On Automatically Detecting Similar Android Apps

Mario Linares-Vásquez, Andrew Holtzhauer and Denys Poshyvanyk
The College of William and Mary, Williamsburg, VA, USA
{mlinarev, asholtzh, denys}@cs.wm.edu

***Abstract*—Detecting similar applications is a challenging problem, since it implies that similar high-level features and their low-level implementations can be detected and matched automatically. We propose an approach for automatically detecting Closely reLated applications in ANdroid (*CLANdroid*) by relying on advanced Information Retrieval techniques and five semantic anchors: identifiers, Android APIs, intents, permissions, and sensors. To evaluate *CLANdroid* we created a benchmark consisting of 14,450 apps along with information on similar apps provided by Google Play. We also compared effectiveness of different semantic anchors for detecting similar apps as perceived by 27 users. The results show that using Android-specific semantic anchors are useful for detecting similar Android apps across different categories. We also measured the impact of third-party libraries and obfuscated code when identifying similar Android apps, and our results suggest that there is significant difference in the accuracy when third-party libraries are excluded.**

I. Introduction

End-users and developers take advantage of search engines in online markets or repositories for browsing and searching software systems that are relevant to their needs. On one hand, end-users frequently search for applications that support their daily activities or substitute apps (i.e., less buggy app from another vendor). On the other hand, mobile app developers use code search engines and markets for opportunistic reuse [1]–[3], prototyping [4], market analysis [5]–[9], or finding similar applications to a system under development [10].

Identifying similarities between mobile apps plays an important role when understanding salient features in successful apps [11], for discovering plagiarism and clones across markets [10], [12]–[17], and for learning how to use APIs (in the case of new Android developers or new API releases). Moreover, the complexity of those tasks is increasing because of the staggering amount of apps that are available in markets and repositories (e.g., 1.5M+ Android apps available at Google Play). Therefore, searching for similar apps is becoming especially relevant due to the popularity of mobile devices and the distribution of mobile applications via online markets.

However, detecting similar applications is a notoriously difficult problem, since it means automatically detecting high-level requirements for these applications that match semantically [18, pages 74,80] [19]. This situation is aggravated by several major factors: (i) many application repositories are polluted with poorly functioning projects [20]; (ii) a match between words in requirement documents with words in the descriptions or in the source code of applications does not guarantee that these applications are relevant to the requirements; (iii) applications may be highly-similar to one another in terms of low-level implementations of some functions even if they do not perform the same high-level functionality [21], and (iv) market-specific search engines identify similar apps by relying on textual descriptions only.

Inspired by the previous work for detecting similar desktop applications [22], we devised an approach, named as *CLANdroid*, for detecting similar Android apps in free app markets or open source repositories. In addition to previously used semantic anchors, such as identifiers and API calls [22], *CLANdroid* relies on Android specific anchors: (i) explicit and implicit intents used in the apps, (ii) user permissions declared in the manifest files, and (iii) sensors used by the apps. Furthermore, we analyzed the impact of (not) using third-party libraries and obfuscated apps when detecting similar apps by relying on the guidelines from our previous work [3].

The results demonstrate that Android-specific semantic anchors are useful when detecting similar apps across different domain categories. In particular, even though previous studies suggest that Android apps are highly dependent on the Android SDK and in particular on a subset of APIs that must be always used [1], [2], [6], our results demonstrate that API calls in Android apps are still useful for identifying variabilities and similarities across different apps. Also, we found that third-party libraries and obfuscated code significantly impact the accuracy of detecting similar Android apps.

In summary, this paper makes the following contributions:

- An approach for detecting similar Android applications, *CLANdroid*, which is useful for developers and users when browsing and searching apps. *CLANdroid* is able to work even when source code is not available;
- A user study with 27 participants aimed at evaluating the capability of *CLANdroid* for detecting similar Android applications from Google Play;
- A large scale study on 14,450 Android apps to evaluate differences of *CLANdroid* and Google Play when detecting similar applications. Our study also analyzes the impact of third-party libraries and obfuscated code when detecting similar Android applications distributed as APK (Android PacKage) files;
- An online version of *CLANdroid* that can be used for detecting similar Android apps using different options (i.e., including third-party libraries and obfuscated apps, excluding third-party libraries or obfuscated apps), and different semantic anchors (i.e., identifiers, API calls, intents, sensors, and user permissions). *CLANdroid* is available at **http://www.semeru.info/clandroid**.

978-1-5090-1428-6/16/$31.00 ©2016 IEEE

II. Our approach for Finding Closely Related Android Applications

CLANdroid is based on the three key ideas behind *CLAN* [22]: (i) similar apps share some semantic anchors (e.g., API calls), (ii) requirements are implemented by a combination of different semantic anchors, and (iii) some semantic anchors are more expressive in terms of describing salient features (i.e., weighted contribution). While *CLAN* uses APIs as semantic anchors for desktop Java applications, Android apps rely on unique programming concepts such as intents, permissions, and sensors to implement underlying features. Therefore, *CLANDroid* extends the concept of semantic anchors to Android-specific elements that might be used to identify similar applications. For instance, if a set of applications use an intent such as `ACTION_DIAL` (which displays the phone dialer with a provided number filled in and is more common in all the Android apps), this intent would have a lower weight as compared to a perhaps rarer intent like `ACTION_CREATE_DOCUMENT` (which allows the user to create a document). Also, similarly to how combinations of API calls fulfill requirements as opposed to a single API call, combinations of intents are used to provide an Android application with all of its functionality, not just a single intent.

