[location_ID, rule_ID])
      equals cursors[rule_ID]+1:
```

```
            # reaching the end of LHS
            If recommendation_rules[location_ID, rule_ID] < 0:
               recommendation.append(rule_ID)
            Else:
               cursors[rule_ID] = recommendation_rules[location_ID,
rule_ID]
   End
```

The variable *recommendation* holds the rules that matched the active user route so far. The system will then sort them by their recommendability according to the scheme discussed in Chapter III, and present the top-n results to the user.

Note that repetition will be common in the resulted recommendations, due to the containment relationship among some matched rules. Such relationship occurs because any sub-sequence of a pattern is still a pattern (due to the Apriori property), therefore a rule's sub-sequences are rules as well. When a rule $<s_{i1}\ s_{i2}\ \ldots\ s_{i(m-2)}s_{i(m-1)}>\rightarrow<s_{im}>$ matches an active user route, there is a chance we can find rules in the form of $<s_{i1}'\ s_{i2}'\ldots\ s_{ij}'\ s_{i(m-1)}>\rightarrow<s_{im}>$, where $j \leq m\text{-}2$, $<s_{i1}'\ s_{i2}'\ldots\ s_{ij}'>$ satisfies *maximal_distance* constraint, and $<s_{i1}'\ s_{i2}'\ldots\ s_{ij}'>\subseteq <s_{i1}\ s_{i2}\ \ldots\ s_{i(m-2)}>$, and these rules will certainly match the active user route. As a result, s_{im} will be presented as recommendation repeatedly. Therefore, after ranking the recommendations, the system needs to remove the repetitive recommendations of lower ranks.

CHAPTER V

EXAMPLES

This chapter simulates the three major processes with sample data.

5.1. PREPROCESSING

The preprocessing module translates the GPS signal stream to route, a sequence of locations. This section is going to demonstrate how to identify a significant location visit by the active user from the GPS signal stream.

Table I lists a number of GPS records read from an experiment along the College Street in downtown Toronto. Figure 12 marks the GPS readings on the map, with record #1 being marked as a green arrow.

Table 1. Sample GPS signals

Record ID	Latitude	Longitude	Time
1	43.66067579833994	-79.38660085201263	10:20am
2	43.6607184875732	-79.3864318728447	10:21am
3	43.66075923635847	-79.38622266054153	10:22am
4	43.660801925532425	-79.38601344823837	10:23am
5	43.66083103176996	-79.3858551979065	10:24am
6	43.66081356802914	-79.38579618930817	10:25am
7	43.66077087886346	-79.38578009605408	10:26am
8	43.66068938128109	-79.38573986291885	10:27am
9	43.660679679180575	-79.38574522733688	11:08am

10	43.66060788358808	-79.38571572303772	11:09am

Figure 12. GPS positions on the map

Suppose the set of significant locations has two items, *rule* = {*A, M*}, where the GPS position of *A*'s entrance is (43.66068367414535, -79.38573536785478), and the GPS position for *M* is (43.66097035334876, -79.38555479049683). Let π = 3 meters, that is, as long as the user appears within 3 meters to the pub's entrance, the user is considered passing the establishment. Also let t_{gap} = 30 minutes, that is the user has to stay in one location for at least 30 minutes so that the location is qualified as an element in the user's route. The objective of the preprocessing is to find out whether the user has visited any member of the set of

significant locations. The procedure runs incrementally on every GPS reading, carries out as follows.

At the beginning, the system reads the first GPS position – the record #1, and stores it in a variable *last_gps_record*. On reading the second GPS position, the system compares the new record with *last_gps_record*, resolving the time gap between the two records, which is one minute. And then the system checks if the $t_{gap} \geq$ *last_gps_record*. In this case, the outcome is negative as one minute < *last_gps_record* = 30 minutes. The system repeats the process on the subsequent records from number three till number eight.

On reading record #9, the system realizes the gap between nine and eight is 11:08 – 10:27 = 41 minutes, which is greater than t_{gap}. As such, the system reaches a conclusion that the user has been indoors for a while. Then the system checks record #8, the position before the user disappears for a while, to see if it is close to any location of the set of significant locations. By comparing to the set of significant locations *Λ*, we know that record #8 is 1.22 meters away from *A*, while 34 meters away from *M*. As a result, a conclusion is made that the user has been drinking in location *A* (which is the Pogue Mahone Irish Pub). And this pub becomes part of the user's bar tour route.

