Roaming Through the Castle Tunnels: An Empirical Analysis of Inter-app Navigation of Android Apps

YUN MA, ZINIU HU, DIANDIAN GU, and LI ZHOU, Peking University, China
QIAOZHU MEI, University of Michigan, USA
GANG HUANG and XUANZHE LIU, Peking University, China

Smartphone applications (a.k.a., apps) have become indispensable in our everyday life and work. In practice, accomplishing a task on smartphones may require the user to navigate among various apps. Unlike Web pages that are inherently interconnected through hyperlinks, apps are usually isolated building blocks, and the lack of direct links between apps has compromised the efficiency of task completion and user experience. In this article, we present the first in-depth empirical study of page-level access behaviors of smartphone users based on a comprehensive dataset collected through an extensive user study. We propose a model to distinguish *informational pages* and *transitional pages*, based on which we can extract page-level inter-app navigation. Surprisingly, the transitional pages account for quite substantial time cost and manual actions when navigating from the current informational page to the desirable informational page. We reveal that developing "*tunnels*" between "isolated" apps under specific usage scenarios has a huge potential to reduce the cost of navigation. Our analysis provides some practical implications on how to improve app-navigation experience from both the operating system's perspective and the developer's perspective.

CCS Concepts: • **Human-centered computing → Empirical studies in ubiquitous and mobile computing**;

Additional Key Words and Phrases: Mobile apps, inter-app navigation, user experience, empirical study

ACM Reference format:
Yun Ma, Ziniu Hu, Diandian Gu, Li Zhou, Qiaozhu Mei, Gang Huang, and Xuanzhe Liu. 2020. Roaming Through the Castle Tunnels: An Empirical Analysis of Inter-app Navigation of Android Apps. *ACM Trans. Web* 14, 3, Article 14 (June 2020), 24 pages.
https://doi.org/10.1145/3395050

This work was supported by the National Key R&D Program of China under Grant No. 2018YFB1004800, the National Natural Science Foundation of China under Grant No. 61725201, the Beijing Outstanding Young Scientist Program under Grant No. BJJWZYJH01201910001004, and the Key Laboratory of Intelligent Application Technology for Civil Aviation Passenger Services, CAAC. Qiaozhu Mei's work was supported by the National Science Foundation under Grant No. 1633370.
Authors' addresses: Y. Ma, Z. Hu, D. Gu, L. Zhou, G. Huang, and X. Liu, Peking University, 5 Yiheyuan Road, 100871, Beijing, China; emails: {mayun, bull}@pku.edu.cn, gudiandian1998@pku.edu.cn, zhouliric@gmail.com, {hg, xzl}@pku.edu.cn; Q. Mei, University of Michigan, 500 S. State Street, Ann Arbor, MI 48109 USA; email: qmei@umich.edu.

1559-1131/2020/06-ART14 $15.00
https://doi.org/10.1145/3395050

1 INTRODUCTION

Alice was wandering through restaurant reviews in the Yelp app. It mentioned that the restaurant reminded people of a classical scene in the movie "*Pretty Woman*," which made her very eager to watch the clip. She had to go back to the OS home screen and then launch the Youtube app. When the landing page of the app was loaded, she looked for the search bar, typed in a query, and navigated through a few results before finding the clip of the scene. Then the moment was gone.

Smartphone applications (a.k.a., apps) are already indispensable in our everyday lives. It is reported that the traffic from smartphones has already surpassed that from PCs, and apps have become the major entry point to the Internet [7]. Smartphone users usually "navigate" among a set of apps on smartphones to satisfy their information needs [18]. We call such behaviors as *inter-app navigation* in this article. In general, inter-app navigation could be either intentional to complete a specific task, or unintentional to occasionally or habitually use some apps together. The above-mentioned scenario of Alice is a very typical example of intentional navigation. As for unintentional navigation, one may switch to the email app to check new emails occasionally while using other apps. Inter-app navigation is quite similar to Web browsing where Web users navigate among Web pages. However, compared to the Web users who can easily navigate through the *hyperlinks* from either anchors in Web pages or bookmarks in the browser, app users like Alice often have to go through a frustrating procedure to manually switch from one app to another. This frustration is amplified when there are more and more apps installed on a device and when the user needs to switch back and forth between various apps.

Inter-app navigation is indeed non-trivial for user interaction and has been drawing a lot of attention. Some solutions have been proposed to help bridge the "isolated" apps [11, 37, 48]. In particular, the recent concept of "deep links" has been proposed to facilitate the navigation from one app to another. Essentially, deep links are URLs that point to specific locations inside an app. When a deep link is invoked, the corresponding app will be launched to open the corresponding page if the app has already been installed on the device. Today, all major mobile platforms, including Android, iOS, and Windows, have supported deep links, and have been encouraging developers to implement and release the deep links to their apps [3–5, 8]. With deep links, users can create shortcuts for pages inside an app so that they can directly visit the target page by just one click, which is quite similar to opening a bookmark in browsers. App developers can embed a set of deep links from other apps into their off-the-shelf apps to provide additional features to facilitate the navigation between apps, which is similar to the hyperlinks between Web pages.

Although the deep link is a desirable concept to facilitate inter-app navigation, it has been reported to have a poor coverage [10]—only approximately 25% of apps have provided deep links, and only a small number of app pages can be directly accessed via predefined deep links. What is holding it back? While there are historical reasons such as the fear of "stolen page views," the most straightforward reason is that it takes non-trivial manual effort to do so [33]. Indeed, unlike hyperlinks that are standardized and facilitated by the HTTP, there is no gold standard for deep links. App developers, platforms, and app stores all have to make substantial efforts to define, implement, standardize, and maintain deep links. Without teleportation spell, tunneling through the castles means tedious human work.

Is it that bad? Perhaps not really. In practice, navigation between apps is actually performed between "*pages*" inside apps. Analogical to a website, an app also contains various pages, e.g., the landing page, advertisement pages, content pages, and so on. Not all apps need to be deeply linked, and for those who do, not every single page needs to be linked from the outside of apps. If one can distinguish those "important" pages from the rest and understand their relationship in navigation, then the effort can be significantly reduced. Developers may focus on creating deep links for only these pages; they may also target the "partner" apps/pages. Furthermore, if the next

page that a user is likely to visit can be predicted and a method that can dynamically generate deep links is available [10], runtime facilities such as Avitate [11] and FALCON [48] can be leveraged to navigate users to the destination page more efficiently. At the minimum, even if either of the wishes comes true, a simple estimation of the potential benefit of deep links could make the "deep link advocacy" more persuasive. Answers to these questions exactly requires an in-depth analysis of the inter-app navigation behavior, which unfortunately does not exist in literature.

This article conducts an empirical study of the page-level inter-app navigation behavior of Android users. We present an in-depth analysis of inter-app navigation based on a behavioral data set collected from 64 users in 389 Android apps, consisting of about 0.89 million records of app-page visit. Inspired by the Web search and browsing, we classify the *informational* pages where users tend to stay and interact with an app for long periods of time, and the *transitional* pages where users only stay awhile or tend to skip when they navigate among informational pages. We investigate why inter-app navigation happens, how inter-app navigation performs, and what are the features of inter-app navigation under different contexts. We believe that the results of our empirical study can provide useful insights to various stakeholders in the app-centric ecosystem.

The main contributions of this article can be summarized as follows:

- We reveal the number of active pages of an app and how long users tend to stay on these pages. We find that users usually access and spend their interaction time on only a small set of pages in an app, i.e., the dwell time of more than 95% pages is less than 5 seconds. In other words, a substantial proportion of pages are not frequently required.
- We propose a classification model to distinguish pages in an app into informational pages and transitional pages, according to the distribution of users' dwell time of these two kinds of pages. The dwell time in transitional pages follows a *Gaussian* distribution with a mean of 4.3 seconds and a small value of variance, while the dwell time of informational pages follows a *log-normal* distribution with the mean dwell time of 29.4 seconds and a large variance, indicating that the time spent on this type of pages can differ dramatically.
- We study the intention of inter-app navigation by analyzing which apps/pages are more likely to be visited during inter-app navigation. We find that apps can be classified into clusters, i.e., apps from different categories can cooperate with one another to accomplish some specific tasks. To further explore the validity of our observations, we conduct a user study to label why inter-app navigation happens. Results indicate that about 80% of inter-app navigation instances are really intentional to complete a specific task.
- We demonstrate the inefficiency caused by transitional pages in the inter-app navigation. The average time of inter-app navigation is around 13 seconds when navigating among apps. To our surprise, the transitional pages account for 28.2% of all navigation time along with tedious manual steps of switching between two information pages. Such an overhead of inter-app navigation is really non-trivial and can probably compromise user experience.
- We reveal the contextual influences on inter-app navigation where users tend to navigate to different kinds of apps under different network conditions, and the frequency of inter-app navigation varies among different app categories.
- We propose some practical implications based on the findings, to facilitate app developers, OS vendors, and end-users. For example, we make a proof-of-concept demonstration by employing a machine learning-based approach to accurately predicting the next informational page from the current state and thus recommend a navigation path between two pages (i.e., a potential deep link) to reduce transitional pages.

Indeed, there have been various efforts made on understanding and predicting app usage. However, to the best of our knowledge, we make the first empirical study on inter-app navigation

at the page level rather than the app level. Therefore, we can more comprehensively study, and even further predict the user behavior at a finer granularity. For example, we demonstrate that the page-level navigation can be accurately predicted, so that users can choose to directly land on the desired pages without having to manually switch several apps, and bookmark the navigation paths for future use. As a result, our work can provide useful suggestions to users, developers, and OS vendors, for better user experiences on apps.

