2015

Understanding and identifying useful patterns in location-based software

Tianyu Meng
Iowa State University

Understanding and identifying useful patterns in location-based software

by

Tianyu Meng

A thesis submitted to the graduate faculty

in partial fulfillment of the requirements for the degree of

MASTER OF SCIENCE

Major: Computer Science

Program of Study Committee:
Leslie Miller, Major Professor
David Weiss
Wensheng Zhang

Iowa State University

Ames, Iowa

2015

TABLE OF CONTENTS

Page

LIST OF FIGURES

Page

LIST OF TABLES

Page

ABSTRACT

As mobile devices have become more and more popular, we are seeing more location-based applications on handheld devices. Since the screen size of a handheld device is very limited, using a map on such kind of devices can be quite difficult. To make these applications easier to use, we want to use a situation model to provide user a better service. In this paper, we analyze user map operations and try to identify some useful patterns that have the potential to result in a significant saving of some unnecessary user operations when used in a situation-based system. Two user studies which are quite different are involved in this research. One is used to understand patterns and another one is used for validation. The result shows that the patterns identified in first user study are still valid in the second user study.

CHAPTER 1. INTRODUCTION

As mobile devices have become more and more popular, we are seeing more location-based applications on handheld devices. Since many of these devices have screens with a very limited size, maps in these applications can be difficult to work with. To make the existing location-based applications easier to use, we build on Ming's situation model [8], which has the potential to improve user's ability to deal with map operations in this environment.

Our current research focuses on understanding the difficulties when a user uses location-based software on a handheld device. We are interested in finding out when a map user is in trouble by identifying map operation patterns. To look at this question, we have analyzed the results of a user study our group did in 2012 and validated our results with a second user study.

The main contribution of this paper is that we were able to identify a set count patterns and show these patterns have the potential to result in a significant saving of unnecessary user map operations.

The remainder of the paper is organized as follows: Chapter 2 introduces some related work. Chapter 3 briefly reviews Ming's situation model. Chapter 4 briefly describes the first user study and the count pattern we discovered. Chapter 5 is the result of our evaluation. Chapter 6 provides a discussion of the result. Chapter 7 is a validation or our result. Finally we have conclusions and future work in Chapter 8.

CHAPTER 2. RELATED WORK

To make an application easier to use, software developers try to guess user's intention by which we can provide a better service to the user. So a very important question we need to answer is: "what the user want?" We try to find a way to know what the user want when a user using a location-based application. Adaptive systems focus on analyzing these types of questions.

The effectiveness of adaptive interfaces depends on the quality of the algorithms that predict user preferences/differences. Benyon [17] said the development of such an algorithm can be based on careful analysis of the user preference/differences space. Langley [18] and Billsus [19] said it can be based on machine learning algorithms. Tsandilas and Shraefel [20] have examined the accuracy of algorithms for predicting user performance and satisfaction. Nguyen and Sobecki [21] think managing the results of predicting user preferences/differences can be built into the interface software or managed as user/interface profiles.

Variations of the user interface itself can consist of different ways of providing the software's functionality and/or different representation formats for instructions/data presented on the screen. Shneiderman [22] has examined the effectiveness of using dynamic menus as a means of simplifying the complexity of using a menu. Benyon [23] investigated the use of different interfaces based on functionality for querying databases. His interface supported both a command line interface and a menu-driven interface. By understanding a user's level of spatial ability, his system presented the user with the interface format best suited to his/her abilities. The result was that users with high spatial ability had higher performance when presented with the command line format, and users with low spatial ability performed better when given the menu format.

Viano et al. [24] is an example of an adaptive interface that focuses on changing what users see. They developed an adaptive process controller interface that takes into account the state of the process and the state of the user. Under normal conditions (user and process are operating at normal levels) the interface presents the user with the normal amount of information. As either the user or the process show signs of stress, the system adjusts the information the operator sees.

A new approach to adaptive systems is adaptive systems that look at the use of situations. Findlater et al. [4] have compared adaptive to adaptable and static menus. Gajos et al. [5] have looked at the design space of adaptive GUIs. Shankar et al. [12] have explored adaptive user interfaces from a context point of view. But as far as we know, there is no work about using situation based system directly into location-based software.

Ming [8,9] have looked at situation-based systems in the context of user intentions. We will have a quick review in Chapter 3 to introduce Ming's situation model.

Bratman [2] thinks the user intention is the mental states that can motivate user's action. Sheer [13] provides another view that he thinks intentions is a course of actions. In his research he tries to explain we could capture one's intention by analyzing his action sequence. This approach is very close to our goal that we try to use situation based system to understand one's intention.

CHAPTER 3. SITUATION MODEL

In Ming [8], a situation is defined as having three parts: a set of user actions (a pattern), system context, and service. (i.e. a situation S = (pattern, context, services).) Based on Ming's definition, we will provide a more detailed look at the use of situations in location-based software.

A "Pattern" is a set of characteristic user operations from which we can guess the user's mind, status, or intention during his interaction with system. It could be some unusual operation, count of the operations, a specific series of actions, or even the distinctive time interval between some user actions. A "Pattern" is the information that we can get from users and by analyzing it we can get a better idea of the user's needs. In location-based software, a pattern consists of some specific or meaningful map operations.

The "Context" means the environment that the user interacting with. It contains the user's current state in the system, the function that the system provided to the user at this state, the data around these functions, and the potential goal that user may want to reach. User's operations are done "in" the "Context" to meet some potential goals that the "Context" provided. Some of these user operations may be picked out as "Patterns" for that "Context". In a location-based application, the context could be something like a map interface which the user is working with.

A "Service" is the way that the system responds to help the user. "Service" is a kind of feedback which will make the interaction between user and system much easier. It may be some advice or help message or even some automatic operation that can help user navigate the "Context". A "Service" will change when there are different "Pattern" and "Context" values. In a location-based application, we could simply reset the map as a simple example of a service. Other potential services might make use of the pattern to change the user interface to make the map easier to manipulate.

All of this together forms the "Situation". A Situation-based map system aims at making the system more user-friendly by detecting different "Patterns" of map operations based one "Context" then gives a proper "Service".

In this paper, we are focus on "Pattern". Several kinds of "count patterns" will be introduced in next chapter. "Count Patterns" are the patterns that make use of a count of operations, which may imply extra map operations in the result. Our study shows that if the patterns could be found early enough in the sequences of map operations, it has the potential to result in significant savings in terms of the number of map operations performed.