To implement the three key ideas mentioned before, we rely on an Information Retrieval (IR) technique called *Latent Semantic Indexing (LSI)* that reduces the dimensionality of the similarity space while simultaneously revealing latent concepts that are implemented in the underlying corpus of documents [23]. In LSI, terms are elevated to an abstract space, and terms that are used in similar contexts are considered similar even if they are spelled differently. LSI automatically makes embedded concepts explicit using *Singular Value Decomposition (SVD)*, which is a form of factor analysis used to reduce dimensionality of the space to capture most essential semantic information. Therefore, each semantic anchor and its frequencies are represented as a separate Term-Document-Matrix (TDM) at app level (i.e., semantic anchors are terms and app are the documents), and then LSI is applied on each matrix to extract latent concepts for each semantic anchor. After applying LSI on each TDM for each semantic anchor, a similarity index can be computed by measuring the distance (e.g., cosine distance) between two different apps for each semantic anchor (e.g., cosine similarities among apps can be computed based on API calls, intents, sensors, etc).

A. *The* CLANdroid *Architecture*

The *CLANdroid* architecture and workflow is depicted in Fig. 1. The mobile apps are downloaded from a marketplace in APK format or from open source repositories such as GitHub or Google Code ①. Then, the semantic anchors are extracted from the apps statically ②. Once all the semantic anchors are extracted, Term-Document Matrices (TDM) are generated for each type of semantic anchor ③: API-Application Matrix ($\texttt{TDM}_{\texttt{API}}$), Sensors-Application Matrix ($\texttt{TDM}_{\texttt{S}}$), Intents-Application Matrix ($\texttt{TDM}_{\texttt{I}}$), Permissions-Application Matrix

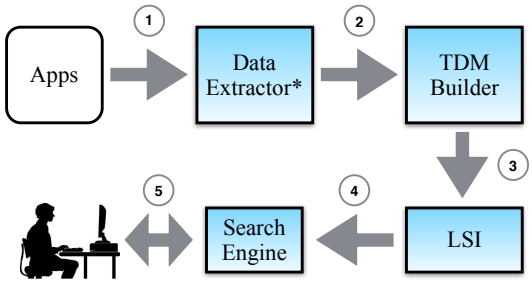

Fig. 1. *CLANdroid* architecture and workflow. *The details of the Data Extractor are depicted in Figure 2.

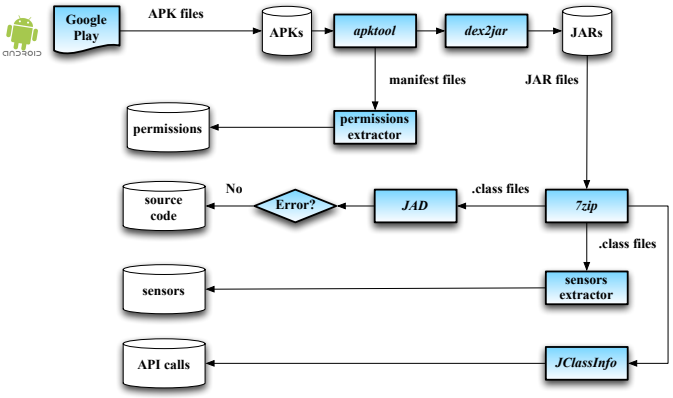

Fig. 2. Data extraction process.

($\texttt{TDM}_{\texttt{P}}$), and Identifiers-Application Matrix ($\texttt{TDM}_{\texttt{ID}}$). *CLANdroid* applies the LSI algorithm to each of these five TDM matrices and computes the cosine similarity between each pair of apps in a similar manner to *CLAN* ④. Finally, a practitioner can select a target app and semantic anchor (e.g., sensors) in the search engine ⑤, and *CLANdroid* retrieves the most similar apps according to the similarity index.

B. *Extracting Semantic Anchors From Android Apps*

CLANdroid downloads the APKs directly from Google Play and then decompiles the APKs to extract source code identifiers, sensors, API calls, and intents, and Android permissions with the infrastructure depicted in Fig. 2. The general process is as the following: (i) the APK files are unziped by using the *apktool*[1] tool, which reveals the compiled Android app code file (note that an APK is just a set of zipped DEX files); then (ii) the DEX files are translated to JAR files using the *dex2jar*[2] tool; then (iii) *CLANdroid* extracts the `.class` files from the JAR files by using the *7zip*[3] tool; and finally (iv) *CLANdroid* decompiles the `.class` files to `.java` files by using the *JAD*[4] decompiler tool (Fig. 2). Once each APK is decompiled into JAR files and source code, *CLANdroid* extracts semantic anchors from different artifacts: identifiers and intents from source code, APIs and sensors from JAR files, and permissions from the `AndroidManifest.xml` files.

CLANdroid is able to operate even in situations where the source code is not available; in that case the APK files

[1]http://code.google.com/p/android-apktool/
[2]http://code.google.com/p/dex2jar/
[3]http://www.7-zip.org/
[4]http://www.varaneckas.com/jad/

need to be decompiled to extract the different semantic anchors. Once each app is decompiled into source code, we ran scripts to extract various data: identifiers, APIs, sensors, and intents from the source code, and permissions from the `AndroidManifest.xml` files. All the details of the data acquisition and extraction are outlined in Section III.

