AppRush: Using Dynamic Shortcuts to Facilitate Application Launching on Mobile Devices

Chen Suna, Jun Zhenga,∗, Huiping Y aoa, Yang Wanga, D. Frank Hsub

aDepartment of Computer Science and Engineering, New Mexico Institute of Mining and Technology, Socorro, NM 87801, USA
bDepartment of Computer and Information Science, Fordham University, New York, NY 10023, USA

Abstract

When there are many applications (apps) installed on a mobile device, the simple task of launching an app could become inconvenient, as the user may need to swipe the screen several times to find the desired app. In this paper, we present AppRush, an adaptive user interface solution for mobile devices, which uses dynamic shortcuts to facilitate app launching. AppRush predicts the apps most likely to be launched at a certain time based on the user’s app usage history. The prediction model utilizes multiple features including recency, frequency, duration, time distribution and app launching sequence. AppRush updates the app shortcuts dynamically on the home screen according to the prediction result. With the correct prediction, the user is able to launch an app from the home screen via one-tap. The evaluation performed on real-world user data demonstrates the effectiveness of the proposed prediction algorithm. In addition, AppRush maintains a low system overhead, which makes it applicable for running on battery-driven mobile devices.

Keywords: Adaptive user interface, dynamic shortcuts, mobile app usage, personalization

1. Introduction

The use of smartphone has seen tremendous growth in recent years. One of the most important features of the smartphone is that users can install and use a large number of various applications (apps). According to a study done in April 2011, an Android phone user has average 35 apps installed on his/her device [1]. Another study showed that in 2011, for the first time, people spent more time on mobile apps than desktop and mobile web browsing [2]. It is obvious that the popularity of mobile apps is still on the rise. In September 2012, Google Play (previously known as Android market) hit 25 billion app downloads, and there were more than 675,000 apps available at that time [3].

From the users’ viewpoint, it is good that they have a huge number of choices on mobile apps, and can install plenty of them with the ample storage capacity of today’s smartphone. To help save the user’s effort and time to find and launch an app, Android system provides a similar concept of shortcut as in Windows...
system, or symbolic link as in Unix-like system. The user can add shortcuts of favorite apps on the home screen to quickly launch them without looking into the list of all apps. This traditional shortcut solution grants a user the ability to organize certain apps for fast launching according to personal preference, but it suffers the size problem of the smartphone screen. The small screen of smartphone can only show a limited number of app shortcuts at a time (for most Android phones available nowadays, at most 16 or 20 shortcuts can be displayed on one screen) that only those apps with shortcuts put on the initial home screen can be launched via one-tap. In most cases, the user may need to swipe the screen multiple times or search the long list of installed apps to find the desired app. In addition, some users may tend to keep a tidy home screen rather than cramming too many shortcuts on the home screen, while some others may prefer to reserve space for various widgets. Thus, there is a need for new solutions to help mobile users quickly launch an app.

In this paper, we present AppRush, an adaptive mobile user interface solution, which organizes shortcuts dynamically according to the user’s app usage history. Specifically, the AppRush system monitors the user’s behavior of app usage, and then predicts the apps most likely to be launched at a certain time. According to the predicted app list, the shortcuts on the home screen will be rearranged dynamically. Different from the traditional static shortcut solution, AppRush adapts to the user’s usage behavior and does not require the user’s effort to organize shortcuts. Ideally, AppRush can provide an unlimited number of app shortcuts on a small screen because the user can always launch an app via just one-tap on the home screen once it is in the predicted app list.

2. System Design and Implementation

We implemented AppRush as an independent Android app and published it in Google Play. Our system implementation tries to balance the prediction performance and power consumption as the app is designed to run on battery-driven mobile devices.

2.1. System Overview

The architecture of AppRush system is shown in Figure 1. The system consists of two major components: a frontend Android widget served as a container to hold dynamic shortcuts, and a background service monitoring app usage, making prediction based on usage history and updating the frontend shortcuts according to any change of the prediction.