CHAPTER 4. METHODS

4.1 User Study

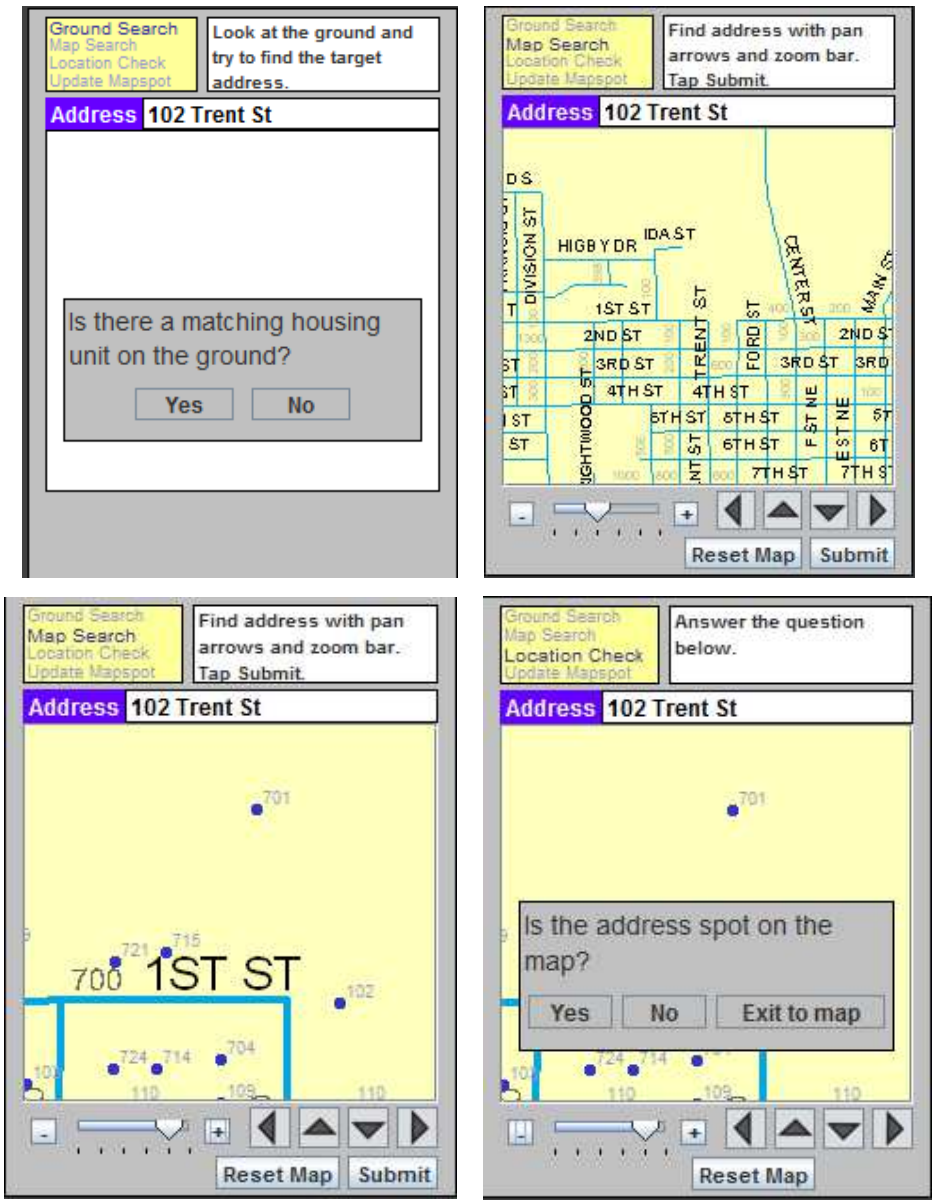
The key point of this work is to determine useful patterns for the situation model. To find potential patterns, we use the data from a user study our group did in Spatial Ability and Map-Based Software Applications [3]. In the user study, 35 people from community were asked to use location-based software to perform address verification tasks. The address verification tasks require a field worker to locate the address in the real world (if it exists) and check the map to make sure that the map matches the real world. The software task requires that the field worker should be able to find the target address on the map. In case that the address doesn't exist in the real world, the user finds the location where the address should be on the map and makes sure the map doesn't include the address. The software used in the study provides a photo with the ground situation and a map of housing units on the ground with the corresponding information. A housing unit on the map will be represented as a spot with its house number assigned to it. There are several possible results: 1) the ground situation is correctly reflected in the map and no further action is needed; 2) there is an error of commission on the map and a delete action is required to remove a spot on the map; 3) there is an error of omission on the map and an insert action is needed to add the spot to the map; 4) there is an error of housing unit location and the spot on the map need to relocated. We call each task of completing the verification of one address a scenario.

Figure 1 shows a snapshot of the software for both the guided and unguided versions. The software has a map and several buttons. This experiment has 10 verification scenarios and each of the scenarios will give a photo of the real world for each side of the street and ask the user to evaluate whether the location of a specific address on the ground is properly represented on the map. To successfully finish the task for each scenario, user need to do following steps in sequence: 1) find the address on the ground (i.e., in the photos presented to the subject), 2) locate the address on the map, 3) answer a question posed by the software as to whether or not the address was on the map, 4) if so, answer a question posed by the software as to whether or not the address was in the correct location on the map, and 5) fix the map if an error was identified

During the experiment, the user is asked to sit in front of two monitors that showing the photo of two sides of the street (Figure 2). The motivation for using this approach was to maximizing the user's attention on the software. In particular this setup removed the navigation associated with address verification. The software will record a log file containing the following information: the time it takes the user to finish each task; the number of scenario involved; user actions. Two treatments, "guided" and "unguided", are used in the experiment. The "guided" interface has some tips on the top of the interface to indicate the general information of next step

to finish the task. There are no tips on the top if it is “unguided”. 17 of 35 people in the experiment are treated as guided and 18 of 35 people are treated as unguided. The map will be reset when a scenario is completed.

In this research, we focus on the user actions in the log file. The software will record all the user actions when a user using it. The actions includes: (1) User does any operations on the map, such as pan, zoom, or click on the map. (2) User clicks any other buttons to do submit, add, reset, or delete. When a user finish the experiment, the software will output a log file which records all the operations the user did during the experiment. Figure 3 is an example of the content of a log file.



a) Guided user interface.

Figure 1. The interface of the user study.

Figure 1. a) Guided user interface. (continued)

b) Unguided user interface.

Figure 1. The interface of the user study. (continued)

Figure 2. The experiment of the user study.

```
Starting master log on Tue Jun 17 12:17:57 CDT 2008
************FileLoggerSingleton: logging enabled on Tue Jun 17
12:18:12 CDT 2008
TreatmentFramework: UNGUIDED treatment started on Tue Jun 17
12:18:12 CDT 2008
Screen0: We have not cycled through all scenarios. Continuing
with next scenario.
MapScrollPaneListener started: Tue Jun 17 12:18:15 CDT 2008
MapScrollPaneListener started: Tue Jun 17 12:18:15 CDT 2008
MapspotLabelListener started: Tue Jun 17 12:18:15 CDT 2008
Entered (345,307)     Background
Move (347,301)   Background
Press on (347,301)    Background
Drag (345,290)   Background
Release (345,290)     Background
Move (334,300)   Background
Press on (334,300)    Background
Release (334,300)     Background
Click on (334,300)    Background
Exited (414,284) Background
Entered (22,60)  activeScreenArea
Move (22,60)     activeScreenArea
Move (23,58)     activeScreenArea
Move (24,57)     activeScreenArea
Move (24,56)     activeScreenArea
Move (24,55)     activeScreenArea
Move (24,54)     activeScreenArea
Move (24,53)     activeScreenArea
Move (24,52)     activeScreenArea
Move (23,51)     activeScreenArea

      53727.0    loop1Stopwatch:       53727.0
      screen1Stopwatch:          53727.0
 Move (32,8)      Screen1NoButton scenario0AStopwatch:
      53767.0    loop1Stopwatch:       53767.0
      screen1Stopwatch:          53767.0
 Move (32,9)      Screen1NoButton scenario0AStopwatch:
      53817.0    loop1Stopwatch:       53817.0
      screen1Stopwatch:          53817.0
 Press on (32,9)  Screen1NoButton scenario0AStopwatch:
      54128.0    loop1Stopwatch:       54128.0
      screen1Stopwatch:          54128.0
 Screen1NoAnswerAction: correctly moving to Screen2.
      scenario0AStopwatch:       54128.0
      loop1Stopwatch:       54128.0
      screen1Stopwatch:          54128.0
 screen1Stopwatch stopped and removed on Tue Jun 17 12:19:50 CDT
 2008
      scenario0AStopwatch:       54128.0
      loop1Stopwatch:       54128.0
      screen1Stopwatch:          54128.0
 CollectionOfStopwatches: stopAndRemoveStopwatch: cannot stop the
 nonexistent stopwatch screen1aStopwatch
 loop1Stopwatch stopped and removed on Tue Jun 17 12:19:50 CDT
 2008
      scenario0AStopwatch:       54128.0
      loop1Stopwatch:       54128.0
 CollectionOfStopwatches: loop2Stopwatch started on Tue Jun 17
 12:19:50 CDT 2008
      scenario0AStopwatch:       54128.0
      loop2Stopwatch:       0.0
```