C. Search Engine

Once all the metadata is extracted, the TDM matrices are generated for each type of semantic anchor. Therefore, the *Term-Document Matrix (TDM) Builder* takes app metadata as its input, and it uses this metadata to produce five TDMs: Class-Application Matrix ($\texttt{TDM}_\texttt{C}$), Sensors-Application Matrix ($\texttt{TDM}_\texttt{S}$), Intents-Application Matrix ($\texttt{TDM}_\texttt{I}$), Permissions-Application Matrix ($\texttt{TDM}_\texttt{P}$), and Identifiers-Application Matrix ($\texttt{TDM}_\texttt{ID}$). We applied the LSI algorithm to each of these five TDM matrices and computed the cosine similarity between each application in an identical manner to *CLAN*. The final result is a set of five similarity matrices LSI_x (one for each semantic anchor x). A *Similarity Matrix*, S_x is a square matrix whose rows and columns designate applications. For any two applications A_i and A_j, each element of S_x, $S_x[i,j]$ is the similarity score between these applications when using x as the semantic anchor that is defined as follows:

$$S_x[i,j] = \begin{cases} cosine(LSI_x(A_i), LSI_x(A_j)) & \text{if} i \neq j \\ 1 & \text{if } i = j \end{cases}$$

When the user enters a query (i.e., target app and semantic anchor) it is passed to the *Search Engine* that retrieves relevant applications with ranks in the descending order using S_x.

III. Empirical Study: Finding Closely Related Android Applications

We conducted an empirical study to determine how effective *CLANDroid* is for finding similar Android apps. This study was driven by the following *goals*: (i) to evaluate which semantic anchors are more effective for detecting similar apps; and (ii) to analyze the impact of third-party libraries and obfuscated code when detecting similar apps, since these two factors have been shown to have significant impact on reuse in Android apps and experiments using APKs [3]. The *context* of the study is comprised of 14,450 free Android apps that were downloaded from Google Play. The *perspective* is of researchers interested in designing an approach for detecting similar Android apps. The *quality focus* is the goldset of similar apps provided by Google Play[5] and the similarity of the apps as perceived by users.

A. Research Questions

In the context of this study we formulated the following research questions (RQs):

- *RQ_1: What semantic anchors used in CLANdroid produce better results when compared to the others?* The purpose of this RQ is to explore semantic anchors that are specific to Android apps such as user permissions declared in the manifest files, sensors used by the apps, and Android intents. Specifically, we evaluated whether these Android-specific semantic anchors outperform API calls, which were previously used by *CLAN* for detecting similar desktop Java systems.
- *RQ_2: How orthogonal are the apps detected by CLANdroid as compared to Google Play?* The purpose of this research question is to compare the apps detected by $CLANdroid$ to the set of similar apps detected by Google Play's search engine.
- *RQ_3: Do third-party libraries and obfuscated apps impact the accuracy of CLANdroid?* Previous studies found that using APK files in empirical studies could introduce threats to validity of the results because of the impact of third-party libraries and obfuscated code [3]. In this study we used APK files to extract API calls, identifiers, user permissions, sensors, and intents. Thus, it is possible that third-party libraries embedded in the APKs and obfuscated code can impact the detection of similar apps.

B. Study Design

To answer RQ_1, we designed a survey aimed at assessing similar apps detected by *CLANdroid*. We asked participants to rank the similarity between a target app and a set of potentially similar apps generated by an instance of *CLANdroid* from the following six choices: $CLANdroid_{API}$ (API calls), $CLANdroid_{Int}$ (intents), $CLANdroid_{Perm}$ (permissions), $CLANdroid_{Sens}$ (sensors), $CLANdroid_{Ident}$ (Identifiers); and a *Combined* approach that combines API calls and identifiers. The participants evaluated 12 apps belonging to different domain categories, and their top-3 similar apps (in the same domain category) retrieved by each one of the *CLANdroid* instances, which accounts for a total of 48 apps that had to be inspected by the participants. The similar apps were never displayed with the ranking (e.g., top-1) to avoid any type of bias of the participants. In the survey we provided links to the apps' websites at Google Play, and we asked the participants to make their judgments based on the following information available on Google Play: apps' descriptions, permissions list, screenshots, and the "what's new" section. Our decision for 12 apps and 3 similar apps was made to reduce early drop-out rate and to avoid invalid answers from participants trying to finish the survey quickly. Our assumption was that inspecting one app website will take a participant one minute on average, thus, a question will be answered in five minutes, which in total is 60 minutes to complete the survey. The similarity was evaluated by the survey participants using the Likert scale:

1) **Completely dissimilar:** the participant is highly confident that the app is dissimilar to the source app.
2) **Mostly dissimilar:** it is unclear if the app is similar to the source app.
3) **Mostly similar:** there are some similarities between the app and the source app.

[5]In addition to the metadata and application store reviews, for a specific app, Google Play provides a list of similar apps in the same category.

4) **Highly similar:** the participant is highly confident that the app is similar to the source app.

The 12 target apps were selected randomly from our dataset of 14,450 free Android. The list of apps and links to Google Play are provided with our online appendix [24].