5.2. MINING

Figure 13 and Table 2 lists a number of sample bar tour routes, to help demonstrate the mining process for popular route patterns.

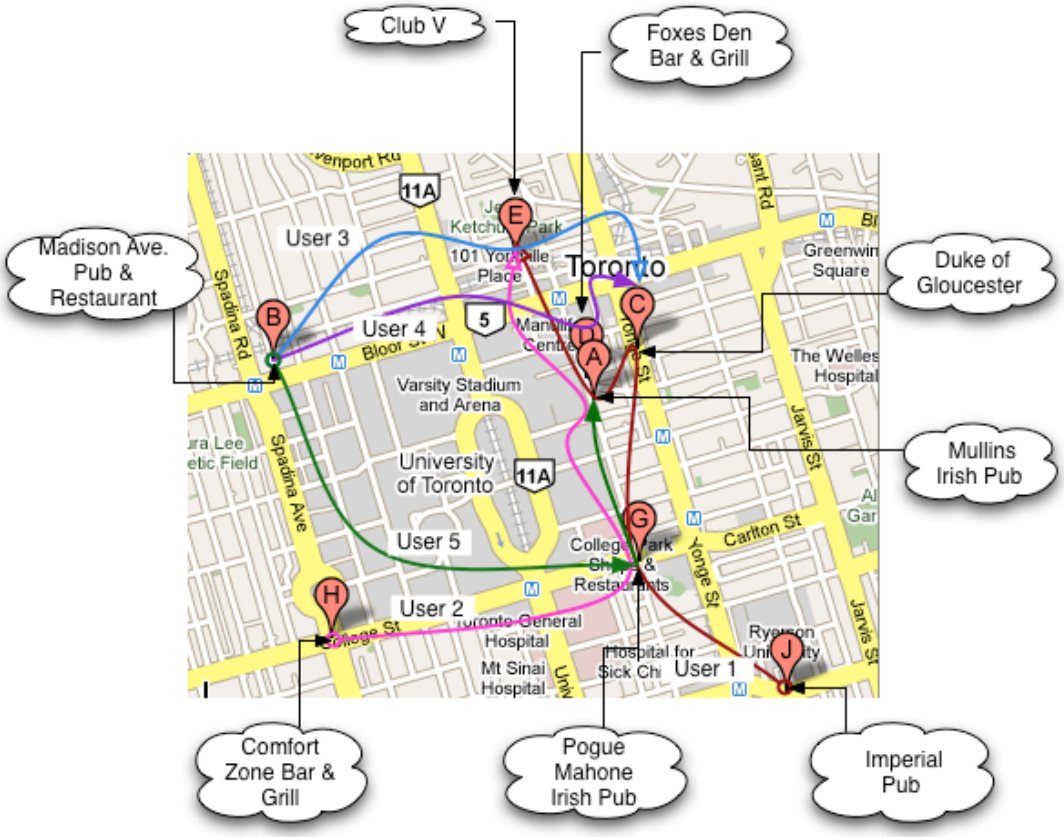


Figure 13. Sample bar tour routes

Table 2. Example routes database

User Id	Routes	Color in map
1	<*J G C A E*>	Brown
2	<*H G A D E*>	Pink
3	<*B E A C*>	Blue
4	<*B D A C*>	Purple
5	<*B G A*>	Green

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, i.e. route patterns of 1-stop long that have over minimal support:

Table 3. Popular 1-sequence patterns

Popular 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:

Table 4. Candidate 2-sequence patterns

Popular 1-sequence pattern	**Candidate 2-sequence pattern**
A	*A B* *A C* *A E* *A G*
B	*B A* *B C* *B E* *B G*
C	*C A*

	C B *C E* *C G*
E	*E A* *E B* *E C* *E G*
G	*G A* *G B* *G C* *G E*

Next, we calculate the supports of the 2-sequence candidates, and eliminate those fall below the minimal support:

Table 5. Popular 1-sequence & 2-sequence patterns

Popular 1-route pattern	**Popular 2-route pattern**	**Candidate support**	**Distance (km)**
A	~~*A B*~~	~~0%~~	1.22
	A C	40%	0.24
	A E	40%	0.55
	~~*A G*~~	~~0%~~	0.67
B	*B A*	60%	1.22
	B C	40%	1.40
	~~*B E*~~	~~20%~~	1.01
	~~*B G*~~	~~20%~~	~~1.56~~
C	~~*C A*~~	~~20%~~	0.24
	~~*C B*~~	~~0%~~	1.40
	~~*C E*~~	~~0%~~	0.57
	~~*C G*~~	~~0%~~	0.81
E	~~*E A*~~	~~20%~~	0.55
	~~*E B*~~	~~0%~~	1.01
	~~*E C*~~	~~20%~~	0.57

	~~*E G*~~	~~0%~~	1.22
G	*G A*	60%	0.67
	~~*G B*~~	~~0%~~	~~1.56~~
	~~*G C*~~	~~20%~~	0.81
	G E	40%	0.57

Note that <*B G*> and <*G B*> do not satisfy the *maximal_distance* constraint. So even after the database gets expanded, and they become popular sub-sequence, they do not qualify as route patterns. Following the same process, we can find the only popular 3-sequence pattern:

Table 6. Popular 3-sequence patterns

Popular 1-sequence pattern	**Popular 2-sequence pattern**	**Popular 3-sequence pattern**	**Support for 3-sequence**
A	*A C* *A E*		
B	*B A* *B C*	*B A C*	40%
G	*G A* *G E*	*G A E*	40%

Without finding further popular patterns, the algorithm soon stops. At this point, we have found all popular route patterns in the database:

<A>,<B>,<C>,<G>,<E>,<A E>,<B C>,<G A>,<G E>,<B A C>,<G A E>

5.3. MAKING RECOMMENDATIONS

By definition, all 1-route patterns cannot be turned into recommendation

rules, and therefore discarded. The rest of the patterns can generate the following set of recommendation rules

Table 7. Recommendation Rules:

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