The rest of this article is organized as follows. Section 2 presents the background of Android apps and formulates the inter-app navigation at the page level. Section 3 presents our dataset and how it was collected. Section 4 describes an empirical study of inter-app navigation at page level and characterizes the intention, cost and contextual features of inter-app navigation. Section 5 presents some practical and potential useful implications that can be explored based on our empirical study. Section 6 discusses the limitations of our work. Section 7 compares our work with existing literature and Section 8 concludes the article along with the future outlook.

2 PAGE-LEVEL INTER-APP NAVIGATION: IN A NUTSHELL

In a sense, we can simply define the problem of inter-app navigation as the action switching from one app to another. Note that the apps involved in an instance of inter-app navigation are not necessarily used together to complete one task, i.e., the instances of inter-app navigation are not always intentional. A user may switch to another app occasionally for some daily routines such as checking email updates and SNS messages, or for being interrupted by system notifications such as news feeds and calendar events. However, as we show later, most instances of inter-app navigation are rather intentional, i.e., users switch between apps to accomplish a specific task.

Essentially, the basic user-interaction unit of an app is a "*page*." Therefore, when performing inter-app navigation, we actually leave the current page that we are browsing, and move to a page of another app. In this section, we first give some background knowledge of app pages on the Android platform, and draw an analogy between app pages and Web pages. Then, we define the concepts of page-level inter-app navigation.

2.1 Concept of App Pages

Due to the platform openness, this article focuses on the page-level navigation between Android apps. At the programming model level, an Android app [2], identified by its `package name`, usually consists of multiple `activities` that are related to one another. An `activity` is a component that provides an interface for users to interact with, such as dialing phones, watching videos, reading news, or viewing maps. Each activity has a unique `class name` and is assigned a window to draw its graphical user interface. In general, an activity is regarded as a template that can instantialize a set of pages with the same structure but different contents. For example, all the Youtube video pages belong to `VideoActivity`, but they have different contents. For simplicity, in this article, we use the term *page* or *app page* to represent the activity that has a UI for users to interact with on their smartphones.

Although we study only Android apps, the similar concept of activity/page also exists in iOS apps (called `Scene`) and Windows Mobile apps (called `Page`). Therefore, the empirical approach can still be generally applied on other platforms.

2.2 A Conceptual Analogy of App Pages and Web Pages

For ease of understanding, we can draw an analogy between Android apps and websites, as compared in Table 1. An Android app can be regarded as a website where the package name of the app is similar to the domain of the website. An activity can be regarded as a template of Web pages and an instance of an activity is like a Web page instance. Different Web pages of the same

Table 1. Conceptual Comparison between Android Apps and Web

Concepts of Android Apps	Concepts of Web	Example
app	website	youtube
package name	domain	www.youtube.com
activity	Web page template	https://www.youtube.com/watch?v=[xxx]
activity instance	Web page instance	https://www.youtube.com/watch?v=qv6UVOQ0F44

template differ in parameter values in their URLs. For example, URLs of different video pages on the Youtube website have the format of https://www.youtube.com/watch?v=[xxx]. This template can be regarded as the VideoActivity in the Youtube app.

2.3 Defining Inter-App Navigation

An app usually consists of various pages, and different pages indeed play different roles under different contexts. Inspired by the preceding study on the Web [14] that classified Web pages according to the interaction density and length, we classify the pages of Android apps into two categories:

- **Informational page.** These pages serve to provide content/information, such as news article pages, video pages, mail editing pages or chatting pages, and so on. Users accomplish their desired intention of one app in its informational pages.
- **Transitional page.** These pages are the intermediate pages along the way to reach the informational pages. Some of the pages serve to narrow the search space of possible informational pages and direct the users to them. Typical examples include the system transitional pages, such as Launcher, where users can select the desired app that may contain the potential informational pages, and in-app transitional pages, such as ListPage in a news app, where users can choose a topic to filter the recommended articles. Other pages are somehow less helpful for navigation from the users' perspective, such as the advertisement pages and the transition splash pages.

It should be mentioned that the same page could be viewed as a transitional page by one user, but regarded as informational page by another, according to their application context. In this article, we simply classify these two kinds of pages according to the length of user-interaction time spent on them, or more specifically, the *dwell time*. As illustrated later in Section 4.2, we can split the pages into two significant clusters that have extremely different distributions of the dwell time.

In this article, we focus on the page-level app navigation. So, we define that the app navigation is the process of reaching the target informational page from the source informational page, probably walking through multiple transitional pages. More specifically, the inter-app navigation is the navigation whose source and target informational pages belong to different apps. The inter-app navigation could be either intentional or unintentional. The intentional inter-app navigation means that the target informational page has some logical relationships with the source informational page, e.g., navigating from a restaurant page in a food app to a taxi-booking page in a map app. In contrast, the unintentional inter-app navigation implies that users could occasionally switch between two apps, e.g., a user reads a news app and then switches to an email app when a notification of new emails is prompted.

To illustrate the navigation process, a simple scenario is described in Figure 1. Suppose a user has found a ramen restaurant in Yelp, and wants to read detailed comments about this restaurant in Reddit. To accomplish this task, the user has to first quit Yelp app to the OS launcher, click the

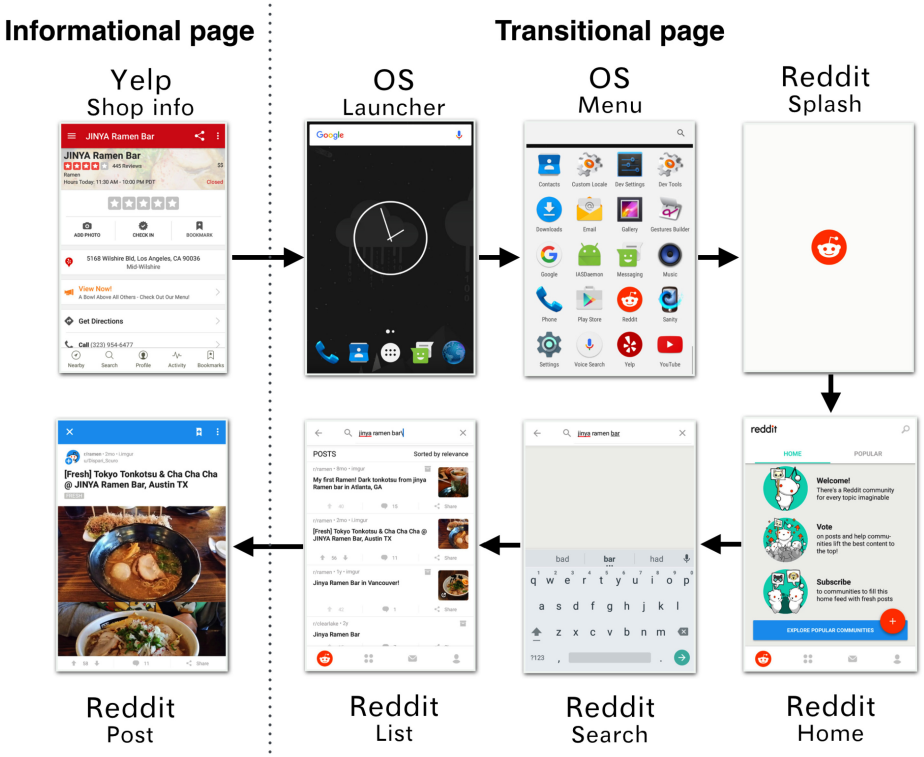


Fig. 1. An example showing a user's navigation process from a ramen restaurant page in Yelp to read comments in Reddit.

app menu, and then opens Reddit app. Afterwards, the user needs to start from the home page of Reddit app, walk through a series of transitional pages such as Search page and List page, and finally reaches the post page to read the comment about this restaurant. In this scenario, the true intention of the user is to directly visit the target informational page, i.e., the post page, to read the comment. However, the user needs to pass through various transitional pages before landing on the final target page.

Intuitively, users tend to spend more time on informational pages and avoid transitional pages. Some apps adopt flat UI design and carefully organize its functionalities and information display to reduce the number of transitional pages for reaching the target informational page. However, due to the fact that most of the apps provide only dedicated functionalities, users usually need to switch among multiple apps to achieve a task, resulting in notable time cost on inter-app navigation.

We formally define a page-level app navigation as a triple $<s, t, \psi>$, where s is the source informational page, t is the target informational page, and $\psi = [p_i]$ is a sequence of transitional pages that represent a navigation path from s to t. We assume that the screen state of the smartphone is always *On* in a whole navigation process. ψ can be empty, indicating that there is a direct link from s to t without having to pass through any transitional pages.

3 DATA COLLECTION

To study the inter-app navigation on Android platform, we conduct a field study by collecting behavioral data from real-world users. In this section, we describe the design of our data collection tool and the description of the data set.

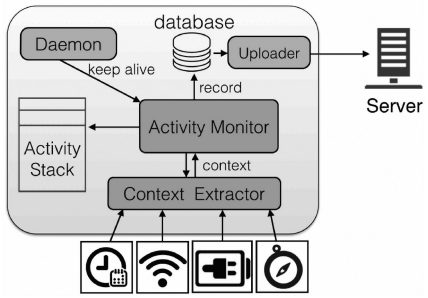


Fig. 2. The architecture of the data collection tool.