Concerning the distribution protocol, we designed six versions of the survey using the Qualtrics [25] tool. The six versions reflect a cross-validation design in which an instance of the survey evaluated 12 apps but the top-3 similar of each two apps were detected using an instance of *CLANdroid*. Therefore, each participant evaluated results obtained with six different instances of *CLANdroid* (i.e., *CLANdroid*$_{API}$, *CLANdroid*$_{Int}$, *CLANdroid*$_{Perm}$, *CLANdroid*$_{Sens}$, *CLANdroid*$_{Ident}$, and *Combined*). It should be noted that the six versions of the survey assure that the results from six *CLANdroid* instances for the 12 apps were evaluated. We recruited participants via email and provided the details of the survey (without mentioning that the similar apps were detected by *CLANdroid*).

In addition to the survey, in order to identify the features of *CLANdroid* when compared to *Google Play* (RQ_2), we used 14,450 free Android apps, and a goldset of similar apps listed by Google Play. Thus, given an app $a_i \in A$, and A the context of our study, we used the six *CLANdroid* instances to detect similar apps to all the $a_i \in A$. The similar apps were detected in the complement set of each a_i (i.e., $A - a_i$). We used the Google Play's *Goldset* as a reference for evaluating the accuracy of the approaches. Our decision is motivated by the fact that manually annotating a set of 14,450 apps is a time-consuming and expensive task, and the goldset availability provided us with a ground truth for evaluating the approaches automatically. Therefore, after detecting similar apps for each a_i we looked for the ranking of the apps belonging to a_i goldset in the list of similar apps retrieved by each *CLANdroid* instance, then, we evaluated the rankings using two metrics: the top rank (TOP_R) of any application in the goldset for a_i, and the average rank (AVG_r) of all the apps in the goldset for a_i. Given an app a_i, and $a_j \in Goldset(a_i)$, $TOP_R(a_i)$ and $AVG_r(a_i)$ are computed as in Eq. 1 and 2.

$$TOP_r(a_i) = \min_{a_j}[rank(a_j)] \quad (1)$$

$$AVG_r(a_i) = \frac{1}{|Goldset(a_i)|} \sum_{j=1}^{|Goldset(a_i)|} rank(a_j) \quad (2)$$

For instance, given an app a_0 with apps a_2, a_{10}, and a_{20} in its goldset, we will check each of the six *CLANdroid* instances to compute the top rank (i.e., position closer to 1) of the apps a_2, a_{10}, and a_{20}. Thus, app a_{20} may be detected at rank 10 for *CLANdroid*$_{API}$, app a_{10} may be detected at rank 5 for *CLANdroid*$_{Ident}$, etc. For the average rank, we computed the average of the rankings for each app in the goldset (e.g., average of rank(a_2), rank(a_{10}), and rank(a_{20})) when using each *CLANdroid* instance.

Finally, for RQ_3 we computed the top rank and average rank of the goldset as in RQ_2. However, we considered only project-specific classes (i.e., excluding third-party libraries), and we removed obfuscated apps. For considering only the project-specific code and removing obfuscated apps, we followed the guidelines from prior work [3]. Note that the results from *CLANdroid*$_{Perm}$ are not impacted by third-party libraries because user permissions are extracted from manifest files.

C. Analysis Method

For RQ_1, we analyzed the survey responses using descriptive statistics; in order to validate whether there are significant differences between the similarity rankings (using the Likert scale) of each *CLANdroid* instance we used the Kruskal-Wallis test [26] with post-hoc test procedure for pairwise comparisons (i.e., Mann-Whitney with Bonferroni correction).

For the goldset-based analysis (RQ_2), we also compared the TOP_r and AVG_r series of the *CLANdroid* instances with the Kruskal-Wallis test with post-hoc procedure. For RQ_3, we only used pairwise comparisons using Mann-Whitney between the values of (i) TOP_r and AVG_r with and without third party libraries, and (ii) TOP_r and AVG_r with and without obfuscated code. For example, to measure if there is an impact of third-party libraries when using $CLANdroid_{API}$, we compared the TOP_r values when using $CLANdroid_{API}$ to the TOP_r values when using $CLANdroid_{API}$ but excluding third-party libraries from the context.

In all the tests we evaluated statistical significance at an alpha level = 0.05 and applied a Bonferroni Correction to the p-values when required. We also computed the Cliff's delta d effect size [27] to measure the magnitude of the difference in all the tests. We followed the guidelines in [27] to interpret the effect size values: negligible for $|d| < 0.147$, small for $0.147 \leq |d| < 0.33$, medium for $0.33 \leq |d| < 0.474$ and large for $|d| \geq 0.474$.). We are assuming neither population normality nor homogeneous variances, therefore, we choose non-parametric methods (Kruskal-Wallis test, Mann-Whitney test, and Cliff's delta).

D. Replication Package

The data set used in our study is publicly available at http://www.cs.wm.edu/semeru/data/clandroid. Specifically, we provide: (i) the list (and URLs) of the studied 14,450 free Android applications; (ii) the questions used for the survey; and (iii) the list of similar apps detected by *CLANdroid*. We also provide an online version of *CLANdroid*, which is available at http://www.semeru.info/clandroid

E. Threats to Validity

In this section, we discuss threats to validity of the experimental design for *CLANdroid* and how we address them.