2.2. Frontend Widget

Dynamic shortcuts are hosted in an app widget with which the user directly interacts, which is available since Android 1.5. This means that theoretically our solution can be ported to all current Android devices. On the home screen, a widget can display an application’s timely or relevant information at a glance which is a perfect fit to implement our solution, as we can programmably control the content shown in the widget.

In the current implementation, we provide two different sized widgets: a single-row widget holding 4 shortcuts and a double-row widget holding 8 shortcuts. By default, the widget has a color background to remind the user that it is a widget rather than a set of normal shortcuts. However, AppRush also gives the user the option to select a transparent background so that the user has a natural and smooth substitute for normal shortcuts. Figure 2 shows the appearance of AppRush widgets on the home screen with different size and background.

2.3. Background Service

The frontend widget is under the control of the background service consisting of three major subcomponents: App Usage Monitor, App Launch Predictor, and Widget Updater.

App Usage Monitor keeps track of when an app is launched and how long it has been used. Detailed time information including month, day of month, day of week, hour, minute, second and duration in millisecond
for each app launch event is kept along with the app’s identifier into a local database. The database keeps the records within the last six weeks as the history data of app usage, and the data serves as the input to App Launch Predictor. Besides monitoring app usage, we also keep track of app install/uninstall events so that the system will get notified when a previously installed app is not available any more to ensure that shortcuts to nonexistent apps will not be provided.

App Launch Predictor implements the prediction model described in Section 4 which uses app usage record to predict the apps mostly likely to be launched at a certain time. For the sake of power-efficiency, some parts of the feature computation work are executed immediately when certain triggering event happens, while other parts of the computation may be performed later at an appropriate time based on a lazy strategy.

Widget Updater reads the app ranking list at a certain time and updates dynamic shortcuts in the frontend widget. In order to save battery life, we try to eliminate unnecessary updates as much as we can. To this end, dynamic shortcuts only get updated in the following three situations: (1) The widget is being initialized, which could happen either when a widget is newly added to the home screen, or when the device boots up; (2) The screen turns on, which indicates that user is probably about to use certain apps; (3) A front-running app quits, which indicates that user will see the home screen and may launch other apps. Furthermore, we set the minimum interval between two updates as 5 minutes. This setting ensures that the frontend widget will not get updated too frequently within a short time period.

2.4. User Options

Besides giving users the option to choose a widget with certain size and background, we also implemented an option called Exclusion List to let a user select certain apps that are not going to be shown in AppRush widget. The primary reason of this design is avoiding duplicated shortcuts of a user’s favorite apps or system apps which already have static shortcunts on the home screen.

3. Data Collection and Observations

3.1. User Data Collection

The data used for building the prediction model of AppRush was collected from anonymous users who installed AppRush app from Google Play. The users were informed about our data collection behavior through a detailed privacy policy notice. Once a user accepted the policy and installed AppRush, the app usage log containing the detailed time information were submitted to our sever on a daily basis. The collected dataset contains the app usage history logs of 52 anonymous users from more than 20 different countries in a period of 49 days (7 weeks). The total number of different apps used by the users during the period varies
from 36 to 250 with a median of 87 and an average of 94.8. In terms of daily app usage, the median of average number of apps used per day and the median of average number of app launches happened per day are 9.9 and 36.0, respectively.

### 3.2. Observations on App Usage

The motivation of providing dynamic shortcuts to facilitate mobile app launching is based on our observations from the collected dataset that there are certain patterns in the use of mobile apps.

Our first observation is that the use of mobile apps is highly repetitive. The user tends to use certain apps over and over again. The repetitive behavior may occur in a short time period, or in a long run, or in both. For example, the user may launch a weather report app to check the weather once per day, and start an email client to receive or send emails several times a day. These recurrences are the most obvious patterns which can be easily captured with the recency and frequency features.

We have also observed that the average usage duration, i.e. the average amount of time an app runs in the foreground, may vary a lot among different apps. Figure 3 shows the average durations of a user’s top 10 most frequently used apps obtained from the collected dataset. The huge difference of interaction times among different apps can be clearly observed in the figure: the official YouTube app (com.google.android.youtube) takes an average of more than 2,000 seconds for each launch, while the average interaction time for Catch Notes (com.threebanana.notes) is a little above 200 seconds. This observation suggests that directly using the length of interaction time to measure the app’s usage score [4] will produce biased results. Therefore, the duration information should be considered in a different way.