Figure 3. An example of the output file.

4.2 Patterns

To finish the task given by the software described in the previous section, several operations, such as: zoom in, zoom out (either click zoom or use the rolling bar), pan left, pan right, pan up, pan down, click and reset, are needed. Generally speaking, our study is based on the count of 1) these operations; and 2) combinations of these operations. A combination is a string linked by several operations. For example, how many pan left a user used to finish a task? How many pan right after a pan left operation a user used to finish a task? etc.

Our study focuses on some characteristic combinations which we call “count patterns”. A count pattern, denoted as n(list of one/or a series of user operation), is detected if the operations in the list appears n times in a user input to finish a scenario. During the study, we find that some specific count patterns are meaningful because they have the potential to save extra user operations after the pattern is detected. That means the operations after some specific patterns have a great chance to include unnecessary operations. According to the situation model, if we can find these “pattern” early enough and invoke the “service” before users do the extra work, it would be easier for users to finish their task.

We have found several meaningful patterns in our study. In this research, we will focus on four of these pattern types which we found to have significant potential for savings:

Pattern 1: n(A) A is on or more pan operations of the same type.
Pattern 2: n(A) n(B), B is the reversal of A.
Pattern 3: n(A B)
Pattern 4: n(A) (d) m(B), the n occurrence of A are strictly before B.

In these four patterns, “A” and “B” mean a specific operation or combination of operations which will be counted. The “n” and “m” stands for the times the operation/combination exists. “d” means the operations or segment that we do not care what it is but it does separate the counts. Note that if n(A) is not separate from n(B), it would be the same as n(A)n(B). An example of 2(AB) for Pattern 3 could be: ABAB. An example of 2(A)3(B) could be AABBB. An example of (A)(d)(B) could be an “A” followed by a operation/combination that we do not care what it is and then end with a “B”.

Pattern 1 is a pattern that only contains pans of the same type. In our research with this experiment, we find the most used operations are pans (pan left, pan right, pan up, pan down). There are 1374 pan operations in an overall 2426 operations of guided data and 1503 pan operations in an overall 2513 operations of unguided data. Since pans take more than half part of the total, we pay more attention on it.

Pattern 2 is a pattern with n occurrences of reversal operations. A couple of reversal operations mean that one operation is the reverse of another one. For example, “pan left” and “pan right” form a reversal; “zoom in” and “zoom out” form a reversal. We call pattern 2 “reversal pattern”. In a reversal pattern, we count the number of occurrences of “A” and the reversal of “A”. Since during the counts there are no differences between n(A)n(B) (B is the reversal of A) and n(B)n(A), so we considered n(B)n(A)=n(A)n(B).

Pattern 3 is a pattern with several repeated consecutive operations. We call it “consecutive pattern”. In consecutive pattern, A and B should be conscutive together and there are no other operations between them.

Pattern 4 is related to Pattern 2 in the sense that we are looking for counts of pattern A and B, but in Pattern 4 the counts of A, for example, n(A) has to appear before the don’t cares and before we start counting the pattern B.

Based on these four kinds of pattern, we did a lot of analysis based on the data comes from the user study described in Section 4.1. The result will be shown in the next section. We design several methods to analyze the data. Our goal is to find a way to make the application more user-friendly based on the patterns we have found. In particular, we are interested in identifying patterns that occur early in the work on an address that appear cause the user to get lost on the map. In the next section, we will also show the result of our analyzing and further explain what can be improved in location-based software by using the situation conception.

CHAPTER 5. RESULTS

5.1 Overview

In this section, results from the analysis of the experiment descripted in the last chapter will be shown. From these results, we can further understand the importance of the patterns for Situations. It is possible to potentially "save" some unnecessary operations by using these patterns. The count patterns described in Chapter 3 will be further discussed in this section by showing the specific examples and doing analysis. The general idea is that we are trying to find some "interesting" patterns which may lead to potential savings by counting the number of different sets of operations. The goal of this research is to use the user study to identify interesting patterns and check whether these can lead to some potential savings. If so, then it means we can use these patterns in a situation system as triggers to help the users. We will use another user study to validate our result in validation chapter.

5.2 Transform the Data

As we mentioned before, the operations performed by a participant in the user study were recorded by the software as shown in Figure 3. Since we need to do analysis on the map operations, we need to convert the log file into the map operations for each address being verified (scenario). Then we can automate the calculation to find the sequences and counts that may lead us to our expected conclusion. We will do following steps:

Step 1: output the important user operations in the logs and generate a file of user's input stream. The log file shown in Figure 3 has a lot of operations that are of less value in our research. For example, the user moving his mouse and clicking some meaningless region, this is an operation with less value. In this research, we only focus on the map operations. The map operations are listed in Table 1. In this step, we output the tokens for operations in Table 1 to a file. In this file, each line is a set of operations recorded from one single scenario. At the end of a scenario, the next operation will be output to a new line.

Step 2: Based on the file generated in Step 1, replace each of the tokens with one-single-length character from Table1. The map of operations-symbols are shown in Table 1. For example, a "pan left" operation will be replaced by a character "<", a "pan right" operation will be replaced by a ">" as its symbol. We call each line a "user input stream". Figure 4 provides the results for 10 scenarios by one user. By doing this, the user input stream can be seen to represent a set of map operations for one scenario.

Step 3: we use a java program to count the operations in user input streams and try to figure out if there are some meaningful patterns of the form introduced in Chapter 3. As mentioned before, we have 17 people treated as guided and 18 people treated as unguided. So the analysis will also be divided into two group, one is guided and another is unguided. We have 3 programs that were implemented in different way to count the result. For Pattern 1 and Pattern 2, since they are simple counting work, we just use the program to count the times the operation exists. For Pattern 3, the count program will have a pattern as input parameter. The program will first read the file line by line, and identify how many patterns included in each line. The program also has an input parameter “n” which is an integer. The program will count how many operations remain after the “n” required patterns are detected in this line. The program for Pattern 4 is similar to the program of Pattern 3. The only two differences are: 1) the program of Pattern 4 has two input parameters, A and B, both are combination of operations. 2) the program of Pattern 4 enforces the order very strictly. It will consider a valid count if and only if A happens before B. This is the general idea of the count program. The more detailed information and result will be shown in Section 5.3, User Study Result.