1) Internal Validity: **Goldsets.** It is important to note that similarity between apps in the goldsets for *CLANdroid* are not symmetrical. Thus, if an app a_j is found in the goldset for an app a_i, this does not mean that the app a_i will be in the goldset for the app a_j. Because we reduce the goldset so that it only contains apps we have in our dataset (for practical reasons we cannot continuously download goldset apps), this means that the app B may have no apps in our dataset that

are also in its goldset. However, although the app a_j may not have relevant goldset apps to be ranked, we cannot discard a_j from the dataset since all apps that reside along with a_j in the same dataset will then suffer, e.g., remove the only app in their own goldset, will turn them into an "outlier" app like a_j.

We also assumed that the goldsets provided by Google Play are always correct. That is, each app listed on Google Play as similar should be in the "definitely similar" category. However, the quality of the similar app suggestions by Google Play varies depending on the context and we do not know the underlying mechanism used by it to detect similar apps.

App Categories. Regarding our similarity rankings based on the category of the app used as a query, note that the similarity score is already affected by the other apps in the dataset even if they are from a different category. That is, even if we only rank apps that are of the same category as the queried app, due to the way TF-IDF is computed, the TDM for LSI is affected, and thus the similarity scores are changed. However, if we decided to run LSI for each app only using the apps from each category, we would have to run both the TDM Builder and LSI computations 155 times (five times regardless of category for the different ranking methods, and 150 times for those five times for each category).

Main Package Extraction. When we extracted the main package from the manifests of each app in order to compute similarities between the apps without including any information from third-party libraries (TPLs), we found that some apps did not specify a main package in their manifest. In these cases, we chose to use the name of the app's package as the main package, as the majority of the apps in our dataset followed this design. However, we also detected 649 apps that specified a main package in their manifest that did not exist in the decompiled source code. For instance, the app Race, Stunt, Fight, Lite! [28] had the package named `ac.lite`, thus, the non-TPL information should be decompiled to `/ac/lite/` , but this directory does not exist within this app.

We investigated this by examining the first activity to be launched in the manifest (the first class to be executed), which we detected by searching for the `Launcher` Android intent also within the manifest. For this app, we found that the first class to be executed was `com.unity3d.player.UnityPlayerProxyActivity`. Unity3D [29] is a game engine that can be used to generate cross-platform applications, thus we know that this app used this game engine to generate some of the code for this application. However, due to the use of a third-party engine such as Unity3D, we are unable to distinguish parts of classes or even entire classes that were created solely by the developers. Some of the other 649 applications use various libraries and engines, such as MonoGame [30] or even the developers own library used for multiple applications. However, because these classes and code were not written specifically for a single application, we opted to exclude this code to prevent high similarities between apps simply because they relied on the same library or engine. Thus, we were unable to extract any information from the source code of these 649 applications since it was not possible to distinguish what code did and did not belong to a TPL because the specified main package did not exist.

2) External Validity: **Survey Participants Application Dataset.** Although the apps in our dataset are representative of all the Google Play categories, we cannot guarantee that these results would hold for the entirety of Google Play (including paid apps). Also, the evaluation of similarities reported by the survey participants might not be representative of the opinions of all the users of Android apps. In the future, we are also planning on computing Android-specific diversity metrics to quantify generalizability of our datasets [31].

IV. Empirical Results

In this section we present the empirical results aimed at answering the RQs. In addition, we include the cases that we manually inspected to support our quantitative findings.

A. RQ_1: Accuracy of CLANdroid semantic anchors

27 people accepted to participate in the online survey. It is worth noting that our target was to evaluate similarity from the viewpoint of users, therefore, we invited not only developers, but also end users. Each participant evaluated 48 apps in 12 questions presenting a target app and three potentially similar apps.

Fig. 3 depicts the similarities reported by the survey participants and Table I reports the comparisons that have statistically significant difference. On average, when considering the results for the Top-3 apps (Fig. 3-a) detected by each instance of $CLANdroid$, the survey participants reported that $CLANdroid_{API}$ is the instance providing the most similar apps with mean similarity of 2.54 (median = 3), which suggests that the apps detected by $CLANdroid_{API}$ are mostly similar. $CLANdroid_{API}$ was also the instance with the most number of apps rated as highly similar. The distribution of answers for $CLANdroid_{API}$ are: 38 (completely dissimilar), 41 (mostly dissimilar), 40 (mostly similar), and 43 (highly similar). In terms of average/median results, the next two instances with the best results are $CLANdroid_{Ident}$ with mean similarity of 2.51 (median = 3) and $CLANdroid_{Perm}$ with average similarity of 2.49 (median = 3). $CLANdroid_{Int}$ is the fourth instance in terms of results (mean similarity = 2.32, median = 2); and the combination of APIs and identifiers (i.e., $Combined$) neither improves the results of $CLANdroid_{API}$ nor $CLANdroid_{Ident}$. Finally, the worst results were achieved by $CLANdroid_{Sens}$ with mean similarity of 1.90 (median =2) and the following distribution: 68 (completely dissimilar), 54 (mostly dissimilar), 27 (mostly similar), and 13 (highly similar). There is a clear and significant difference between $CLANdroid_{Sens}$ and the rest of the instances, which can be observed not only in the boxplots, but also confirmed with the p-values of the Mann-Withney tests and the effect sizes (i.e., Cliff's d). All the comparisons using $CLANdroid_{Sens}$ as the treatment group (Table I) are the only cases with p-values less than the targeted alpha (after