Table 2. Data Example

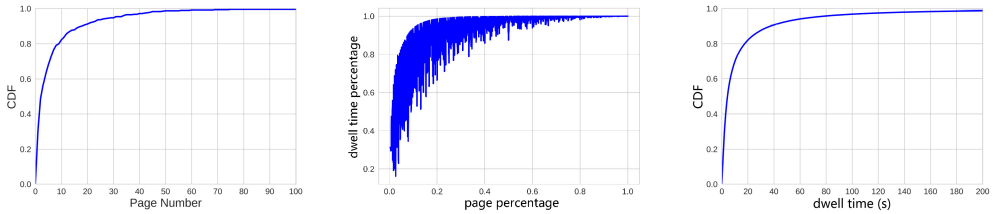
User ID	App	Activity	Time	Network
99000478917655	com.eg.android.AlipayGphone	fund.ui.FundMainNewActivity	Mar. 31, 15:25:0	cellular
353925062095766	com.tencent.mm	ui.LauncherUI	May 2, 22:17:13	Wi-Fi
863473022828516	com.android.contacts	activities.ContactDetailActivity	May 9, 14:49:26	Off
357523051693018	com.sina.weibo	page.NewCardListActivity	Feb. 11, 15:20:50	cellular

As mentioned above, we focus on the page-level inter-app navigation. To this end, we develop a tool to monitor the system events representing such behaviors, as shown in Figure 2. The tool is a monitoring app running in the background of Android platform. It consists of four modules, *Activity Monitor*, *Context Extractor*, *Daemon*, and *Uploader*. The activity monitor reads the top of the activity stack from the system every one second and produces a record entry whenever the top activity changes, indicating that a transition between app pages occurs. Meanwhile, the monitor invokes the context extractor to collect the user's context information, including the type of network connection (cellular/Wi-Fi/Off) and the local time. In addition, every time the user turns on or off the screen, the tool will also produce a record, indicating the screen status (ON/OFF). The records are stored in a local database, and the uploader will upload the records to our server once a day at night when the device is charging and connected with Wi-Fi. Therefore, the tool does not influence the normal usage of mobile devices. To keep our tool from being killed by the system, the daemon module periodically checks the status of the tool and re-launches it if necessary. To protect user privacy, we anonymize the device ID with a hash string.

We recruited some student volunteers without compensation in the data collection via an internal social network site in Peking University. We got 64 college students who fully agreed with the collection statements and were willing to participate in the user study. These students were all 18 to 25 years old, consisting of 38 male and 26 female. They were all regular smartphone users and were reported to be familiar with normal operations of Android system. We installed the data-collection tool on each volunteer's Android smartphone, and ensured that the data can be correctly collected and uploaded. The data collection lasted for successive three months, and we finally collected 894,542 records, containing 3,527 activities from 389 apps.[1]

Table 2 provides some illustrating examples: `User ID` denotes the unique and anonymized identifier of the user; `App` and `Activity` denote which app and page (i.e., activity) that the user is

[1]The data collection and analysis process was conducted with IRB approval from the Research Ethic Committee of Institute of Software, Peking University. The dataset can be found at https://github.com/pkuoslab/iandata.



(a) Distribution of the page number of user-visited apps (b) Distribution of dwell time of pages accounting for the interaction (c) Distribution of dwell time of pages

Fig. 3. The distribution of page visit in apps.

interacting with, respectively; `Time` refers to the local Beijing time when the page is visited; `Network` indicates the network type when the page is visited, i.e., cellular, Wi-Fi, or offline.

4 EMPIRICAL RESULTS AND ANALYSIS

In this section, we present an empirical study on our collected user behavioral data. We first study the page-level access of mobile apps. Then, we propose a clustering approach to identifying informational pages and transitional pages, based on which, we investigate the intention, cost, and contextual features of inter-app navigation, respectively.

4.1 Characterizing Page-Level Access

We investigate how users interact with an app at the page level. Recalling the previous analogy between apps and websites, we focus on three aspects, i.e., the number of user-accessed pages in an app, the proportion of frequently accessed pages, and the dwell time of pages.

We first report the distribution of the number of user-accessed pages of an app in Figure 3(a). Interestingly, we can find that users visit only less than 10 pages in over 80% apps. Then, we count the number of all unique activities visited by the app users, and divide the number by the total number of activities, to compute the page visit ratio of the app. The result shows that users visit only less than 20% of activities for most apps (about 87%). Such a result indicates that users visit only a small subset of pages in mobile apps. However, users can access more than 20 pages in less than 5% apps. Such an observation is explainable, because some popular apps, such as WeChat and AliPay, are likely to integrate rich functionalities to meet various kinds of user requirements. Due to the limited screen size of smartphones, each functionality is usually implemented in a dedicated app page. Therefore, popular apps may have more pages visited by their users.

Next, we want to know how much users interact with an app. Just like the similar experiences on the Web pages [31], a simple but intuitive measure is based on the length of dwell time spent on the pages. To this end, we report the CDF of dwell time of all pages accounting for the app that these pages belong to, as shown in Figure 3(b). We can see that less than 10% pages account for more than 90% dwell time of an app. Such an observation is compliant to the preceding distribution of page numbers.

We further explore how long users interact with a page by exploring the length of dwell time as shown in Figure 3(c). Interestingly, we can see that users stay for less than 5 seconds on more than 70% pages. Hence, we can infer that users tend to perform very few interactions on most pages. In contrast, users tend to spend more than 60 seconds on some pages (less than 5%), implying more user interactions.

From these results, an intuitive finding is that users tend to interact with only a very small portion of pages in every single app. In the next section, we further explore how such behaviors can affect the inter-app navigation.

Table 3. Top Five Pages with Shortest and Longest Average Dwell Time

Category	Page Name	App Name	Page Description	Average st
Shortest dwell time	lenovo.Launcher	Lenovo System	System Home Screen	1.08 s
	kugou.LockScreenActivity	Kugou Music	Music LockScreen	1.28 s
	dazhihui.InitScreen	dazhihui Investment	App Home Page	1.67 s
	miui.home.Launcher	Miui System	System Home Screen	1.72 s
	hupu.ListNewActivity	Hupu Sports	News List Overview	1.85 s
Longest dwell time	mobileqq.TextPreviewActivity	QQ	Chatting Interface	73.6 s
	android.AlarmAlertFullScreen	System	System Clock	72.1 s
	papd.HealthDailyPopActivity	Pingan Doctor	Healthy Daily News	66.3 s
	renren.ChatContentActivity	Renren	Chatting Interface	64.4 s
	meitu.mtxx.MainActivity	Meitu Xiu Xiu	Photo Beautification	62.8 s

4.2 Clustering the App Pages

According to the definition in Section 2, to identify the inter-app navigation, we should distinguish informational pages from transitional pages in our collected records of page usage. Intuitively, users are likely to spend longer time on informational pages. In contrast, the transitional pages are mainly visited for navigation purpose and thus users are likely to stay for a short time. Since our data records the sequence of pages, we are able to calculate the time interval for every single page by the timestamp when this page is visited and the timestamp when its subsequent page is visited. As each page could appear several times, we can obtain the distribution of the dwell time for every single page.

Table 3 lists some typical pages with the longest and shortest average dwell time. From the name of these pages, we can speculate that the pages with longer dwell time are more likely to be the pages that provide substantial information and available services, while the pages with shorter dwell time have names containing "list" or "launcher," indicating that these pages are more likely to be transitional pages. These examples imply that the dwell time can be a vital clue to identify transitional pages.

Inspired by the work by Van Berkel et al. [44], who proposed a classification approach to determining the session time threshold of user interaction, we design an unsupervised clustering approach to separating the pages according to the users' dwell time on the pages. A simple way is to leverage the average dwell time by all users as the threshold. However, simply using the average dwell time cannot reflect the overall dwell time distribution of mobile users. Thus, we represent each page with the probability distribution of all visits' dwell time.

To measure the distance between the dwell time distribution of pages, we can use the classic Kullback-Leibleer divergence (KLD) [27] and its symmetric version, i.e., the Jensen-Shannon divergence (JSD) [21]. In this article, we choose JSD to measure the distance between the distribution of every pair of pages, because the JSD's symmetric property makes it more appropriate for clustering. Then, we use the spectral clustering method [35] to cluster the pages. More specifically, such a process first learns a low-dimensional representation of every single page according to the distance matrix between the pages, then deploys the K-means algorithm to cluster the pages based on the low-dimensional representations. We assign different numbers of clusters in the K-means algorithm and select the optimal number of clusters through the Silhouette score [40]. Figure 4 visualizes the clustering results with different numbers of clusters using the Fruchterman-Reingold force-directed algorithm [26]. The best clustering results are obtained when the number of clusters equals 2, which well verifies our intuition that the pages can be classified into two categories.

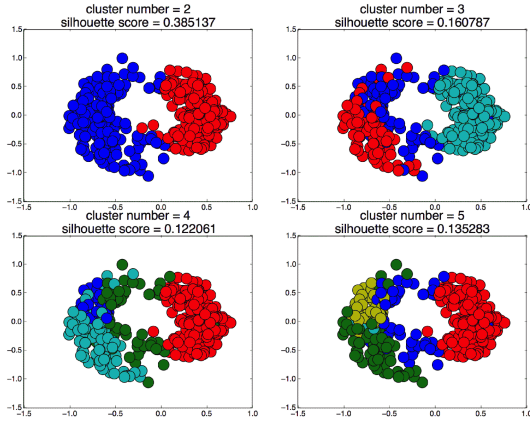


Fig. 4. Clusters of pages. Visualized using the Fruchterman-Reingold force-directed algorithm. The optimal cluster number is 2.

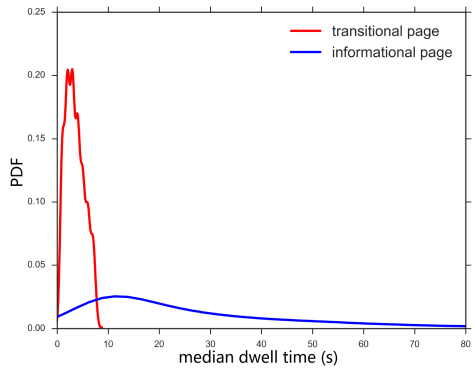


Fig. 5. Probability density distribution of dwell time of two types of pages.