Our third observation is that for some users there exists a pattern of when the user launches a certain app. Figure 4 shows the time distribution of app usage by a user from the collected dataset. The colors represent different ranges of the number of times the app being launched in certain hour of the day and day of the week. It can be seen from Figure 4(a) that the Facebook app was launched mainly in the morning and at night, but seldom in the afternoon. We can also observe that the user used Facebook app mostly at night except Friday. Obviously the midnight of Tuesday was the user’s favorite time for Facebook. Figure 4(b) shows the usage pattern of all apps for the same user. As shown in the figure, the user also actively used other apps in the afternoon, and the midnight of Saturday was the most active time.
4. Prediction Model

Based on our observations on users’ app usage patterns, we developed a prediction model for AppRush utilizing multiple features including recency, frequency, duration, time distribution and app launching sequence to rank the apps.

4.1. Recency and Frequency

Recency measures how much time has passed since the last launch of an app, while frequency measures how many times an app has been launched. These two features are the most commonly used ones for adaptive user interface tasks. Lee et al. [5] proposed a Combined Recency and Frequency (CRF) model which considers both recency and frequency. For ranking purpose, each item (in our case, the item is an app) is associated with a CRF value. The CRF value of an app \( k \) at a given time \( t \) is calculated as

\[
    crf_{kt} = \sum_{i=1}^{n_k} \left( \frac{1}{p} \right) \lambda (t-t_i)
\]

where \( n_k \) is the total number of launches of app \( k \), \( t_i \) is the time when the \( i \)th launch happened. \( p \) and \( \lambda \) are parameters used for weighting each launch. The effect is that the CRF value is reduced to \( \frac{1}{p} \) of the original value after every \( \frac{1}{\lambda} \) time step.

By doing some transformations on Equation (1), we can find that the computation of CRF value is incremental. This means that knowing the CRF value of an app \( k \) at time \( t_i \), we can easily compute its CRF value at time \( t_{i+1} \) without recalculating the contribution of all of the past launches, as shown in Equation (2)

\[
    crf_{k_{i+1}} = b_{k_{i+1}} + r \times crf_{k_i}
\]

where \( b_{k_{i+1}} \) is a base score used to credit every launch, \( b_{k_{i+1}} = 1 \) if there is a launch of app \( k \) at time \( t_{i+1} \), or \( b_{k_{i+1}} = 0 \) if app \( k \) is not launched at time \( t_{i+1} \), and \( r = 1/p^\lambda \) is a real number in (0, 1) used as a weighting factor to age the CRF value at an earlier time. A smaller \( r \) value weights less to past launches which gives the result closer to that of the Most Recently Used (MRU) method. On the contrary, a larger \( r \) value gives more weight to past launches so that the result is closer to that of the Most Frequently Used (MFU) method.

4.2. Duration

Duration measures how long an app is used for a launch. According to our observation that the average durations for different apps may vary a lot, any solution using a universal criteria to measure durations among all apps would lead to a biased result. We came up with a solution comparing the duration of an app use event only to the average duration of its past uses.

We incorporate the duration into the original CRF calculation in Equation (2) by replacing the base score \( b_{k_{i+1}} \) with a dynamic duration weighting score \( dw_{k_{i+1}} \). The new CRFD (Combined Recency, Frequency, and Duration) value is calculated as

\[
    crfd_{k_{i+1}} = dw_{k_{i+1}} + r \times crf_{k_i}
\]