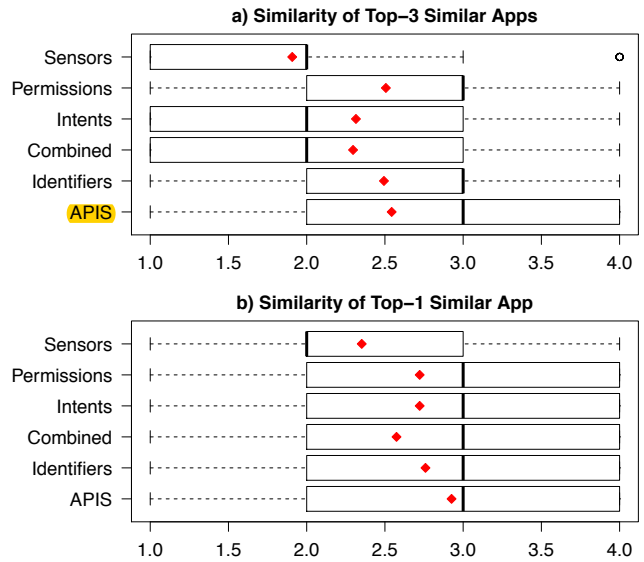

Fig. 3. Similarity (reported by survey participants) of a) Top-3 similar apps, and b) Top-1 similar app, detected by each *CLANdroid* instance. The values in the x-axis are based in the discrete values of the similarity scale: 1 (completely dissimilar), 2 (mostly dissimilar), 3 (mostly similar), and 4 (highly similar).

TABLE I
PAIRWISE COMPARISONS WITH STATISTICALLY SIGNIFICANT DIFFERENCES FOR THE SIMILARITY VALUES REPORTED BY THE PARTICIPANTS (TOP-3 SIMILAR APPS). P-VALUES AND CLIFF'S d ARE REPORTED. THE ALPHA VALUE AFTER BONFERRONI CORRECTION IS $\alpha = 0.0033$ (I.E., 0.05 /15 COMPARISSONS)

Control	Treatment	p-value	Cliff's $\|d\|$
$CLANdroid_{API}$	$CLANdroid_{Sens}$	2.72e-07	0.31790
$CLANdroid_{Ident}$	$CLANdroid_{Sens}$	7.46e-07	0.30571
$Combined$	$CLANdroid_{Sens}$	0.00076	0.20694
$CLANdroid_{Int}$	$CLANdroid_{Sens}$	0.00064	0.20995
$CLANdroid_{Perm}$	$CLANdroid_{Sens}$	9.79e-07	0.30251

Bonferoni correction), and the comparisons report medium effect sizes.

We also investigated the case of the Top-1 similar apps. Fig. 3-b depicts the distribution of the similarity values reported by the survey participants for the Top-1 similar apps detected by each instance of *CLANdroid*. All the instances (except for $CLANdroid_{Sens}$) had a median similarity of 3, and the average values are in the range (2.5, 3), which suggest that the participants reported the top-1 apps detected by almost all the instances of $CLAN$ are **highly similar**. However, the Mann-Whitney tests report only one significant difference between $CLANdroid_{API}$ and $CLANdroid_{Sens}$ with p-value=0.004, and medium effect size $|d|$ =0.306.

To understand the performance of the $CLANdroid$ instances based on Android-specific attributes we manually checked the apps detected as similar. Sensors are not widely used in our dataset, and there are only 13 types of sensors than can be used by Android apps [32]. *CLANdroid* ranks applications with the exact same similarity values at the same rank, therefore, the sensors attributes may not be as useful for detecting similar apps. For instance, if an app a_0 has three applications in its goldset (applications a_1, a_2, and a_3), and apps $a_1$and a_2 both utilize the exact same sensors as the application a_0, then they will both have a similarity value of 1.0 when compared to a_0. Thus, both applications will be ranked at position 1, and an app a_3 will be ranked at position 3, as it is the third-most similar application to application a_0. Due to the low number of unique sensors used in our dataset (10), it can be common for apps to use the same combination of sensors, especially if the application only uses one or two sensors. This also means that all applications that do not use any phone sensors have a perfect similarity value as well. 11,385 of the applications in our dataset make no use of any of the sensors, and thus all of these applications are deemed similar when ranked by sensors alone. However, while the sensors ranking method alone may not be the most effective, it might be combined with other ranking methods to help detect similar applications more accurately.

Permissions are a unique way of detecting similar apps due to its wide variance in ranking apps. Although there is a list of 145 official Android permissions [33], we detected over 10 times this amount of unique permissions in our dataset. This is possibly due to applications being able to create custom permissions. Thus, the more permissions an app has, the more likely that the top ranked apps by *CLANdroid* are to be functionally similar. The opposite also holds true: if an app has a single permission such as `android.permission.INTERNET`, then every app, which has only this permission, will be marked with a perfect similarity. One example that demonstrates the ineffectiveness of permissions is when considering the app *Slots Royale - Slot Machines* [34]. This app has four Android standard permissions: `READ_PHONE_STATE`, `ACCESS_COARSE_LOCATION`, `ACCESS_NETWORK_STATE`, and `INTERNET`. The app *Tennis Score* [35] has these exact same permissions, and thus is marked as a perfectly similar app in $CLANdroid_{Perm}$ (and because it is perfectly similar, it must be at rank 1). However, the app *Slots Free (5 Slot Machines)* [36] is a similar app part of the Google Play goldset, and while it contains the four permissions that *Slots Royale* has, it also has five additional permissions. Simply adding these additional permissions pushes the similarity ranking of this app down from the highest similarity level (rank 1) to rank 1,550.