With the above set of recommendation rules, we are ready to make recommendations. Suppose Maz, has started his bar tour night. So far he has visited a number of bars: <*J G C*>, and he is just coming out of *A*. He's having a good time, quite satisfied with the recommendations. So he pulls out his GPS capable cell phone, and check what can be the next interesting hop.

First, we need to transform the recommendation rules into a 2-dimensional array storage, as follows:

Table 8. Array of *recommendation_rules*

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

When Maz was hopping through the route <*J G C*>, i.e. the route before reaching bar A, the system has been matching his route against the *recommendation_rules* on every location he visited. Without going into too much detail, the *cursor* array has the following values at this moment:

Table 9. Array of *cursor*

Rule ID	Position
1	1
2	1
3	1
4	0
5	0
6	0
7	0

As Maz coming out of bar *A*, the system consults the *recommendation_rules* table, and obtain the list of rules containing *A* – rule #1, #4, #6, and #7. By joining the two arrays on rule ID, we have the following result:

Table 10. A joined table from the *cursors* and the *recommendation_rules* on rule ID

Rule ID	Cursor Position	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 neither of them contains location *A*, the rule #2, #3 and #5 can be ignored. The position of *A* in rule #1 is −2, the absolute value of which is greater than the cursor value of rule #1 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-1 location, and set *bias_similarity* to 0.7. Let us first work out the similarity of both rules to Maz's route:

Table 11. Function *simularity(rule)*

Eligible Rules	***length(LHS(rule))***	***length(route$_{userID}$)***	***simularity(rule)***
<*A*>→*C*	1	4	0.25
< *A* >→*E*	1	4	0.25
<*G A*>→ *E*	2	4	0.5

Then we work out the strength of each rule:

Table 12. Strength of the rules

Eligible Rules	**LHS support**	**Rule support**	**Confidence**	**Strength**
<*A*>→*C*	100%	40%	0.4	0.16
< *A* >→*E*	60%	60%	1	0.6
<*G A*>→ *E*	60%	40%	2/3	0.27

Putting them together, we can obtain the recommendability of each rule as follows:

Table 13. Recommendability of the rules

Rules	similarity	bias_similarity	1-bias_similarity	strength
<*A*>→*C*	0.25	0.7	0.3	0.22
<*A*>→*E*	0.25	0.7	0.3	0.35
<*G A*>→ *E*	0.5	0.7	0.3	0.43

As a result, <*G A*>→ *E* is more recommendable than <*A*>→*E*, and then <*A*>→*C*. Getting rid of the *LHS* of each rule, we are left with the recommendation <*E, E, C*>, which can be further reduced to <*E, C*>, after removing the repetitive recommendation *E* of lower ranking. As Maz only wants the top-1 recommendation, the final output is *E*.