To further validate the distribution of dwell time of these two types of pages, for each page, we calculate the median of its dwell time and then depict the Probability Density Function (PDF) of all pages in Figure 5. The blue curve indicates the PDF of dwell time of informational pages, and the red curve indicates transitional pages. The distribution of the transitional pages's dwell time performs as the Gaussian distribution with a mean dwell time of 4.3 seconds and a small value of variance. The distribution of informational pages' dwell time follows a log-normal distribution with the mean dwell time of 29.4 seconds and a large variance, indicating that the time spent on this type of pages can differ dramatically.

To evaluate the derived cluster, we randomly select 50 pages from each of the two clusters, retrieve an instance of each page according to its activity name, and capture a screenshot of the page. Then, we manually classify whether each page is informational or transitional based on its activity name, median dwell time and the screenshot. The result of such manual classification is completely accorded with our cluster labels, i.e., the accuracy is 100%, which can demonstrate the reliability of the proposed clustering approach.

After identifying informational and transitional pages, we extract all the instances of navigation from the records. Since we focus on only the navigation between different apps, we select all the inter-app navigation. The amount of inter-app navigation records is 41,619, which accounts

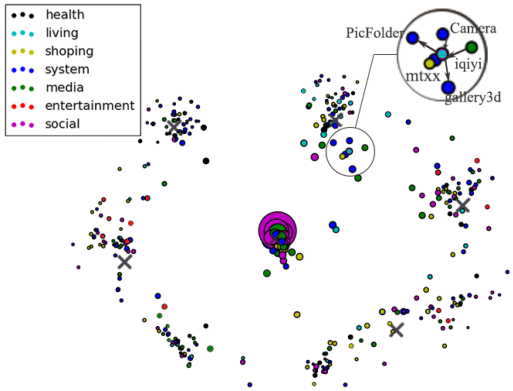


Fig. 6. Inter-app navigation graph connecting informational pages from different apps. Each node is labeled by the category of the app. Edges are not displayed to clearly show patterns.

for 26.9% of the total navigation records. Our following analysis is conducted on these inter-app navigation records.

4.3 Intensions of Inter-App Navigation

We first investigate why the inter-app navigation occurs. To this end, we draw a graph representing all instances of inter-app navigation by removing the transitional pages. As illustrated in Figure 6, each node corresponds to an independent informational page, and is labeled with the category information of the app that the page belongs to. We assign an edge between two nodes from different apps if there exists an instance of navigation between them. We then measure the distance by the static transition probability between every single informational page, which is calculated by the frequency of a particular navigation divided by the total navigation times. In this way, we can obtain the distance matrix of the graph and visualize it with the classic Fruchterman-Reingold force-directed algorithm [26], where the size of each node is determined by its out-degree.

Interestingly, there are some significant clusters. In particular, there is a "core" cluster, in which most of the pages belong to the SOCIAL, SYSTEM, and MEDIA apps. Other clusters around the core include smaller nodes from different categories. From the distance matrix, we find that those small clusters are mutually isolated with each other, while the nodes in these clusters are strongly cohesive. Additionally, all these small clusters have a strong correlation with the core cluster. We also find that apps consisting of these small clusters belong to different categories.

For example, Figure 6 shows the pages of one cluster that is zoomed in. We can see that these pages come from a photo beautification app called `MeiTuXiuXiu` (mtxx for short), a gallery app called `PicFolder`, a video app called `iQiyi`, system camera and gallery, along with the directed connections among them. Such an observation may reflect the following scenario: A user captures a screenshot from a video page of `iQiyi` or takes a photo using the camera, then he/she navigates to the edit page in the `mtxx` and decorates this image. Eventually he/she saves the image into the `PicFolder`.

However, as explained before, inter-app navigation could be *intentional* where the two apps involved in an inter-app navigation are used to complete a specific task, or *unintentional* where the user just switches between two apps occasionally or habitually. As a result, we cannot determine whether apps involved in a cluster are "really" used together to accomplish a specific task. To address this issue, we further conduct a user study to estimate the proportion of intentional inter-app navigation. We randomly choose 100 instances of inter-app navigation from the dataset, and

Table 4. Results of the User Labelling Experiment

		Stu1		
		intentional	unintentional	indistinghuishable
Stu2	intentional	58	0	2
	unintentional	1	15	1
	indistinghuishable	2	4	17

manually capture screenshots for all the involved pages. We ask two students to separately label each navigation as intentional if the target informational page has relationship with the source informational page, unintentional if the two informational pages are just casually used together, or indistinguishable if it is hard to determine. It should be noted that the page content on the screenshot may not be the same with the original one when the navigation was recorded during the data collection, because app pages are dynamic and the contents are always changing. So, we require that the labels of inter-app navigation should be determined not by the actual content, but by the semantic relationship between the informational pages. Table 4 shows the reported results. To better illustrate the results, we compute the Cohen's kappa coefficient to measure the consistency of labels, and the value is 0.82, indicating that the labels annotated by the two students are quite consistent. Considering the instances of navigation that receive the same labels from both two students, 58 are intentional and 15 are unintentional, meaning that about 80% of agreed inter-app navigation instances are regarded as intentional. Furthermore, we find that the apps of 95% intentional navigation instances belong to the same small cluster in Figure 6, while all the unintentional navigation instances are related with apps in the core cluster.

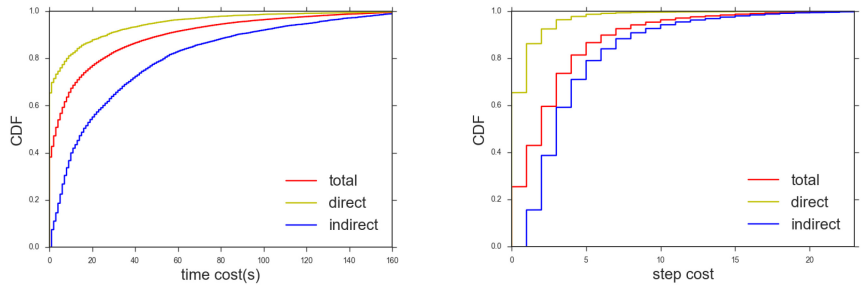
Based on the preceding findings, we can then conclude that inter-app navigation can be intuitively classified into some clusters. There is one cluster where apps are likely to be related with unintentional inter-app navigation. Each of the other clusters probably represents a specific usage scenario, where apps from different categories are collaboratively used to accomplish a task. Such findings imply the future work that we can combine session-level analysis to comprehensively specify the task characteristics [23].

4.4 Cost of Inter-App Navigation

Intuitively, users would like to directly navigate between two informational pages with as less transitional pages as possible on the navigation path. To explore whether the current inter-app navigation performs in such a user-desired fashion, we use two metrics to quantitatively measure: (1) *time cost*, which is the aggregated value of dwell time for all the transitional pages between the source informational page and target informational page; (2) *step cost*, which is the number of transitional pages on a navigation path.

The distributions of *time cost* and *step cost* for all instances of inter-app navigation are illustrated as the red curve and labeled as total in Figure 7. The average *time cost* is 13.01 seconds, and the average *step cost* is 2.69, indicating that users have to pass through about 3 pages and spend 13 seconds before reaching the desirable pages. Additionally, we find that the time cost on the transitional pages takes up to 28.2% of the total time in the entire inter-app navigation, which is the total time cost of the source informational page, the subsequent transitional pages and the target informational page during a single navigation process. Therefore, the overhead of inter-app navigation is non-trivial for smartphone users.

We observe that most instances of inter-app navigation do not jump directly from the source app to the target app. Instead, a large number of inter-app navigation instances involve pages from



(a) CDF of time cost. (b) CDF of step cost.

Fig. 7. Overhead of inter-app navigation.

the Android system-wide apps such as `android.Launcher`, which is the system home screen, or `android.RecentsActivity`, which is the system list page showing all the recently used apps. We call this kind of navigation "indirect navigation." However, a small number of inter-app navigation instances do not contain these system pages, but consist of transitional pages only from the source and target apps, e.g., when user clicks a `Share to FaceBook` button in the Reddit app, the FaceBook app will be directly launched rather than having to pass through system pages. We call this kind of navigation as direct navigation between apps.

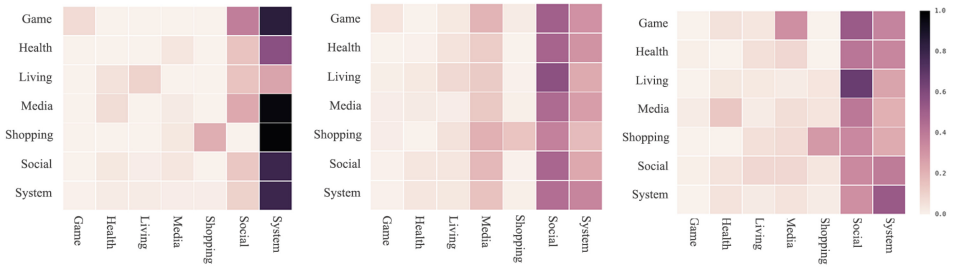
Comparing the distribution of *time cost* and *step cost* between direct navigation and indirect navigation in Figure 7, we have the following observations:

- The average time costs of direct and indirect navigation are 8.97 and 24.43 seconds, respectively; the average step cost of direct and indirect navigation are 0.97 and 3.86, respectively. This result indicates that the overhead of direct navigation is much smaller than that of indirect navigation.
- All the instances of indirect navigation have at least one intermediate step, i.e., the transitional page belongs to the Android system. In contrast, 66% instances of direct navigation have no such cost, indicating no navigation overhead. As a result, the direct navigation is more appreciated.