Summary for RQ$_1$. *The evaluation of the results collected with our survey suggest that $CLANdroid$ is an effective tool for detecting similar apps, in particular when using APIs, identifiers, permissions, and intents as semantic anchors. Using sensors for detecting similar apps appears to be ineffective, because the set of sensors used by Android apps is reduced, and the detection based only on sensors results in too many false positives. Moreover, although, there is no statistically significant difference between using APIs and the other semantic anchors, the $CLANdroid$ instance based on APIs provided the highest number of apps rated as "highly similar".*

B. RQ$_2$: CLANDroid vs. Google Play

As mentioned in Section III, Google Play provides, for a target app, a list of "similar apps". We also used that list of similar apps to evaluate the performance of $CLANdroid$

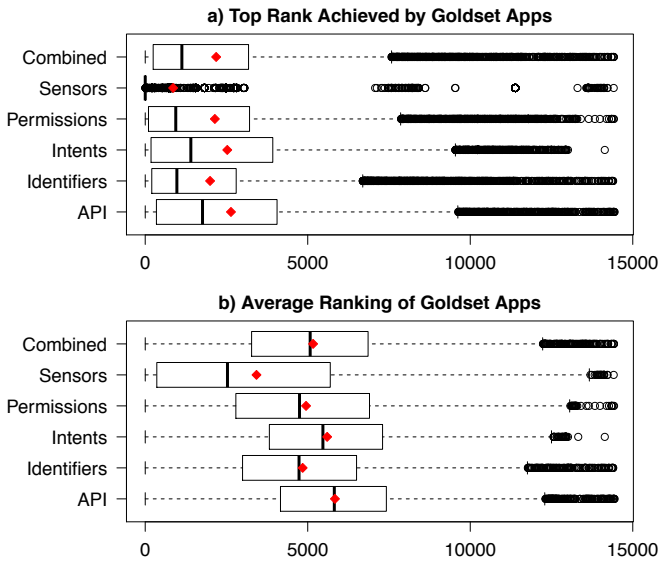

Fig. 4. *CLANdroid* instances and goldset behavior described by a) top ranked app, and b) average ranking in goldset.

instances when ranking the apps in the Google Play goldset. We used the $CLANdroid$ instances with the complete dataset of apps to consider also similar apps belonging to different categories (it is worth noting that for the survey we used apps detected as similar in the same category of the target app).

The results of the goldset-based analysis are summarized in Fig. 4. The figure depicts the distribution of the rankings provided by $CLANdroid$ to the apps reported as similar by Google Play (i.e., Google Play goldset). On average, using APIs and Intents deliver the worst rankings for the goldset, when analyzing TOP_r (Fig. 4-a). The mean values in Fig. 4-a ordered by best rankings are: 858.2 for $CLANdroid_{Sens}$ (median = 1), 1,996 for $CLANdroid_{Ident}$ (median = 974.5), 2,143 for $CLANdroid_{Perm}$ (median = 940), 2,183 for *Combined* (median = 1,128), 2524 for $CLANdroid_{Int}$ (median = 1,404), and 2,638 for $CLANdroid_{API}$ (median = 1,763). This result is also reflected in the case of AVG_r (Fig. 4-b). From best AVG_r to worst, based on the average value in the boxplot, we obtain: $CLANdroid_{Sens}$ (mean=3,424, median=2,531), $CLANdroid_{Ident}$(mean = 4,844, median=4,733), $CLANdroid_{Perm}$ (mean =4,948, median = 4,747), *Combined* (mean=5,165, median = 5,075), $CLANdroid_{Int}$ (mean = 5,597, median = 5,467), and finally $CLANdroid_{API}$ (mean = 5,837, median = 5,818).

The post-hoc procedures with the Mann-Whitney tests show statistically significant differences in all the cases for AVG_r and TOP_r, except for the comparison between $CLANdroid_{Ident}$ and $CLANdroid_{Perm}$, and $CLANdroid_{Int}$ and *Combined*. However, when looking into the magnitude of the differences, (i.e., Cliff's delta) in most of the comparisons, the differences are negligible (i.e., $|d| < 0.147$) and small (i.e., $0.147 \leq |d| < 0.33$). The magnitudes are only medium and large when comparing the results of using sensors as semantic anchors against the others; this case is confirmed with the boxplots, which shows that the best rankings for the goldsets are provided when using sensors ($CLANdroid_{Sens}$). Without considering $CLANdroid_{Sens}$, the best ranking of the goldset apps is achieved when using only identifiers ($CLANdroid_{Ident}$). Ranking by permissions is the second best, outperforming both APIs and intents.