CHAPTER VI

CONCLUSIONS AND FUTURE WORK

This essay 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. And 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.

Sequential pattern mining is the pivotal process in the solutions to this problem. Currently GSP is being employed to mine sequential patterns. However, with its inheriting limitation, GSP does not scale well to meet any realistic demand, such as the demand from the Bar Tour Guide application. Therefore, an urgent need is required to adopt a more advanced sequential mining method for behavior pattern extraction. Given the incremental nature of the routes database used in the Bar Tour Guide, incremental sequential pattern mining methods, such as the ISM algorithm (Parthasarathy, Zaki, Ogihara, & Dwarkadas, 1999), and the IncSP algorithm (Lin & Lee, 2004), are the most helpful candidate replacements. Therefore, future research should look into how to extend existing incremental

sequential pattern mining methods to honor the *maximal_distance* constraint, and apply them into the behavior pattern extraction process.

More work is also needed in the pre-processing phase. As pointed out in Chapter 4, finding which bar the user has visited (the function of *is_a_bar*) is a time consuming process. Future research should look to collision detection methods to speed up this process.

The current application limits to bar touring. Any non-bar locations visited by the users are ignored by the system. It would find more useful applications if it can recognize and categorize other types of visited locations, such as restaurants, boutiques, café, flower shops, etc. This would be of great research values.

REFERENCES

- Agrawal, R., Imielinksi, T. & Srikant, R. (1994). Mining association rules between sets of items in large database. In *the Proceedings of the ACM SIGMOD Conference on Management of Data,* (SIGMOD 1993), May 26-28, 1993, Washington, D.C, 207 – 216.

- Agrawal, R. & Srikant, R. (1995). Mining sequential patterns. In *the Proceedings of the Eleventh International Conference on Data Engineering*, (ICDE 1995), Taipei, Taiwan, Mar. 6-10, 1995, 31 – 60.

- Balabanovic M. & Shoham, Y. (1997). Fab: Content-Based, Collaborative

Recommendation. *Communications of the ACM, 40*(3)*,* 66 – 72.

- Billsus, D., Brunk, C.A., Evans, C., Gladish, B. & Pazzani, M. (2002) Adaptive Interfaces for Ubiquitous Web Access. *Communications of the ACM*, *45*(5), 34 – 38.

- Cheng, H., Yan, X. & Han, J. (2004). IncSpan: incremental mining of sequential patterns in large database. In *the Proceedings of the tenth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, (KDD 2004), August, 22 – 25, 2004, Seattle, WA, USA, 527 – 532.

- Dijkstra, E. W. (1959). A note on two problems in connexion with graphs. *Numerische Mathematik, 1*, 269 – 271.

- Goldberg, D., Nichols, D., Oki, B.M., & Terry, D. (1992). Using Collaborative Filtering to Weave an Information Tapestry. *Communications of the ACM, 35*(12)*,* 61 – 70.

- Goldberg, K., Roeder, T., Gupta, D. & Perkins, C. (2001). Eigentaste: A Constant Time Collaborative Filtering Algorithm. *Information Retrieval, 4*(2)*,* 133 – 151.

- Hart, P. E., Nilsson, N. J., & Raphael, B. (1968). A Formal Basis for the Heuristic Determination of Minimum Cost Paths. *IEEE Transactions on Systems Science and Cybernetics, 4*(2)*,* 100 – 107.

- Kanoh, H. & Hara, K. (2008). Hybrid genetic algorithm for dynamic multi-objective route planning with predicted traffic in a real-world road network. In *the Proceedings of the 10th Annual Conference on Genetic and Evolutionary Computation*, (GECCO 2008), July 12 - 16, 2008, Atlanta, GA, USA, 657 – 664.

- Lang, K. (1995). Newsweeder: Learning to Filter Netnews. In the *Proceedings of the 12th International Machine Learning Conference*, (ML 1995), July 9, 1995, Tahoe City, California, USA, 331 – 339.

- Lin, M.-Y., & Lee, S.-Y. (2004). Incremental Update on Sequential Patterns in Large Databases by Implicit Merging and Efficient Counting. *Information Systems*, 29(5), 385-404.

- Lin, M.C. (1993). *Efficient Collision Detection for Animation and Robotics*. Unpublished Doctoral Dissertation. University of California, Berkeley.