Table 1. The map operations that we focus on and their symbols.

Pan left	<
Pan right	>
Pan up	A
Pan down	V
Click	X
Zoom in	+
Zoom out	-
Reset	R
One level zoom in using the scroll bar	B
Two level zoom in using the scroll bar	C
One level zoom out using the scroll bar	B
Two level zoom out using the scroll bar	C

```
+x+>+x+++>>>><<x+x-
++A<>>>>+<>++
++ V<>>>
++>AAV
+x+><
+x+VAAAAVVVVV>>++x+<<VVV
<<+VV<>>>>+<>++--+x+x
+x+V>A
++A
+x+-+xVA
```

Figure 4. An example of user input stream. Each line is a scenario finished by a participant.

5.3 Result

In this section, we will show the results of our work using the first user study. The data is divided to two pieces based on whether or not the participants were guided or not, since the interface of guided and unguided treatment are different.

Since we want to know if the situation system can really help in improving the location-based application, we should check if the location-based application that could be improved. The idea is we want to find some patterns which may indicate that some extra user operations are being performed. If we can find patterns after which there exist unnecessary user operations, it means the situation system can intervene and help the user.

In this research, our goal is to identify meaningful patterns and try to find out if there are some potential savings after a pattern is encountered. For each scenario, there is an optimal input stream to finish the task. In the user study, each scenario was finished by several different users. From these users' input stream, we pick the shortest one as the optimal. An optimal input stream can stand for an actual need for a user to finish a task of a scenario. In a user input stream, if there remains a string which length is greater than the optimal input after a pattern is detected, we consider this input stream contains some potential saving. Generally speaking, in this research we focus on the remaining part in each input stream after a pattern is encountered. If the remaining part is very long and even longer than the actual need to finish the task (the optimal), it means in this input stream the user did a lot of unnecessary operations. In our research, we call this "remaining part" as "potential saving".

By analyzing the input stream we get from the user study described in Chapter 3, we can successfully identify some patterns that lead to potential savings. We divide these patterns into four categories as defined in Chapter 3.

Pattern 1: n(A)

In this pattern, A is a pan operation. During our study, we found there were 1503 pan operations in unguided data set whose total operations are 2513. That means there are 59.8% operations are pans. Similarly, there are 1374 pan operations, account for about 58.8%, in the guided data set with 2336 operations as total. Pan operations take more than half of the total. So we try to do some counts on pans and identify if there are some patterns with only pans that have the potential to save extraneous operations.

Table 2. The number of scenarios (out of 170) in the guided data set that contain pan patterns of n=2 and n=3, respectively.

Pan Count Patterns	Count n=2	Percentage n=2	Count n=3	Percentage n=3
n(>)	52	30.6%	30	17.6%
n(<)	52	30.6%	31	18.2%
n(A)	72	42.4%	45	26.5%
n(V)	65	38.2%	39	28.8%

Table 3. The number of 17 participants in the guided data set that contain pan patterns of n = 2 and 3, respectively.

Pan Count Patterns	Count n = 2	Percentage n = 3	Count n=3	Percentage n = 3
n(>)	17	100%	13	76.5%
n(<)	15	88.2%	13	76.5%
n(A)	17	100%	14	82.4%
n(V)	17	100%	15	88.2%

Table 2 shows the number of scenarios (out of the 170) that contains at least two or three (n=2 and n=3) pan operations. The column “Count” shows the number of scenarios have pan patterns detected, and the column “Percentage” shows the percentage it takes in overall. Table 3 shows the number of participants whose input stream contains these patterns. Similarly, the “Count column” shows the number while the “Average” column shows the percentage. Both Table 2 and Table 3 are counted with the guided data set.

Table 4. The number of scenarios (out of 180) in the unguided data set that contain pan patterns of n=2 and n=3, respectively.

Pan Count Patterns	Count n = 2	Percentage n = 3	Count n=3	Percentage n = 3
n(>)	66	36.7%	45	25.0%
n(<)	70	38.9%	48	26.7%
n(A)	82	45.6%	56	31.1%
n(V)	68	37.8%	51	28.3%

Table 5. The number of 18 participants in the unguided data set that contain pan patterns of n = 2 and 3, respectively.

Pan Count Patterns	Count n = 2	Percentage n = 2	Count n=3	Percentage n = 3
n(>)	17	100%	13	76.5%
n(<)	15	88.2%	13	76.5%
n(A)	17	100%	14	82.4%
n(V)	17	100%	15	88.2%

Table 4 and Table 5 are the same results for unguided data set. Table 4 shows how much percentage of scenarios that contains listed pan patterns. Table 5 shows how much percentage of users who has used this pattern in their input stream.

Table 6. The number of operations after a pan pattern is encountered and the number of scenarios that contains one or more pan patterns in guided data set.

N	Operations After Pattern	Scenario impacted out of 170
2	1130	104
3	886	75

Table 7. The number of operations after a pan pattern is encountered and the number of scenarios that contains one or more pan patterns in unguided data set.

N	Operations After Pattern	Scenario impacted out of 180
2	1536	111
3	1239	82

Table 6 and Table 7 show the total number of map operations remaining in scenarios after a pan pattern has been detected and the number of scenarios that contains one or more of the pan patterns in the guided and unguided data set, respectively. Note that a pan pattern is the pattern we talked above who has a form like n(A) where “A” is a pan operation.

Pattern 2: n(A)n(B)

In Pattern 2, B is a reversal of A. The reversal operation is an obvious type of pattern that we should pay attention to. Since two reverse operations will offset each other, they don’t get the user closer to his goal when using a location-based application. More important, we find instances where such reversals, especially, with zooms that can get the user lost. An example is a zoom reversal without centering. So we count the reversal pairs to find if there are some proper patterns that tend to generate extraneous operations.

Table 8. The number of scenarios in the guided data set that contains reversal patterns.

Reversals Patterns	Count n = 2	Percentage n = 2	Count n=3	Percentage n = 3
n(z+)n(z-)	19	11.2%	10	5.9%
n(>)n(<)	36	21.2%	20	11.8%
n(A)n(V)	55	32.4%	36	21.2%

Table 9. The number of 17 participants in the guided data set whose input stream contains reversal patterns.

Reversals Patterns	Count n = 2	Percentage n = 2	Count n=3	Percentage n = 3
n(z+)n(z-)	13	76.5%	9	52.9%
n(>)n(<)	15	88.2%	9	52.9%
n(A)n(V)	17	100%	13	76.5%

Table 8 shows the number of guided scenarios (out of 170) that contains at least two or three (n=2 and n=3) reversal operations. The column “Count” shows the number of scenarios has reversal patterns detected, and the column “Percentage” shows the percentage it takes in overall. Table 9 shows the number of participants whose input stream contains these patterns. Similarly, the “Count column” shows the number while the “Percentage” column shows the percentage. Both Table 8 and Table 9 are generated with the guided data set.

Table 10. The number of scenarios (out of 180) in the unguided data set that contains reversal patterns of n=2 and n=3, respectively.

Pan Count Patterns	Count n = 2	Percentage n = 2	Count n=3	Percentage n = 3
n(z+)n(z-)	33	18.3%	19	10.6%
n(>)n(<)	53	29.4%	33	18.3%
n(A)n(V)	58	32.2%	44	24.4%

Table 11. The number of 18 participants in the unguided data set that contains reversal patterns of n = 2 and 3, respectively.