For the duration weighting score, if there is no launch of app \( k \) at time \( t_{i+1} \), \( dw_{k_{i+1}} = 0 \). Otherwise, the \( dw_{k_{i+1}} \) value is determined according to the comparison between the duration of current use of app \( k \) and the average duration of its past uses. If there is a launch of app \( k \) at time \( t_{i+1} \), we define \( d_{k_{i+1}} \) as the duration of current use and \( \overline{d}_{k_i} \) as the average duration of its past uses. The duration weighting score \( dw_{k_{i+1}} \) is then calculated as

\[
    dw_{k_{i+1}} = \begin{cases} 
        1.2, & d_{k_{i+1}} \in \left[ 1.5 \times \overline{d}_{k_i}, \infty \right) \\
        1.0, & d_{k_{i+1}} \in \left( 0.5 \times \overline{d}_{k_i}, 1.5 \times \overline{d}_{k_i} \right) \\
        0.8, & d_{k_{i+1}} \in \left[ 0, 0.5 \times \overline{d}_{k_i} \right] \\
        0, & \text{App } k \text{ is not launched at time } t_{i+1} \end{cases}
\]

The basic idea behind Equation (4) is that we give a higher weight to the use whose duration is significantly longer than the average duration of past uses of the same app, while lower weight is given to the use whose duration is significantly shorter than its past average value.
4.3. Time Distribution

To capture the patterns of app usage shown in Figure 4, we use two time-related scores: the HOD (Hour of Day) score measuring the probability of an app being used at the current time of a day, and the DOW (Day of Week) score measuring the probability of an app being used on the same day of week as today. Denote the current day of week as \( d \) and the current hour of day as \( h \). The HOD and DOW scores of app \( k \) are calculated by Equations (5) and (6), respectively. The values of both HOD and DOW scores are in the range of \([0, 1] \).

\[
\text{hod}_{kh} = \frac{\text{Number of launches of } k \text{ in } [h-1, h+1]}{\text{Max number of launches of all apps}} \tag{5}
\]

\[
\text{dow}_{kd} = \frac{\text{Number of launches of } k \text{ on } d}{\text{Max number of launches of all apps}} \tag{6}
\]

4.4. App Launching Sequence

Our last feature is from the sequence of app launching referred as the Trigger and Follower in [6] or more formally a Markov chain in [7]. This feature describes the relationship that the launching of one app may be often followed by the launching of some other apps. For example, right after a user takes a picture via the camera app, often he will open the Facebook app to post the picture on his Facebook page, or sometimes he may send the picture to a friend using an email app. This relationship of two consecutive app launchings can be calculated with Bayesian probability

\[
P(K_{n+1} = k | K_n = k_n) = \frac{|k_n \rightarrow k|}{|k_n|} \tag{7}
\]

where \(|k_n \rightarrow k|\) denotes the number of launching of app \( k_n \) followed directly by launching of app \( k \), and \(|k_n|\) is the total number of launches of app \( k_n \). We use \( \text{seq}_k = P(K_{n+1} = k | K_n = k_n) \) to represent the SEQ (sequence) score of app \( k \) whose value is in the range of \([0, 1]\). So if the last used app is \( k_n \), the app \( k_i \) with a higher SEQ score is considered more likely to be launched next.

4.5. App Ranking

Finally, we combine all the aforementioned features to calculate the AppRush score for ranking. We first normalize the CRFD score of each app, so that CRFD, HOD, DOW, and SEQ scores are all in the closed unit interval \([0, 1]\). Next, we calculate the AppRush score using a weighted sum model as shown in Equation (8).

\[
apprush_k = w_c \times \text{crfd}_k + w_h \times \text{hod}_k + w_d \times \text{dow}_k + w_s \times \text{seq}_k \tag{8}
\]

where \( w_c, w_h, w_d, w_s \) are weights used to adjust the importance of each score. Each weight is determined dynamically from the corresponding feature’s cumulative prediction accuracy. At a specific time, we rank all the apps based on their AppRush scores from high to low and update the frontend widget with shortcuts of the top \( N \) apps.

5. Evaluation

In this section, we evaluate the prediction performance of AppRush based on the collected real-world user dataset. The system overhead of AppRush is also estimated.

5.1. Prediction Performance

To measure the accuracy of AppRush prediction algorithm, we use the hit rate as shown in Equation (9). A hit happens when a user is about to launch an app and the app is among the top \( N \) results in the prediction list. Otherwise it is a miss.

\[
\text{Prediction accuracy} = \frac{\# \text{ of Hits}}{\# \text{ of Hits} + \# \text{ of Misses}} \tag{9}
\]
Figure 5 shows the boxplots of average prediction accuracy for 52 users after the 49th day. We compare our method with MFU, MRU and CRF for different number of dynamic shortcuts ($N = \{4, 8\}$). The aging parameter $r$ is set to 0.85 for both 4 and 8 shortcuts which achieves best performance for CRF model. It can be seen that AppRush prediction algorithm outperforms all other methods.