We further explore how direct navigation could happen. Some navigation instances contain the system-wide apps such as Call, Maps and Camera, which can be directly invoked via APIs provided by the Android system. Other links navigate the users directly from the source page to the target page of third-party apps without passing through system navigational pages, such as Launcher. These links are likely to be implemented by the emerging popular *deep link* [8]. Similar to hyperlinks of Web pages, deep links also employ URI to locate pages and can be executed to directly open the target page from the source page. For example, when a user reads an interesting article and wants to share it on Facebook, he/she can click the sharing button that executes a deep link to navigate Facebook interface without passing through multiple transitional pages.

We are interested in whether the direct navigation is achieved by means of deep link. To this end, we first label the category information of each app,[2] and filter out the system-wide apps. The remaining third-party apps account for 82.7% of all apps. Next, we check the manifest file of the third-party apps that contain the target pages to ensure they provide deep link interfaces. Finally, we employ a mature Android analysis tool, called IC3 [12], to check whether the source

[2]In this article, we simply use the categorization system from a leading Android market, called Wandoujia http://www.wandoujia.com.



(a) Offline (b) Wi-Fi (c) Cellular

Fig. 8. Heatmaps of transitional probability between pages from different categories. The three figures refer to the transition under different network types, respectively.

page actually invokes the target page's deep link interfaces. Given two apps, IC3 can compute all the inter-component communications (ICC) between these apps. If there exists an ICC between the activities of the source page and target page, then we can conclude that the source page has an entry to invoke the target page's deep link interfaces. Results show that within the direct navigation pairs, all the target pages provide deep link interfaces and all the source pages have an entry to invoke the corresponding interfaces, indicating that these direct navigation instances are highly likely to be implemented by deep links.

4.5 Contextual Features of Inter-App Navigation

Finally, we are interested whether the context can possibly influence the inter-app navigation, i.e., whether some kinds of navigation are more likely to occur under specific contexts. From the features of our collected data, we focus on two types of contexts, i.e., *network* and *time*.

We first analyze whether the type of network affects user's preference of inter-app navigation. We classify all instances of navigation into three clusters based on their network type. For each cluster, we calculate the static transition probability between each pair of informational pages. Then, we plot a heatmap for each network-type cluster, where pages are denoted by their categories. The source pages are plotted on the Y-axis and the target pages are plotted on the X-axis. The results are presented in Figure 8. It is observed that only the SYSTEM is significant when the users are offline, while all the other categories are rather shallow. It indicates that users may mainly switch between SYSTEM apps such as SMS, Dialing, Camera, and so on, when their devices are not connected to the network. When the network is under Wi-Fi and cellular, the inter-app navigation is more likely to happen, and SOCIAL apps are usually involved. However, the frequency of different kinds of inter-app navigation can vary between these two network types. When users are under Wi-Fi, MEDIA apps are more likely to be linked with other apps. The reason may be that MEDIA apps such as video players may require high-bandwidth and stable connection. In a sense, the network condition can be employed as a potential feature for predicting inter-app navigation.

Then, we investigate whether some kinds of navigation are *time* sensitive. We compute the visit frequency of a page during a specific period of time, with a metric, namely, Usage Time Gap (UTG), which is defined as the interval between current time when a page is visited and the time when this page was latest visited. We draw the boxplot of the UTG for each category in Figure 9. The UTG varies among different app categories. For SOCIAL apps, the median value and variance of UTG are the smallest, indicating that users may frequently visit pages of these apps in a short time interval. The median UTG of Media and Living apps are much larger than others, indicating that these apps are not "daily routines" for users. In other words, these apps are not likely to be revisited by users in a short time interval after they are launched. Intuitively, with the metric such as UTG,

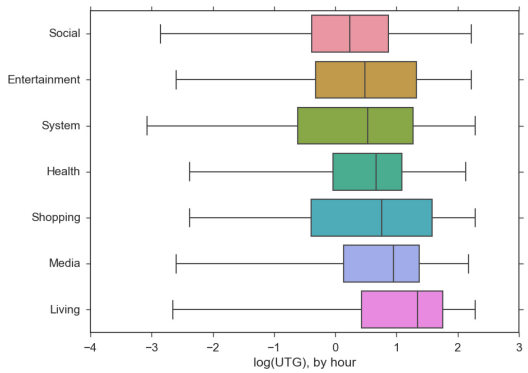


Fig. 9. The boxplot of the usage time gap for seven categories.

one can infer which pages are more likely to be visited. Combined with the sequential information between two informational pages, it would be possible to prefetch these pages according to their last visited time.

5 IMPLICATIONS

In this section, we discuss how our preceding findings can be explored to improve user experiences. On the one hand, we demonstrate that the page-level inter-app navigation can be predicted so that promising techniques, such as pre-fetching resources, pre-launching the app, and creating user-defined deep links, can be adopted to facilitate the inter-app navigation. On the other hand, we discuss the possibility of page-level app installation to provide Web-alike experience for mobile users.

5.1 Predicting Page-level Navigation

The most intuitive implication is whether the navigation could be predicted, i.e., given a source informational page s, which target informational page t a user is likely to visit? If the potential page candidates can be accurately predicted, then we can provide users some recommendations (either a single page or a composition of various pages) when they are browsing an informational page. Hence, users can automatically navigate to the target page without manually switching among apps, and bookmark these recommendations for the future use. In addition, users can choose to navigate to the target pages by skipping all transitional pages to reduce system-wide resource overhead.

5.1.1 Prediction Problem Statement. To make a proof-of-concept demonstration, we use a machine learning technique to conduct the prediction. Intuitively, such a task can be treated as a *classification* problem. We can treat a pair of (s, t) as an instance. Each pair of navigation (s, t) can be represented with different types of features such as the basic features of source informational page s and target informational page t, along with the contextual features when the navigation occurs.

With the actual inter-app navigation behaviors in our collected data, we can observe many positive instances of inter-app navigation pairs. However, given a source informational page s, it is unknown which target information pages t will never be visited. Theoretically, this is known as a typical *one-class* classification problem [43] in literature. Here, we adopt a state-of-the-art algorithm *PU* proposed by Liu et al. [29, 30] to solve this problem. The basic idea of the algorithm is to use the positive data to identify a set of informatively negative samples from the unlabelled

data. (Here, each possible pair of source and target informational pages can be treated as an unlabeled instance.) In our experiments, we compare different variants of the algorithm, including S-EM, Spy+SVM, Spy+SVM-I, NB+EM, NB+SVM, and NB+SVM-I, and a naive solution that simply samples some random target pages for each source page and treats them as negative samples.

The overall classification process can be simply described in Algorithm 5.1. In the beginning, the algorithm generates an unlabelled data set U by randomly sampling some target pages t for each source page s. Then the algorithm identifies informatively negative samples according to the positive training data. For prediction, given a source page s, we can rank the candidate target page t according to the probability of the pair (s, t) belonging to the positive class.

ALGORITHM 5.1: One-class classification algorithm for prediction.

Input: PU Method $\mathcal{PU}$, Positive dataset P, Candidate Page Set S=($p_1, p_2, \ldots, p_N$)
Output: Recommendation List L

$U \leftarrow geneate\_unlabelled\_data(P)$
$N \leftarrow extract\_negative(U, P)$ according to [30]
$\mathcal{PU}.classifier.fit(P \cup N)$
foreach $tp \in S$ **do**
  $prob[tp] = \mathcal{PU}.classifier.predict\_probability((s, t))$
end
$L \leftarrow sort\_by\_probability(P, prob)$
return L

5.1.2 Feature Selection. In Table 5, we illustrate two types of features to represent each navigation pair (s, t), including the task-specific features of the pages and the contextual features associated with the pairs. For the task-specific features, three different features are identified including the name of the pages, the name of the app that the pages belong to, and the name of the category that the app belongs to. The contextual features include the network type and UTG, representing the elapsed time of the target app, since it was used last time until now. Note that all the features except UTG are nominal variables, so we process them in one-hot encoding.

5.1.3 Proof-of-Concept Prediction Results and Analysis. We report the prediction results with different combinations of features to illustrate their significance. As a ranking problem, we evaluate the prediction result according to a well adopted measure for ranking, called *Mean Reciprocal Rank (MRR)* [6]. The MRR of a recommendation list is the multiplicative inverse of the rank for the correct answer. The MRR score and the prediction performance are positively correlated, i.e., higher MRR scores imply more accurate prediction results. Indeed, there are different variants of the PU algorithm. Therefore, we first randomly filter 30% of the data reserved for subsequent evaluation, and then conduct a five-fold cross validation to compare different variants of the algorithms. The results show that S-EM algorithm yields the best results.

We compare the proposed approach with two other commonly used algorithms including *Most Frequently Used* (MFU) approach and the *Markov Chain*-based approach. The *MFU* approach ranks the pages according to their usage frequency, and the *Markov Chain*-based approach is ranked by the static transition probability for a page to be navigated from the given source page. To evaluate our approach, we use the 30% data as the test set that is previously reserved for evaluation and the remaining 70% data as the training set.