Pan Count Patterns	Count n = 2	Percentage n = 2	Count n=3	Percentage n = 3
n(z+)n(z-)	13	72.2%	10	55.6%
n(>)n(<)	16	88.9%	12	66.7%
n(A)n(V)	16	88.9%	14	77.8%

Table 10 and Table 11 provide the same results for unguided data set. Table 10 shows the percentage of scenarios that contains listed pan patterns. Table 11 shows the percentage of users who has used this pattern in their input stream.

Table 12. The number of operations after a pan pattern is encountered and the number of scenarios that contains one or more pan patterns. This is in guided data set.

N	Operations After Pattern	Scenario impacted out of 170
2	1130	104
3	886	75

Table 13. The number of operations after a pan pattern is encountered and the number of scenarios that contains one or more pan patterns. This is in unguided data set.

N	Operations After Pattern	Scenario impacted out of 180
2	1167	84
3	863	59

Table 12 and Table 13 shows the number of map operations after a reversal pattern is detected and the number of scenarios that contains one or more this kind of pattern for n=2 or 3. Note that a reversal pattern is the pattern we talked above who has a form like n(A)(d)n(B) where "A" is a reversal of "B".

Pattern 3: n(A B) or n(ABC)

In this pattern, "A" and "B" and "C" are consecutive map operations. To find this kind of pattern, we use a data mining approach to pick some strings as candidates of "AB" or "ABC". We designed a function Kgram(length,times) to pick out the string combinations we want. This function has two input parameter: one is the length of the target strings and another one is the times the target string exists. The program will read the data file first and call this function to require some outputs. The Kgram function will generate a list of string combinations that have a length equals to the "length" in the input parameter and whose existing frequncy is greater than the "times" parameter. The program will output a list of these strings and we call those strings candidates. Table 14 is a output list generated by using Kgram(3,50). So it lists

all the string combinations in the result file whose length is equal to 3 and who exists more than 50 times. The first column is the candidate strings. The second column is the times it existed overall. The third column means the percentage of lines (scenarios) that contained at least one target string combinations. It is calculated by the lines contains in the target strings divided by the total lines. Table 15 is the same output of Kgram(2,140). During our test, we find that the strings with length 2 or 3 has a good existing rate so we decide to choose 2 and 3 as the first parameter in Kgram function. To choose the second parameter in Kgram function, we test a lot and find when the value is 50 and 140 respectively, the output would not be too big or too small. The strings that have the highest existing rate will be listed in the output files.

Table 14. An example of Kgram(3, 50).

Key Strings	Overall	Percentage (Line)
+x+	236	50.57%
x+x	106	24.29%
+xV	50	11.14%
VAA	69	16.57%
AVV	59	14.0%
VVV	110	15.71%
VVA	65	16.0%
AAA	81	13.71%
AAV	65	15.43%

Table 15. The output of Kgram(2,130).

Key Strings	Overall	Percentage (Line)
+x	446	62.29%
x+	287	52.29%
+>	98	23.71%
+A	89	22.0%
<>	110	22.86%
<+	101	21.71%
+V	100	21.71%
VV	233	29.14%
+<	107	23.43%
VA	156	26.86%
AA	210	30.29%
AV	119	25.71%

From Table 14**,** we can see the candidates. But before we do the count, we have to remove some of them since they may not be the strings we really need. For each scenario, there should be an optimal solution to satisfy the task. We consider the shortest input stream among all participants that can finish the task in the scenario as the optimal one. In Table 14**,** it is shown that string “+x+” and “x+x” has the highest existing rate. However, we find “+x+” and “x+x” is contained in almost every optimal input and they are actually necessary to finish the most of the scenario tasks. So we should remove them from the candidates. In Table 15**,** we choose to remove “+x” and “x+” for the same reason. Since we have already counted strings like”<>”, “VA”, and “AV” in the work of Pattern 2, so they should also be removed. Finally, we pick 4 strings respectively from Table 14 and Table 15 that have the highest existing rate as our candidates for Pattern 3. The final result is shown in Table 16.

Table 16. The strings we finally choose to use as Pattern 3.

Key Strings	Overall	Percentage (Line)
VV	233	29.14%
AA	210	30.29%
>>	134	19.43%
<<	140	19.71%
Key Strings	Overall	Percentage (Line)
VAA	69	16.57%
VVV	110	15.71%
VVA	65	16.0%
AAV	65	15.43%

Now, let's see the count result of Pattern 3. Table 17 is the count result of Pattern 3 which shows the number of scenarios in the guided data set that contains the consecutive patterns. Table 18 shows the same result for unguided data set. Table 19 is the number of 17 participants in the guided data set that contains consecutive patterns of n = 2 and 3. Table 20 is the same result for unguided data set. Table 21 and Table 22 are the numbers of operations after consecutive patterns has been encountered when n= 2 and 3 for guided and unguided data set respectively.

Table 17. Number of scenarios in the guided data set that contains the consecutive patterns.

Consecutive Patterns	Count n = 2	Percentage n = 2	Count n=3	Percentage n = 3
n(>>)	12	7.1%	4	2.6%
n(<<)	15	8.8%	7	4.1%
n(AA)	24	14.1%	11	6.5%
n(VV)	21	12.4%	11	6.5%

Table 18. Number of scenarios in the unguided data set that contains the consecutive patterns.

Consecutive Patterns	Count n = 2	Percentage n = 2	Count n=3	Percentage n = 3
n(>>)	14	8.2%	9	5.3%
n(<<)	15	8.8%	8	4.7%
n(AA)	21	12.4%	9	5.3%
n(VV)	24	14.1%	13	7.7%

Table 19. The number of 17 participants in the guided data set that contains consecutive patterns of n = 2 and 3, respectively.

Reversals Patterns	Count n = 2	Percentage n = 2	Count n=3	Percentage n = 3
n(>>)	7	41.5%	2	11.8%
n(<<)	7	41.5%	5	29.4%
n(AA)	10	58.8%	7	41.5%
n(VV)	10	58.8%	8	47.1%

Table 20. The number of 18 participants in the unguided data set that contains consecutive patterns of n = 2 and 3, respectively.

Reversals Patterns	Count n = 2	Percentage n = 2	Count n=3	Percentage n = 3
n(>>)	10	55.6%	7	38.9%
n(<<)	9	50.0%	6	33.3%
n(AA)	11	61.1%	6	33.3%
n(VV)	14	77.8%	8	44.4%

Table 21. The number of operations after a consecutive pattern is encountered and the number of scenarios that contains one or more consecutive patterns. This is in guided data set.

N	Operations After Pattern	Scenario impacted out of 170
2	648	38
3	395	21

Table 22. The number of operations after a consecutive pattern is encountered and the number of scenarios that contains one or more consecutive patterns. This is in unguided data set.

N	Operations After Pattern	Scenario impacted out of 180
2	735	46
3	482	24

Table 23. Number of scenarios in the guided data set that contains the consecutive patterns.

Consecutive Patterns	Count n = 2	Percentage n = 2	Count n=3	Percentage n = 3
n(VAA)	1	0.59%	0	0%
n(VVV)	10	5.88%	6	3.53%
n(VVA)	2	1.18%	0	0%
n(AAV)	6	3.53%	1	0.59%

Table 24. Number of scenarios in the unguided data set that contains the consecutive patterns.