![Boxplots of average prediction accuracy for 52 users](image)

**Fig. 5.** Average prediction accuracy of 52 users when using: (a) 4 shortcuts; (b) 8 shortcuts

5.2. **System Overhead**

For a system runs on battery-driven mobile devices, it is important to maintain a low system overhead. Because the computation cost of AppRush is affected by many factors such as the number of used apps, the frequency of app launching, the frequency of the screen being turned on, and the size of the widget etc., it is hard to accurately measure the system overhead of AppRush.

To obtain a general estimation of AppRush’s system overhead, we consecutively monitored the CPU usage as well as the memory usage of AppRush on a Samsung Galaxy Nexus phone with the latest Android 4.1 OS for 7 days. During the monitoring period, there were on average 11 apps used and 31.9 app launches happened per day. A third-party Android app named Battery Mix\(^2\) was used to watch the CPU usage information. Battery Mix gives the percentage of CPU usage taken by each process in the last 24 hours. The memory usage of AppRush was obtained through the Android system settings. Both the CPU usage and the memory usage of AppRush were recorded twice every day at 11:00 am and 11:00 pm. We found that the average CPU usage of AppRush was less than 1% and it used about 4-6 MB memory which can be considered as an acceptable overhead.

6. **Related Work**

Providing an adaptive user interface to facilitate reuse or revisiting activities can date back to the time of non-mobile computer systems. Greenberg and Witten [8] demonstrated an adaptive user interface for menu-driven application. Recently, Fitchett and Cockburn [7] proposed an algorithm using recency, frequency, temporal clustering, and time of day to predict revisitations and reuses in various contexts, such as file accesses, website visits, windows switches, and command lines. For mobile systems, Fukazawa et al. [9] built an automatic menu customization system on Windows Mobile phone using rankSVM (Support Vector Machine) to rank system functions based on user’s operation history. Vetek et al. [10] proposed SmartActions, an application on Nokia S60 smartphones which presents a list of the 5 most likely functions.

---

\(^2\)https://play.google.com/store/apps/details?id=jp.smapho.battery_mix
the user will use in the current context on the homescreen. Most recently, Shin et al. [11] proposed a dynamic home screen solution similar to us but employed a significantly different prediction algorithm based on Naïve Bayesian (NB) using much more features than us (e.g. average 19.3 and 13.2 features per user used for the user-NB and app-NB models, respectively). Some of the features are power consuming such as GPS-related ones. The performance improvement of their prediction algorithm over MFU is 8%.

Besides all the above works that aim to provide adaptive user interface based on user context, there were works that modeling and predicting user behavior on mobile devices for other purposes, such as recommendation system, battery management, and system optimization. Yan and Chen [4] applied a RFD (Recency, Frequency, and Duration) model to quantify the usage of mobile apps, and made personalized app recommendations for Android users. Ravi et al. [12] explored the context-aware battery management for mobile phones. Part of their work was to use the average duration of user’s phone calls in each hour of the day to predict the call time needs of the user. Yan et al. [6] built a system for app prelaunching on Windows Phone platform using context information including temporal bursts, locations and trigger/follower sequences.

7. Conclusion

AppRush is an adaptive user interface solution which uses dynamic shortcuts to facilitate app launching on battery powered mobile devices. The system can achieve a good prediction of the user’s app launching behavior with low system overhead. There are several enhancements planned in our future work. Currently, the performance evaluation doesn’t reflect the user’s interest in using dynamic shortcuts. To find out the user’s acceptance to our system, we need to count how many times the user launches apps via dynamic shortcuts and how many times through other ways. We will also investigate the possibility of incorporating other context features such as battery level and network status which are easy to obtain and low power consuming to achieve better prediction accuracy. Some other options to enhance the proposed dynamic shortcuts solution such as gesture based control will also be explored in future.

References