Figure 10 presents the prediction results with different algorithms and features. First, the classification algorithm with all the features outperform the naive Most Frequently Used and Markov Chain methods, with its MRR score of 0.328, implying that the actual page is on average ranked at

Table 5. Feature Table

Feature Type	Feature Name	Dimensions
task-specific feature	Page name	816*2=1632
	app name	245*2=490
	category id	20*2=40
contextual feature	Network type	3
	Usage Time Gap (UTG)	1

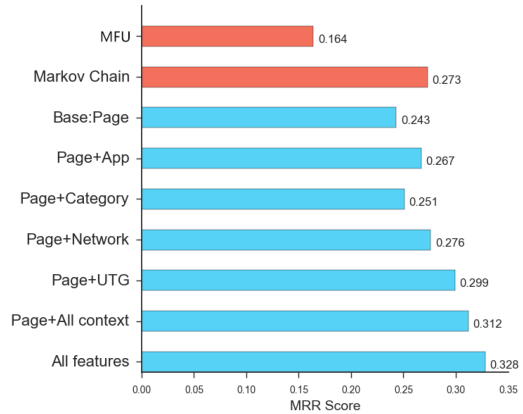


Fig. 10. Prediction results with different algorithms and features.

the third place of our recommendation list. This is because the classification algorithm can effectively integrate the task-specific features of pages and contextual features for prediction, which have been proved to be very important according to the exploratory analysis in the previous section. Second, we try various combinations of features for making prediction. We treat the page name as the basic feature. Adding the name of an app or the category of an app improves the prediction accuracy very marginally. The underlying reason may be that all the three features are about the functionalities of the pages, which are strongly correlated, and the page name can well represent the task-specific properties of the pages. Adding the contextual features can significantly improve the result, indicating that the contexts are indeed very vital factors to affect the inter-app navigation. As a result, by combining all the features, the best-of-breed prediction result is then obtained.

5.1.4 Exploring Situational Applications. The preceding proof-of-concept demonstration indicates that the inter-app navigation could be predicted to some extent, and we can further explore some potential situational applications for different stakeholders, including app developers, OS vendors, and end-users.

Exploring Potential Partner Apps. Based on the information of inter-app navigation, app developers can more efficiently identify the *upstream partner* apps from which users can navigate to the current app, and *downstream partner* apps to which users can navigate from the current app. On one hand, app developers could expose "deep links" to upstream apps to attract more user visits from upstream apps. For example, our recently released work, namely, Aladdin [33], can integrate such "navigation" knowledge and automatically generate a deep link for a page. On the other hand, app developers can consider integrating deep links from downstream apps to precisely

introduce users for downstream apps. Similar to the hyperlinks of Web pages, deep links can introduce more "*page visits*" of an app. In particular, the in-app ads are the major revenue channels for app developers, hereby establishing the deep links can potentially increase the possibility of ads clicking.

Optimizing Transitional Pages. For the state-of-the-art system-wide smart assistant such as Aviate [11], our findings can provide insights to further improve user experiences. Aviate predicts the next app that is likely to be used by users and displays the shortcut of apps on the system home screen. However, the prediction is performed at the *app level*, thus users still have to manually locate the desired page by several tedious steps. Combined with our analysis, the prediction can be performed at a fine granularity with less "transitional" overhead but navigate users directly to the page that they are really interested in. However, we do not mean to enforcedly remove the transitional pages. Instead, we can ask developers to customize the transitional pages. End-users can also bookmark the navigation, and employ some system-level record-and-replay techniques (such as ReRAN [22]) to automate the visit of these transitional pages. In other words, manual efforts are mitigated without having to remove transitional pages. Another potential optimization is that the predicted page can be more accurate to contexts (e.g., the current network condition), and can be prefetched into RAM to enable fast launch.

Producing Dynamic Deep Links. Finally, one future plan of our study is to build a recommendation mechanism that can predict which app pages a user is likely to reach, and dynamically generate deep links for such desired pages. Thus, the user can automatically pass through a few transitional pages that are automatically visited by the recommendation assistant, and ignore the time-consuming navigation steps. Since recent research efforts such as uLink [10] have already proposed practical approaches to dynamically generating deep links for app pages, the key point of such a recommendation mechanism is to accurately predict which pages should be navigated. Our proof-of-concept prediction technique can tell which kind of informational pages the user is likely to visit. Based on the prediction, we can recommend the possible pages and generate a deep link for the page, enabling the user to choose the desired page and navigate directly to that page. For example, if we predict that a user is going to watch a movie, then the system can offer a recommended movie list for the user to select. Our results can be combined with the recent efforts on in-situ semantic analysis [18], by which the system can more precisely predict which app pages the user desires.

5.2 Supporting Flexible App-Page Installability

From our observation, it is interesting that only a small portion of pages are frequently accessed and account for a substantial proportion of user interaction time. Such an observation makes us rethink the current "complete" package-based delivery and release model of apps, i.e., users need to download the whole application package. It is a well-known fact that the volume of application packages keeps increasing along with app versions, while inevitably requiring more computation resources of smartphones and possibly leading to longer user latency and more energy consumption.

As a result, we argue that it would be more desirable for an app to be downloaded and installed at the page level rather than the whole package. By this, we mean that users do not have to install all pages, but access the needed pages and download their corresponding resources in an "on-the-fly" fashion. Such a desire is reasonable and lessons can be learnt from many successful Software-as-a-Service applications such as Salesforce and Google Docs, where users focus on only the features that they really need and do not have to install the whole body of the applications including code, libraries, resources, and so on. In terms of inter-app navigation, such an "installation-free" style

can be achieved by clicking through a hyperlink to access a desired page of an app rather than downloading the whole app.

To achieve the goal of "page-level" installability of apps, current development, deployment, and release models of apps need to be significantly rethought. Here, users still need to download the installation packages of apps. However, apps need to be decomposed into several "loosely coupled" and less interrelated sets of pages that can be customizable by developers and users for on-demand use. Given that current apps are developed by mainstream Object-Oriented programming languages such as C++, Object-C, and Java, traditional programming principles should be more carefully considered by developers, such as enhancing the cohesion of a module, reducing the frequency of cross-module method call, and so on. However, developers may complain that such concerns may increase their manual efforts. It shall be more practical to develop efficient tools to facilitate developers to decompose their apps that are still compliant to existing programming styles. Advanced static analysis techniques are required to ensure the correctness and consistency of the decomposed apps. For example, the recent work such as PRIMO [36] and Aladdin [33] are moving toward the comprehensive analytics of inter-component communication (ICC) to support consistent decoupling of activities. Developers can apply these tools and techniques to identify the dependencies among the pages, and choose those "mandatory" pages that must be contained and/or the informational pages that are very frequently visited by users, in the released installation package. Those "optional" pages can be deployed as a service on the application servers running in a cloud and can release a set of "deep links" through which they can be accessed and loaded when users need them. Certainly, the downloaded application packages should include the reference (such as deep links) for these optional pages. When a user requests such pages, the app can simply download the code and resources and execute locally, or make the page executed and rendered at the cloud and merge the returned results with local environment. Additionally, such a deployment model should not compromise user experiences, and efficient system supports, such as API virtualization [25] and AppStreaming [1], can be leveraged.

One potential problem of page-level app installability is that under poor or even unavailable network condition, users may fail to access an app page that has not been downloaded, severely influencing the user experience. However, given today's pervasive deployment of cellular network and Wi-Fi, such an issue shall not be serious in practice. In addition, we can predict the pages that a user may visit based on his/her page-visiting history and pre-fetch these pages when the device is connected with Wi-Fi.

6 DISCUSSION

So far, we have made the empirical study to derive the knowledge of inter-app navigation, and some surprising findings are found. As an empirical study, some potential limitations may be threats to narrow the generalizability of our findings. In this section, we analyze the potential threats and discuss how to mitigate the limitations.

6.1 Impact of User Scale

Our results come from the field study that was conducted on 64 student volunteers' smartphones for three months, with 0.89 million records of user behaviors. Indeed, such a selected user group cannot be comprehensive enough to represent all smartphone users, i.e., reflecting the preferences of only on-campus students in China. In this way, the empirical results and derived knowledge may not well generalize to users from other groups. However, the process of our analysis approach and prediction techniques can still be generalized. On one hand, we use the distribution of dwell time for every single page to determine whether it is an informational or transitional page, since we assume that users would spend more time in informational pages than in transitional ones. Although

the absolute dwell time of the same page can vary among different user groups, e.g., elder people may spend more time in a page compared with the college students in our study, and the relative difference of dwell time between informational pages and transitional pages still remain the same. Therefore, our clustering approach to distinguishing informational and transitional pages can be generalized for different user groups. On the other hand, our proof-of-concept approach to predicting page-level navigation does not rely on features that are correlated with special user groups. Therefore, the proposed prediction technique is also generalizable.

6.2 Scalability of Data Collection Tool

Our data collection tool can introduce minor additional overhead when running at the background. Indeed, we should avoid compromising system performance along with user experiences when planning to deploy our tool at scale. Based on our previous experiences of large-scale user study collaborated with leading app store operators [28] and input method apps [32], we plan to optimize and integrate our data collection tool in these apps, so that we can learn more comprehensive knowledge of inter-app navigation. In addition, with the access to a large number of user profiles, we can add features of user modeling to enhance the analysis.

6.3 Granularity of User Interaction

Another limitation of our empirical study is that our data collection is focused at the page (activity) level. However, modern Android apps make use of fragment, which is a specific part of user interfaces (e.g., tabs) in an activity and can be displayed in the way similar to dynamically produced sub-pages by AJAX. Ideally, it is more comprehensive to measure the session time that the user spends on each fragment and on each activity, respectively. To address such a problem, we plan to employ some dynamic analysis based on user-interaction traces.

6.4 Value of Transitional Pages

Our findings evidence the potential overhead caused by those transitional pages. Indeed, from the user's perspective, such overhead could compromise user experiences and shall be avoided. These navigation pages can be technically removed by developers if they do not contain the functional features. To this end, some static analysis techniques can be applied to make the removal correct.