Consecutive Patterns	Count n = 2	Percentage n = 2	Count n=3	Percentage n = 3
n(VAA)	4	2.22%	2	1.11%
n(VVV)	12	6.67%	6	3.33%
n(VVA)	5	2.78%	1	0.56%
n(AAV)	2	1.11%	0	0%

Pattern 4: n(A) (d) n(B)

In Pattern 4, A and B could be either single user operation or a combination of some operations. Similar to Pattern 3, we have to find some candidates of A and B. In this part, we will use the output result that we generate for Pattern 3. Since we don't want to do some meaningful counts we decided to use the candidates from Pattern 3 as the candidates of A and B in Pattern 4. So we will try to find the combinations which consist with the candidates in Pattern 3 as the candidates of Pattern 4. Note that a very important difference between Pattern 3 and Pattern 4 is that Pattern 3 only counts but Pattern 4 will take some care on the order of the strings. That means, a pattern with n(A) (d) m(B) means, the m occurrences of B should strictly happens after the n occurrences of A. This is very important. Since when the length of A and B becomes to 2 and 3, this kind of patterns happens very rarely, so we can only find four instance of this pattern which can be detected in both guided and unguided dataset. Table 25 and Table 26 show the count result of the order patterns detected in the guided data set and unguided data set when n=2, m=2 and n=3, m=3 respectively.

Table 25. Number of scenarios in the guided data set that contains the order patterns.

Order Patterns	Count n=2, m=2	Percentage n=2, m=2	Count n=3, m=3	Percentage n=3, m=3
n(AA)(d)n(VVV)	4	2.35%	4	2.35%
n(VVV)(d)n(AA)	8	4.71%	1	0.59%
n(AA)(d)n(VV)	6	3.53%	4	2.35%
n(VV)(d)n(AA)	9	5.23%	5	2.94%

Table 26. Number of scenarios in the unguided data set that contains the order patterns.

Order Patterns	Count n=2, m=2	Percentage n=2, m=2	Count n=3, m=3	Percentage n=3, m=3
n(AA)(d)n(VVV)	5	2.78%	4	2.22%
n(VVV)(d)n(AA)	5	2.78%	2	1.11%
n(AA)(d)n(VV)	9	5.0%	5	2.78%
n(VV)(d)n(AA)	7	3.89%	4	2.22%

CHAPTER 6. DISCUSSION

In this chapter, we will have a discussion about the results we gave in Chapter 5. The goal of this study was to find patterns and show that by identifying these patterns have a possibility to intervene and save some unnecessary operations. From our research results, we find that some of the map operations used has the potential to cause a user to get lost on the map. They are not necessary in the optimal inputs but they tend to lead to a lot of user operations after they occur in a scenario.

Our expectation is that the situation service triggered by the patterns would give the user an opportunity to complete the task more efficiently. That is, we believe these patterns can be detected when the user is doing some unnecessary work. If operations exist in every optimal input stream, the pattern cannot be a good pattern, because it is necessary to complete the task. If there are very few operations remain after a pattern is encountered, it unlikely to be a good pattern. In addition, if a string exists very rarely in the user input stream, it is unlikely to be a good pattern. The good patterns we need to find should exist often in the user inputs, but they are not the necessary in the optimal solution. They should typically have quite lot operations after they are detected to finish the address verification.

Since this research only focuses on looking for potential situation patterns, we are mainly concerned finding potential savings. We cannot say if these operations are necessary or not to finish the task. What we only know is if the remaining part after a pattern in encountered is much longer than the optimal inputs, there is a possibility to improve it. For example, by using Ming's [8] situation environment model, the software can provide some proper service based on our patterns to let user know he may have some trouble in using the software. Our current work is to identify the meaningful patterns that can be used in a situation model to make the location-based software easier to use.

From the count results described in the last chapter, we can see the patterns we have found can match the all three conditions for good patterns we talked above. For Pattern 1 and Pattern 2, all of them are not necessarily part in the optimal inputs. From Tables 2, 4, 8, and 10, we know that the pan patterns and reversal patterns we've found exist very frequently in the user's input stream. They exist in 10% to 30% of the total scenarios. Note this rate is just right because too high of rate may reflect the operations are necessary to finish the task or are very common operations to use to finish the task. Too low rate may reflect the fact that these operations exist very rarely so they has less value. From Tables 3, 5, 9, and 11, we can see that almost all the participants had been used these patterns in their input stream (At least 55.6%, about 70% as average). This is a very high rate. In Tables 6, 7, 12, and 13, we can see that there are a lot of operations after a pattern is encountered. About 10.8 map operations per impacted scenario happen after a pattern is detected in guided

treatment. In the unguided treatment it happens even more. About 13.2 map operations per scenario impacted. Since the optimal input can be finished in about 4 to 8 operations, it means there are quite a lot of additional operations appears after the pattern. If we can find a proper service there will be a possibility to greatly reduce the user's frustration. So we can say the reversal patterns and the pan patterns we have found should be the good patterns at least in this location based application we used in the user study.

For Patterns 3 and 4, there are some differences. The first thing is that since Patterns 3 and 4 have a relatively bigger size (they contain 2 or 3 consecutive operations as one single segment in our counting, such as in Pattern 3, a pattern is counted as n(AAV), which is longer than a pattern in Pattern 1, such as n(>)), so it obviously should have a relatively lower rate of existence. In Table 17 and Table 18**,** we can see that for a consecutive pattern whose segment length is 2, it still has a high existing rate. Each of them has an existing rate around 10% at n=2 and about 5% when n=3. But when the length of a segment comes to 3, in Table 22 and Table 23 we can see that some of the rate drops to 0. Consider the results about Pattern 1 and Pattern 2, we can conclude that the length of the pattern may be determined by the complexity of a task. Our count result also matches this point. Besides, in Table 21 and Table 22, we can see the consecutive pattern we have found in Pattern 3 leads to a lot of operations when n= 2 and 3. In the guided data set, there are about 648 operations remains after a consecutive pattern is encountered when n=2, and 395 operations when n=3. For the unguided data set, there are about 735 operations remains after a consecutive pattern is encountered when n=2, and 482 when n=3. The number of potential saving is still very high. Since the potential saving is much more than the optimal inputs, so we can conclude that these patterns we found in Pattern 3, as consecutive patterns, are good patterns to be a trigger to call a intervene. In Table 23 and Table 24**,** we can see that some of the pattern whose length is 3 happen very rarely when n=2 or 3. Though they are detected many times in the data set, but they are rarely detected more than one times in a single scenario. For this kind of pattern, they are not as useful. For example, n(VVA) and n(VAA) are not that meaningful in our result. On the other hand, the pattern n(VVV) often happens repeatedly in a single scenario, 10 are detected when n=2 in guided data set and 6 are detected when n=3. The numbers are even high in unguided data set. So for this pattern, it appears to be a good pattern in a situation system.

For Pattern 4, we can see that though there are still some potential savings when an instance of Pattern 4 is detected under n = 2 and 3. But we should note that it is much less than the potential savings in Pattern 1 and 2, since there are less scenarios that contain Pattern 4 when n =3. Even when n=2, the number is still very small. From this, we know that this kind of pattern is quite specific or it is not that general. When the application changes, we hardly able to find this pattern. But it is still useful in this user study. Further evaluation should be done to check if it still works on other types of location-based software.