- Linden, G., Smith, B., & York, J. (2003). Amazon.com Recommendations: Item-to-Item Collaborative Filtering. *IEEE Internet Computing*, *7*(1), 76 – 80.

- Marmasse, N. & Schmandt, C. (2000). Location-aware information delivery with ComMotion. In *the Proceedings of the 2nd International Symposium on Handheld and Ubiquitous Computing*, (HUC 2000), September 25-27, 2000, Bristol, UK, 157 – 171.

- Miller, B.N., Albert, I., Lam, S.K., Konstan, J.A. & Riedl J. (2003). MovieLens Unplugged: Experiences with an Occasionally Connected Recommender System. In *the Proceedings of the 8th International Conference on Intelligent User Interfaces*, (IUI 2003), January 12 - 15, 2003, Miami, Florida, USA*,* 263 – 266.

- Nanayakkara, S.C., Srinivasan, D., Lai Wei Lup, German, X., Taylor, E., & Ong, S.H., (2007). Genetic Algorithm based route planner for large urban street networks. *IEEE Congress on Evolutionary Computation,* (CEC 2007)*,* September 25-28, 2007, Singapore, 4469 – 4474.

- Park, K., Bell, M., Kaparias, I. & Bogenberger, K. (2007). Learning user preferences of route choice behaviour for adaptive route guidance. *IET Intelligent Transport Systems, 1*(2)*,* 159 – 166.

- Parthasarathy, S., Zaki, M. J., Ogihara, M., & Dwarkadas, S. (1999). Incremental and interactive sequence mining. In *the Proceedings of the 8th International Conference on Information and Knowledge Management*, (CIKM 1999), November 2-6, 1999, Kansas City, Missouri, USA, 251 – 258.

- Pei, J., Han, J., Mortazavi-Asl, B., Wang, J., Pinto, H., Chen, Q., Dayal, U., & Hsu, M.-C. (2001). Mining sequential patterns by pattern-growth: the PrefixSpan approach. *IEEE Transactions on Knowledge and Data Engineering, 16*(11)*,* 1424 – 1440.

- Rogers, S. & Fiechter, C. (1999). A Route Advice Agent that Models Driver Preferences. In *the Proceedings of the 1999 AAAI Symposium*, (AAAI 1999), March 22-24, 1999, Palo Alto, California, USA, 106 – 113.

- Shani, G., Brafman, R. I., & Heckerman, D. (2005). An MDP-Based Recommender System. *Journal of Machine Learning Research, 6*(2), 1265 – 1295.

- Snively, J. (2009). GPS Accuracy - How Accurate is it? Retrieved February 25, 2009, from the World Wide Web: http://www.maps-gps-info.com/gps-accuracy.html

- Srikant, R., & Agrawal, R. (1996). Mining sequential patterns: Generalizations and performance improvements. In *the Proceedings of the International Conference on Extending Database Technologies*, (EDBT 1996), March 25-29, 1996, Avignon, France, 3 – 17.

- Stentz, A. (1994). Optimal and efficient path planning for partially-known environments. In *the Proceedings of the 1994 IEEE International Conference on Robotics and Automation*, (ICRA 1994), 8-13 May 1994, San Diego, CA, USA, 3310-3317.

- Terveen, L., Hill, W., Amento, B., McDonald, D. & Creter, J. (1997). PHOAKS: A System for Sharing Recommendations. *Communications of the ACM, 40*(3), 59 – 62.

- Tseng, V. T., & Lin, K. W. (2006). Efficient mining and prediction of user behavior patterns in mobile web systems. *Information and Software Technology, 48*(6), 357 – 369.
- Yang, Z. & Kitsuregawa, M. (2005). LAPIN-SPAM: An Improved Algorithm for Mining Sequential Pattern. In *the Proceeding of the 21st International Conference on Data Engineering Workshops*, (ICDE 2005), 05-08 Apr., 2005, Tokyo, Japan*,* 1222 – 1222.
- Zaki, M. J. (1998). Efficient enumeration of frequent sequences. In *the Proceedings of the 7th International Conference on Information and Knowledge Management*, (CIKM 1998), November 3-7, 1998, Bethesda, Maryland, USA, 68 – 75.

APPENDIX A

SYMBOLS

LOWER CASE ALPHABET

UPPER CASE ALPHABET

LATIN SYMBOLS

FUNCTIONS