However, **the results do not consequently mean that the transitional pages are not meaningful**. For example, the Yelp's transitional pages let users choose between categories of Restaurants, Bars, Food, Reservations, and Order Delivery, all of which support different kinds of information needs and tasks. From the developer's perspective, some of the transitional pages, such as those that contain some in-app ads, cannot be roughly removed as they could contribute to increase developers' revenues. Hence, removing all transitional pages is not feasible. In contrast, we need to carefully justify whether a transitional page should be contained or not on the navigation path, e.g., allowing the developers to configure when releasing deep links, or the end users to decide by their own preferences. As we discussed previously, supporting flexible and customizable page installation could be a potential solution.

6.5 Comparison between App Navigation and Web Navigation

As compared in Section 2.2, there are some analogies between app pages and Web pages. So there exist similarities between app and Web navigation. For example, both app navigation and Web navigation involve several transitional pages between informational pages. However, the biggest difference between app and Web pages is that each Web page has a unique URL with which the Web page can be directly accessed. Therefore, the most common case of Web navigation originates from search engines (such as Google, Bing, and Baidu) to other informational Web pages. Such a

case does not exist in the app navigation, since there is no central search engines that index the app pages. Instead, the most common case of app navigation is from social network apps to other apps, indicating that the social network apps have become the dominant entrance on mobile devices.

7 RELATED WORK

In this section, we discuss the related existing literature studies and compare with our work.

• **Field Study of Smartphone Usage.** There have been some field studies to investigate user behaviors on smartphones. Ravindranath et al. [39] developed the AppInsight to automatically identify and characterize the critical paths in user transactions, and they conducted a field trial with 30 users for over 4 months to study the app performance. Rahmati et al. [41] designed LiveLab, a methodology to measure real-world smartphone usage and wireless networks with a reprogrammable in-device logger designed for long-term user studies. They conducted a user experiment on iPhone and analyzed how users use the network on their smartphones. In their following work [38], they conducted a study involving 34 iPhone 3GS users, reporting how users with different economic background use smartphones differently. Mathur et al. [34] carried out experiments with 10 users to model user engagement on mobile devices and tested their model with smartphone usage logs from 130 users. Desolda et al. [17] conducted an elicitation study on 65 users to understand how they interact with different apps on their own mobile devices for collaborative sensemaking, and further proposed spatially-aware cross-device techniques to assist users in navigating through mobile apps and websites using one or more devices. Ferreira et al. [19] collected smartphone application usage patterns from 21 participants and participants' context to study how they manage their time interacting with the devices. To this end, they proposed a sampling technique, to understand when a micro usage can occur. Similarly, our study is based on the controllable set of volunteers to understand the interaction time over apps. However, unlike preceding studies [19, 34] that focus on the app level, our work is made at the page level. Such a finer granularity enables us to predict the exact page to access and provide the shortcut for the navigation, which is totally not covered by existing efforts.

• **Mining Navigation Patterns.** A lot of research efforts have been made on mining the navigation patterns on the Web. Some existing literatures leveraged graph models. Borges et al. [13] proposed an N-gram model to exploit user navigation patterns and they used the entropy as an estimator of the user sessions' statistical property. Anderson et al. [9] proposed a Relational Markov Model (RMM) to model the behavior of Web users for personalizing websites. Liu et al. [31] proposed a method of computing page importance by user browsing graph rather than the traditional way of analyzing link graph. Chierichetti et al. [16] studied the extent to which the Markovian assumption is invalid for Web users. Other literatures use sequential pattern mining to exploit association rules. Fu et al. [20] designed a system that actively monitors and tracks a user's navigation, and applied A-priori algorithm to discover hidden patterns. Wang et al. [45] divided navigation sessions into frames based on a specific time internal, and proposed a personalized recommendation method by integrating user clustering and association-mining techniques. West et al. [47] studied how Web users navigate among Wikipedia with hyperlinks. Our work is inspired by these previous efforts on the Web, and we focus on how users navigate among apps. Similar to our work, Srinivasan et al. [42] developed a service called MobileMiner that runs on the phone to collect user usage information. They conducted a user experiment and use a sequential mining algorithm to find user behavior patterns. Jones et al. [24] presented a revisitation analysis of smartphone use to investigate whether smartphones induce usage habits. They distinguished the pattern granularity into macro and micro level, and found unique usage characteristics on micro level. Both of them

explore app-level usage patterns, but our work focuses on page-level navigation patterns rather than the app level.

• **Predicting App Usage.** Predicting the apps to be used not only facilitates mobile users to target the following apps but also can be leveraged by mobile systems to improve the performance. Abhinav et al. [37] proposed a method similar to a text compression algorithm that regards the usage history as sequential patterns and uses the preceding usage sequence to compute the conditional probability distribution for the next app. Yan et al. [48] proposed an algorithm for predicting next app to be used based on user contexts such as location and temporal access patterns. They built an app FALCON that can pop the predicted app to the home screen for fast launching. Richardo et al. [11] proposed Parallel Tree Augmented Naive Bayesian Network (PTAN) as the prediction model, and used a large scale of data from Aviate log dataset to train their model, and achieve high precision results. Wang et al. [46] correlated the apps being used with the user's location to accurately predict users' future locations and app usage. Chen et al. [15] used the approach of heterogeneous graph embedding to performing context-aware app usage prediction, taking location, time and app with type information into account. Zhao et al. [49] borrowed the idea of Doc2Vec and proposed AppUsage2Vec to model app usage records for prediction by considering the contribution of different apps, user personalized characteristics, and temporal context.

Our work differs from previous efforts much in several aspects. On one hand, the granularity of our research is at the *page* level, which is more fine-grained than the app level. As a result, we cannot exploit the existing dataset that mainly records the user behaviors at the app level. On the other hand, we reveal that the inter-app navigation has many transitional pages, which is a tremendous noise for the purpose of predicting informational pages. Thus, we propose a clustering approach to filtering out the transitional pages and enable more accurate prediction.

8 CONCLUSION

In this article, we have presented an empirical study of the "castle tunnel" on Android, i.e., the inter-app navigation behavior. The analysis was based on the behavioral data collected from a real user study, which contains nearly a million records of page visits. We found that transitional pages visited between two inter-app informational pages can account for 28.2% of the time in the entire inter-app navigation, a cost that can be significantly reduced through building direct links between the informational pages. An in-depth analysis reveals clear clustering patterns of inter-app pages, based on which, we demonstrate the possibility of predicting the next informational page a user shall visit. Our results provide actionable insights to end-users, app developers, and OS vendors.

While we have demonstrated the feasibility of predicting the next page using a standard algorithm, it is not our intent to optimize the prediction performance in this study. It is a meaningful future direction to build advanced prediction models, especially to consider the previous navigation sequences in the same session and to personalize the prediction. It is also intriguing to enlarge the scale of the user study to cover users with diverse demographics.

REFERENCES

[1] Marketing Land. 2015. Google App Streaming: A Big Move In Building "The Web Of Apps." Retrieved from http://marketingland.com/google-app-streaming-web-of-apps-152449.
[2] Android. 2019. Android Guide. Retrieved from http://developer.android.com/guide/components/index.html.
[3] Microsoft. 2019. Bing App Linking. Retrieved from https://msdn.microsoft.com/en-us/library/dn614167.
[4] Facebook. 2019. Facebook App Links. Retrieved from https://developers.facebook.com/docs/applinks.
[5] Google. 2019. Google App Indexing. Retrieved from https://developers.google.com/app-indexing/.
[6] Wikipedia. 2019. Mean reciprocal rank. Retrieved from https://en.wikipedia.org/wiki/Mean_reciprocal_rank.

[7] Comscore. 2019. Mobile Internet Usage Skyrockets in Past 4 Years. Retrieved from http://www.comscore.com/Insights/Blog/Mobile-Internet-Usage-Skyrockets-in-Past-4-Years-to-Overtake-Desktop-as-Most-Used-Digital-Platform.

[8] Wikipedia. 2019. Mobile deep linking. Retrieved from https://en.wikipedia.org/wiki/Mobile_deep_linking.

[9] Corin R. Anderson, Pedro M. Domingos, and Daniel S. Weld. 2002. Relational Markov models and their application to adaptive web navigation. In *Proceedings of the 8th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (SIGKDD'02)*. 143–152.

[10] Tanzirul Azim, Oriana Riva, and Suman Nath. 2016. uLink: Enabling user-defined deep linking to app content. In *Proceedings of the 14th Annual International Conference on Mobile Systems, Applications, and Services (MobiSys'16)*. 305–318.

[11] Ricardo Baeza-Yates, Di Jiang, Fabrizio Silvestri, and Beverly Harrison. 2015. Predicting the next app that you are going to use. In *Proceedings of the 8th ACM International Conference on Web Search and Data Mining (WSDM'15)*. 285–294.

[12] Ravi Bhoraskar, Seungyeop Han, Jinseong Jeon, Tanzirul Azim, Shuo Chen, Jaeyeon Jung, Suman Nath, Rui Wang, and David Wetherall. 2014. Brahmastra: Driving apps to test the security of third-party components. In *Proceedings of the 23rd USENIX Security Symposium*. 1021–1036.

[13] José Borges and Mark Levene. 1999. Data mining of user navigation patterns. In *Proceedings of International Workshop on Web Usage Analysis and User Profiling (WEBKDD'99)*. 92–111.

[14] Andrei Broder. 2002. A taxonomy of web search. In *ACM SIGIR Forum*, Vol. 36. 3–10.

[15] Xinlei Chen, Yu Wang, Jiayou He, Shijia Pan, Yong Li, and Pei Zhang. 2019. CAP: Context-aware app usage prediction with heterogeneous graph embedding. *Interact. Mobile Wear. Ubiq. Technol.* 3, 1 (2019), 4:1–4:25.