Another interesting thing has been found during our research in the counting step, is that we find that for a person, who has input a pattern we identified in Chapter 4, and has a great chance to input the same pattern in another scenario. And for these people, they usually do many more operations than the task requires. For example, a person who types pattern 2(AAV) in one of his task, usually types 2(AAV) in the other scenarios as well. Furthermore, all of his inputs in which we can detect a pattern, are much longer than the optimal one. So we think that if we find a person whose input has a lot of patterns detected frequently, he may have some trouble using the software and may be helped by intervention from the situation model. From this, we can also say, our pattern is quite useful as a trigger for calling a service in a situation system.

The following list is the pattern we have successfully identified in our research and we consider them as good patterns (n is equal to 2 or 3):

Count Pan Patterns: n(<), n(>), n(A), n(V).
Count Reversal Patterns: n(<)n(>), n(A)n(V), n(Z+)n(Z-).
Count Consecutive Patterns: n(>>), n(<<),n(AA),n(VV),n(VVV).
Order Patterns: n(AA)(d)m(VVV), n(AA)(d)m(VV), n(VVV)(d)m(AA), n(AA)(d)m(VVV).

From our research, we find that Pattern 1 and Pattern 2 have more potential savings than Pattern 3 and Pattern 4. The reason is that Pattern 3 and Pattern 4 happens very rarely when n become bigger and bigger. When n=2, there are still some potential savings can be found in the inputs. But when they grow to 3, 4 or even 5, the potential savings will quickly drop to a very low value. Since the goal of this research is to find some valuable patterns in location based application, we consider Pattern 1 and Pattern 2 are more valuable than Pattern 3 and Pattern 4. Pattern 3 and Pattern 4 are more specific. But Pattern 1 and 2 are more general and more flexible. Based on this point, to verify if pattern 1 and 2 are still valid in a very different implementation of the address verification software, we will evaluate the patterns in another implementation of the address verification task.

CHAPTER 7. VALIDATION

In this chapter, we will describe another user study based on address verification, let's call it user study 2. We use user study 2 to validate the patterns we found in user study 1. The location-based software used by user study 2 is totally different than the one user study 1 used.

Thirty-one participants performed the address verification task for 6 addresses in the second study. They are divided into two groups: the real group and the VR group. The real group will navigate in a 3x4 block neighborhood of Ames, Iowa, and had to navigate as well as perform the address verification on the software. The area is shown in Figure 6. The VR group do the same thing but they will be in Iowa University's C6 (a fully immersive virtual reality environment) and use a virtual model to simulate their navigation of the neighborhood. The detail of this user study can be found in Batinov, et al [1].

A key difference between the two studies is that in user study 2, the participants could choose the scenario they wanted to work on in any order. They could revisit any scenario as well. Another difference between user study 1 and 2 is that after a user finish a scenario in user study 2 the software would not reset to map. Figure 5 shows the interface of user study 2.

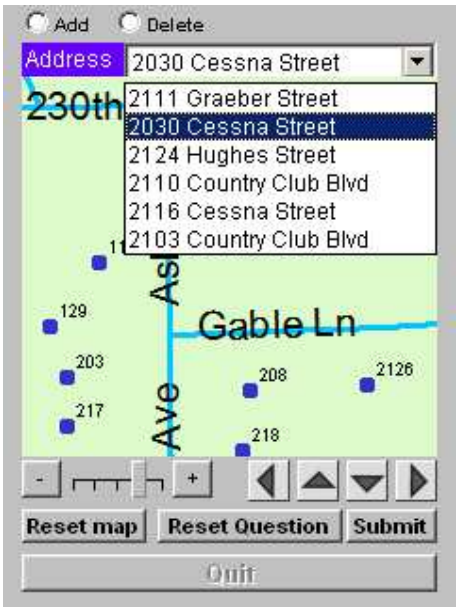


Figure 5. Interface of user study 2.

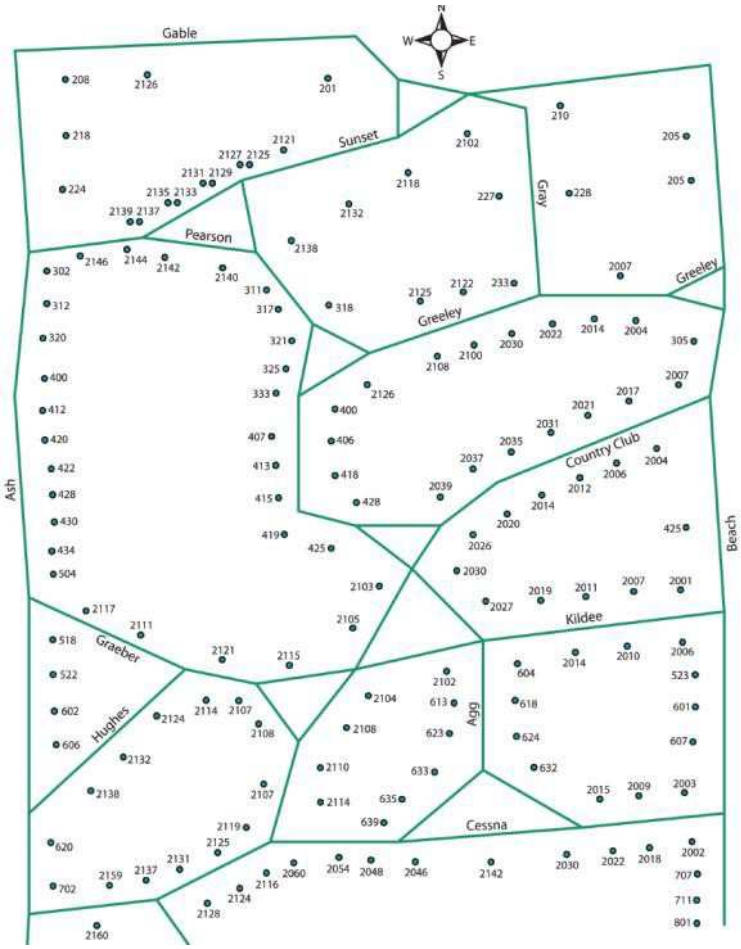


Figure 6. Area in user study 2.

Figure 7 shows the map operations for one of the 31 participants in user study 2. There are some new symbols added in this user study: JZ, KZ, LZ, MZ, NZ and PZ indicate the selection of one of the six scenarios in this user study. Note that the user data shown in Figure 7 illustrate that the user worked on some scenarios more than once (e.g. NZ & JZ).

JZ--A<V><AVV><>Ax
PZ<V<AAVV>cBxBV<V<>AA><><BV><<>V<>
MZ
PZ
MZV<>V<>A<<VA>b>--++>>><<-+V**
NZ<>>><<<<>>><A<<V>>>>*<><A<<V+-
KZ>>><<><>><<<A
NZ<>>-->-++-++--++<V---
LZxA<<>>V
NZ<<A
LZ<>-->>AA<<VVxAVA
JZ>V>>V

Figure 7. Map operations showing one line for each open scenario for Study 2 for one participant.

Interestingly, the result shows the pan patterns and the reversal patterns will still lead to a significant potential savings. The results are given in Tables 27,28, and 29.