[16] Flavio Chierichetti, Ravi Kumar, Prabhakar Raghavan, and Tamás Sarlós. 2012. Are web users really Markovian? In *Proceedings of the 21st World Wide Web Conference (WWW'12)*. 609–618.

[17] Giuseppe Desolda, Carmelo Ardito, Hans-Christian Jetter, and Rosa Lanzilotti. 2019. Exploring spatially-aware cross-device interaction techniques for mobile collaborative sensemaking. *Int. J. Hum.-Comput. Studies* 122 (2019), 1–20.

[18] Earlence Fernandes, Oriana Riva, and Suman Nath. 2016. Appstract: On-the-fly app content semantics with better privacy. In *Proceedings of the 22nd Annual International Conference on Mobile Computing and Networking (MobiCom'16)*. 361–374.

[19] Denzil Ferreira, Jorge Goncalves, Vassilis Kostakos, Louise Barkhuus, and Anind K. Dey. 2014. Contextual experience sampling of mobile application micro-usage. In *Proceedings of the 16th International Conference on Human-Computer Interaction with Mobile Devices & Services (MobileHCI'14)*. 91–100.

[20] Xiaobin Fu, Jay Budzik, and Kristian J. Hammond. 2000. Mining navigation history for recommendation. In *Proceedings of the 5th International Conference on Intelligent User Interfaces (IUI'00)*. 106–112.

[21] B. Fuglede and F. Topsoe. 2004. Jensen-Shannon divergence and Hilbert space embedding. In *Proceedings of the IEEE International Symposium on Information Theory (ISIT'04)*.

[22] Lorenzo Gomez, Iulian Neamtiu, Tanzirul Azim, and Todd D. Millstein. 2013. RERAN: Timing- and touch-sensitive record and replay for Android. In *Proceedings of the 35th International Conference on Software Engineering (ICSE'13)*. 72–81.

[23] Aaron Halfaker, Oliver Keyes, Daniel Kluver, Jacob Thebault-Spieker, Tien T. Nguyen, Kenneth Shores, Anuradha Uduwage, and Morten Warncke-Wang. 2015. User session identification based on strong regularities in inter-activity time. In *Proceedings of the 24th International Conference on World Wide Web (WWW'15)*. 410–418.

[24] Simon L. Jones, Denzil Ferreira, Simo Hosio, Jorge Goncalves, and Vassilis Kostakos. 2015. Revisitation analysis of smartphone app use. In *Proceedings of the ACM International Joint Conference on Pervasive and Ubiquitous Computing (UbiComp'15)*. 1197–1208.

[25] Taeyeon Ki, Alexander Simeonov, Bhavika Pravin Jain, Chang Min Park, Keshav Sharma, Karthik Dantu, Steven Y. Ko, and Lukasz Ziarek. 2017. Reptor: Enabling API virtualization on Android for platform openness. In *Proceedings of the 15th Annual International Conference on Mobile Systems, Applications, and Services (MobiSys'17)*. 399–412.

[26] Stephen G. Kobourov. 2012. Spring embedders and force directed graph drawing algorithms. *CoRR abs/1201.3011* (2012). *arXiv:1201.3011*. http://arxiv.org/abs/1201.3011.

[27] S. Kullback. 1987. The Kullback-Leibler distance. *Amer. Stat.* 41, 4 (1987), 340–341.

[28] Huoran Li, Wei Ai, Xuanzhe Liu, Jian Tang, Feng Feng, Gang Huang, and Qiaozhu Mei. 2016. Voting with their feet: Inferring user preferences from app management activities. In *Proceedings of the 25th International Conference on World Wide Web (WWW'16)*. 1351–1361.

[29] Xiaoli Li and Bing Liu. 2003. Learning to classify texts using positive and unlabeled data. In *Proceedings of the 18th International Joint Conference on Artificial Intelligence (IJCAI'03)*. 587–594.

[30] Bing Liu, Yang Dai, X. Li, Wee Sun Lee, and P. S. Yu. 2003. Building text classifiers using positive and unlabeled examples. In *Proceedings of the 3rd IEEE International Conference on Data Mining (ICDM'03)*. 179–188.

[31] Yuting Liu, Bin Gao, Tie-Yan Liu, Ying Zhang, Zhiming Ma, Shuyuan He, and Hang Li. 2008. BrowseRank: Letting web users vote for page importance. In *Proceedings of the 31st Annual International ACM SIGIR Conference on Research and Development in Information Retrieval (SIGIR'08)*. 451–458.

[32] Xuan Lu, Wei Ai, Xuanzhe Liu, Qian Li, Ning Wang, Gang Huang, and Qiaozhu Mei. 2016. Learning from the ubiquitous language: An empirical analysis of emoji usage of smartphone users. In *Proceedings of the ACM International Joint Conference on Pervasive and Ubiquitous Computing (UbiComp'16)*. 770–780.

[33] Yun Ma, Ziniu Hu, Yunxin Liu, Tao Xie, and Xuanzhe Liu. 2018. Aladdin: Automating release of deep-link APIs on Android. In *Proceedings of the World Wide Web Conference (WWW'18)*. 1469–1478.

[34] Akhil Mathur, Nicholas D. Lane, and Fahim Kawsar. 2016. Engagement-aware computing: Modelling user engagement from mobile contexts. In *Proceedings of the ACM International Joint Conference on Pervasive and Ubiquitous Computing (UbiComp'16)*. 622–633.

[35] Andrew Y. Ng, Michael I. Jordan, and Yair Weiss. 2001. On spectral clustering: Analysis and an algorithm. *Proceedings of Conference on Neural Information Processing Systems (NIPS'01)* 14 (2001), 849–856.

[36] Damien Octeau, Somesh Jha, Matthew Dering, Patrick D. McDaniel, Alexandre Bartel, Li Li, Jacques Klein, and Yves Le Traon. 2016. Combining static analysis with probabilistic models to enable market-scale Android inter-component analysis. In *Proceedings of the 43rd Annual ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages (POPL'16)*. 469–484.

[37] Abhinav Parate, Matthias Hmer, David Chu, Deepak Ganesan, and Benjamin M. Marlin. 2013. Practical prediction and prefetch for faster access to applications on mobile phones. In *Proceedings of the ACM International Joint Conference on Pervasive and Ubiquitous Computing (UbiComp'13)*. 275–284.

[38] Ahmad Rahmati, Chad Tossell, Clayton Shepard, Philip Kortum, and Lin Zhong. 2012. Exploring iPhone usage: The influence of socioeconomic differences on smartphone adoption, usage and usability. In *Proceedings of the 14th International Conference on Human-Computer Interaction with Mobile Devices and Services (MobileHCI'12)*. 11–20.

[39] Lenin Ravindranath, Jitendra Padhye, Sharad Agarwal, Ratul Mahajan, Ian Obermiller, and Shahin Shayandeh. 2012. AppInsight: Mobile app performance monitoring in the wild. In *Proceedings of the 10th USENIX Symposium on Operating Systems Design and Implementation (OSDI'12)*. 107–120.

[40] Peter J. Rousseeuw. 1987. Silhouettes: A graphical aid to the interpretation and validation of cluster analysis. *J. Comput. Appl. Math.* 20, 20 (1987), 53–65.

[41] Clayton Shepard, Ahmad Rahmati, Chad Tossell, Lin Zhong, and Phillip Kortum. 2010. Livelab: Measuring wireless networks and smartphone users in the field. *ACM Sigmetrics Perform. Eval. Rev.* 38, 3 (2010), 15–20.

[42] Vijay Srinivasan, Saeed Moghaddam, Abhishek Mukherji, Kiran K. Rachuri, Chenren Xu, and Emmanuel Munguia Tapia. 2014. MobileMiner: Mining your frequent patterns on your phone. In *Proceedings of the ACM Conference on Ubiquitous Computing (UbiComp'14)*. 389–400.

[43] D. M. J. Tax. 2001. *One-Class Classification*. Doctoral Thesis. Delft Technical University.

[44] Niels Van Berkel, Chu Luo, Theodoros Anagnostopoulos, Denzil Ferreira, Jorge Goncalves, Simo Hosio, and Vassilis Kostakos. 2016. A systematic assessment of smartphone usage gaps. In *Proceedings of the Conference on Human Factors in Computing Systems (CHI'16)*. 4711–4721.

[45] Feng Hsu Wang and Hsiu Mei Shao. 2004. Effective personalized recommendation based on time-framed navigation clustering and association mining. *Expert Syst. Appl.* 27, 3 (2004), 365–377.

[46] Huandong Wang, Yong Li, Sihan Zeng, Gang Wang, Pengyu Zhang, Pan Hui, and Depeng Jin. 2019. Modeling spatio-temporal app usage for a large user population. *Interact. Mobile Wear. Ubiq. Technol.* 3, 1 (2019), 27:1–27:23.

[47] Robert West and Jure Leskovec. 2012. Human wayfinding in information networks. In *Proceedings of the 21st World Wide Web Conference (WWW'12)*. 619–628.

[48] Tingxin Yan, David Chu, Deepak Ganesan, Aman Kansal, and Jie Liu. 2012. Fast app launching for mobile devices using predictive user context. In *Proceedings of the 10th International Conference on Mobile Systems, Applications, and Services (MobiSys'12)*. 113–126.

[49] Sha Zhao, Zhiling Luo, Ziwen Jiang, Haiyan Wang, Feng Xu, Shijian Li, Jianwei Yin, and Gang Pan. 2019. AppUsage2Vec: Modeling smartphone app usage for prediction. In *Proceedings of the 35th IEEE International Conference on Data Engineering (ICDE'19)*. 1322–1333.

Received September 2018; revised January 2020; accepted April 2020