Table 27. Number of map operations/impacted lines after encountering a reversal or a pan count pattern for the VR treatment when n=2 and 3.

N	pans	reversals	Both
2	997/88	834/67	1039/91
3	730/58	559/51	782/62

Table 28. Number of map operations/impacted lines after encountering a reversal or a pan count pattern for the real treatment when n=2 and 3.

N	pans	reversals	Both
2	610/65	515/57	664/73
3	405/45	298/36	460/51

Table 29. Number of map operations/impacted lines after encountering a reversal or a pan count pattern for the complete dataset when n=2 and 3.

N	pans	reversals	Both
2	1607/153	1350/124	1703/164
3	1135/103	857/87	1242/113

From Table 27, 28 and 29, we can see that in user study 2, there are 1607 and 1135 potential saved operations as total when n=2 and n=3 respectively. This means our pattern 1 and pattern 2 still works in the software used by user study 2. That is a very good result.

CHAPTER 8. CONCLUSION AND FUTURE WORK

In this research, we were able to identify some interesting patterns. These patterns have a proper existing rate and will result to a significant potential savings. From our result, we can know that Pattern 1 and 2 can save the most potential operations because they are more general and flexible. In the user study 2, we also verified that our count patterns 1 and 2 works as well. By here, it means we have already found some useful patterns. Since our final goal is to use these patterns in situation system, we need to do more experiments to check if these patterns have general meaning. That is, they will be valuable in all kinds of location based software. Also, if the sample of the participant for our user study could be enlarged, we may find more interesting patterns. From this research, we believe our pattern can be helpful in a situation system and further make the location-based software easier to use.

REFERENCES

[1] Anand S. Rao and Michael P. Georgeff. Modeling rational agents within a bdi-architecture. In James F. Allen, Richard Fikes, and Erik Sandewall, editors, *KR*, pages 473–484. Morgan Kaufmann, 1991.

[2] Bratman, Michael. *Intentions, Plans, and Practical Reasoning*. Harvard University Press, 1991.

[3] Chang, Carl K., Hsinyi Jiang, Hua Ming, and Katsunori Oyama. Situ: A situation-theoretic approach to context-aware service evolution. *IEEE T. Services Computing*, 2(3):261–275, 2009.

[4] Findlater, Leah and Joanna McGrenere. A comparison of static, adaptive, and adaptable menus. In *Proceedings of the SIGCHI conference on Human factors in computing systems*, CHI '04, pages 89–96, New York, NY, USA, 2004. ACM.

[5] Gajos, Krzysztof Z., Mary Czerwinski, Desney S. Tan, and Daniel S. Weld. Exploring the design space for adaptive graphical user interfaces. In *Proceedings of the working conference on Advanced visual interfaces*, AVI '06, pages 201–208, New York, NY, USA, 2006. ACM.

[6] Liu, Jiming, Chi Kuen Wong, and Ka Keung Hui. An adaptive user interface based on personalized learning. *Intelligent Systems, IEEE*, 18(2):52–57, mar-apr 2003.

[7] Ming, Hua. 2012. *Situf* : A Domain Specific Language and A First Step Towards the Realization of *Situ* Framework. PhD Dissertation in Computer Science, Iowa State University.

[8] Ming, Hua, C.K. Chang, K. Oyama, and Hen i Yang. Reasoning about human intention change for individualized runtime software service evolution. In *Computer Software and Applications Conference (COMPSAC), 2010 IEEE 34th Annual*, pages 289 – 296, july 2010.

[9] Ming, Hua, K. Oyama, and C.K. Chang. Human-intention driven self adaptive software evolvability in distributed service environments. In *Future Trends of Distributed Computing* 95 *Systems, 2008. FTDCS '08. 12th IEEE International Workshop on*, pages 51 –57, oct. 2008.

[10] Ramachandran, Krish. Adaptive user interfaces for health care applications. Available at
http://ibm.com/developerWorks, 2009.

[11]Rusch, M. L., S. M. Nusser, L. L. Miller, G. I. Batinov, and K. C. Whitney. (2012). Spatial Ability and Map-Based Software Applications. *The Fifth International Conference on Advances in Computer-Human Interactions.* Valencia, Spain. January 30-February 4, 2012, pp. 35-40.

[12] Shankar, Anil, Sushil J. Louis, Sergiu Dascalu, Linda J. Hayes, and Ramona Houmanfar. User-context for adaptive user interfaces. In *Proceedings of the 12th international conference on Intelligent user interfaces*, IUI '07, pages 321–324, New York, NY, USA, 2007. ACM.

[13] Sheer, Richard. The mental state theory of intentions. *Philosophy*, 79:121–131, 2004.

[14] Son, Young-Jun. 2014. An Extended BDI-Based Model for Human Decision-Making and Social Behavior: In: Martine Ceberio and Vladik Kreinovich (Editors). *Various Applications in Constraint Programming and Decision Making.* pp. 171-174.

[15] Tsandilas, T. and M. C. Schraefel. Usable adaptive hypermedia systems. *New Review of Hypermedia and Multimedia*, 10(1):5–29, 2004.

[16] Viano, Gianni, Andrea Parodi, James Alty, Chris Khalil, Inaki Angulo, Daniele Biglino, Michel Crampes, Christophe Vaudry, Veronique Daurensan, and Philippe Lachaud. Adaptive user interface for process control based on multi-agent approach. In *Proceedings of the working conference on Advanced visual interfaces*, AVI '00, pages 201–204, New York, NY, USA, 2000. ACM.

[17] Benyon, D., A. Crerar and S. Wilkinson. 2001. Individual differences and inclusive design. In *User interfaces for all – Concepts, methods and tools*, C. Stephandis (Ed.). Mahwah, NJ: Lawrence Erlbaum, pp. 21-46.

[18] Langley, P. 1997. User Modeling in Adaptive Interfaces. *Seventh International Conference on User Modeling. Banff*, Alberta. 357-71.

[19] Billsus, D., C. A. Brunk, C. Evans, B. Gladish, and M. Pazzani. 2002. Adaptive interfaces for ubiquitous web access. *Communications of the ACM.* 45(5). 34 - 38.

[20] Tsandilas, T. and M. C. Shraefel. 2004. Usable adaptive hypermedia systems. *New Review of HyperMedia and Multimedia.* 10(1) 5-29.

[21] Nguyen, N. T. and J. Sobecki. 2003. Using consensus methods to construct adaptive

interfaces in multimodal web-based systems. *Universal Access in the Information Society.* 2(4). 342-358.

[22] Mitchell, J. and B. Shneiderman 1989. Dynamic versus static menus: an exploratory comparision. *ACM SigCHI Bulletin.* 20(4). 33-37.

[23] Benyon, D. 1993. Accommodating individual differences through an adaptive user interface. In M. Schneider-Hufschmidt, T. Kuhme, and U. Malinowski (eds.). *Adaptive User Interfaces – Results and Prospects.* Amsterdam, North-Holland: Elsevier Science Publications.

[24] Viano G., J. Alty, I. Angulo, D. Biglino, m. Crampes, V. Daurensan, A. Parodi, C. Khalil, C. Vaudry, and P. Lachaud. 2004. Adaptive user interface for process control based on multi-agent approach. *AVI 2004.* Palermo, Italy. 201-204.