On average, rankings of the Google Play's *Goldset* apps are far from the top-positions in all the approaches. When using TOP_r, there were only 471 apps in our dataset that had an app in their goldset ranked at position 1 for any ranking method (e.g., an app a_i may have an app from its goldset ranked at position 1 for $CLANdroid_{Ident}$, while an app a_j may have an app from its goldset ranked at position 1 for $CLANdroid - Sens$). When using TOP_r but only taking into consideration apps from our dataset that are of the same category as the queried application, this number increases to 1,134. This shows that at least 663 apps have an app from a different category being ranked higher than an app in the queried app's goldset for any ranking method.

We found evidence of apps ranked by *CLANdroid* in top positions, which do not belong to the *goldset*, but are still closely related. For example, when we checked the rankings for the popular game app *Angry Birds* [37], the top ranked app for each approach was *Angry Birds Space* [38]. The second ranked app by intents was *Hamster: Attack!* [39] (by Backflip Studios), an app in the `Casual` category. The third ranked app by APIs and identifiers is *The Sims FreePlay* [40]. The apps in the goldset (e.g., *Angry Monkey* [41] and *NinJump* [42]) do not appear to be functionally similar to *Angry Birds* and this suggests that Google Play might use textual similarity between the apps descriptions and names to detect similar apps.

Summary for RQ$_2$. *$CLANdroid$ is able to retrieve similar apps belonging to different categories, while Google Play lists as similar only apps in the same category. Our results suggest that Google Play's detection mechanism is likely to be based not only on textual similarities of descriptions, but also on sensors. The latter observation is supported by the fact that the best rankings for the goldset apps were obtained using the $CLANdroid_{Sens}$. However, as explained in Section IV-A, using sensors as the only source of information results in a higher rate of false positives for detecting similar apps.*

C. RQ_3: The impact of third-party libs and obfuscated apps

Fig. 5 depicts the distribution of the rankings provided by the $CLANdroid$ to the apps in the Google Play's *goldset* when excluding the third-party libraries from the analysis. A complete list with the results of the statistical tests is provided with our online appendix. We did not include the results of $CLANdroid_{Perm}$ because user permissions were extracted from manifest files, therefore, excluding third-party libraries from the analysis does not change the results.

In terms of AVG_r, $CLANdroid_{Ident}$ is the best approach and $CLANdroid_{API}$ is the worst: $CLANdroid_{Sens}$ (mean = 1,058, median = 1), $CLANdroid_{Ident}$ (mean = 4,350 , median = 3,991), $CLANdroid_{Int}$ (mean = 4,740, median = 4,860), $CLANdroid_{API}$ (mean = 5,253, median = 5,040); in

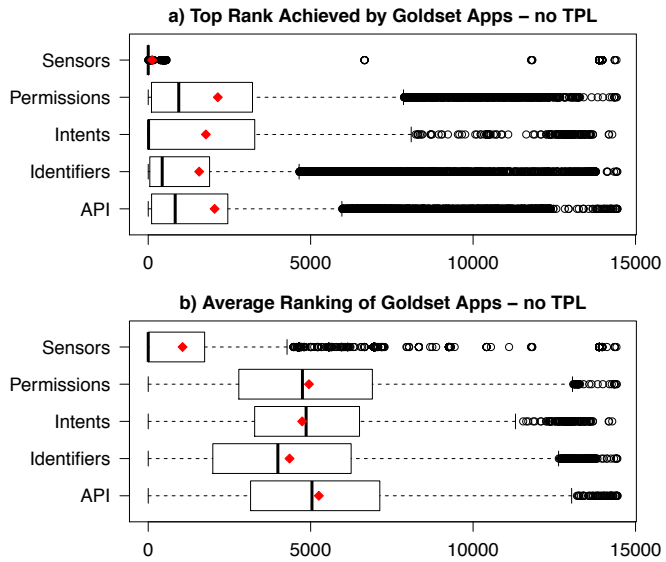

Fig. 5. *CLANdroid* instances and goldset behavior described by a) top ranked app, and b) average ranking in goldset, when excluding third-party libraries.

terms of TOP_r the results also hold: $CLANdroid_{Sens}$ (mean = 120.4, median = 1), $CLANdroid_{Ident}$ (mean = 1,569, median = 430.5), $CLANdroid_{Int}$ (mean = 1,775, median = 11), $CLANdroid_{API}$ (mean = 2,044, median = 830.5). There are significant differences between the values of TOP_r and AVG_r produced by the $CLANdroid$ approaches when including/excluding third-party libraries. In fact, the rankings are improved, in all the cases, after excluding third-party libraries from the analysis. The magnitude of the differences reported by the Cliff's d effect size are medium-large.

An example that illustrates the impact of third-party libraries is the following. The app *Night Vision Cam* [43] experiences a large increase in its average and top goldset rankings when removing third-party libraries from the data. One application in the goldset for *Night Vision Cam* is *LiveKey Camera* [44]. When using the whole dataset including third-party libraries, the similarity rankings of *Night Vision Cam* were 4,928 for $CLANdroid_{API}$ and 6,852 for $CLANdroid_{Ident}$. However, when excluding third-party libraries, these rankings improved drastically to positions 50 and 266 respectively, with the other attributes also reflecting this change. Upon further investigation, we found that while *LiveKey Camera* contained only app-specific code (i.e., no third-party libraries), *Night Vision Cam* utilized Google Ads. We observed that *Night Vision Cam* had only 21 classes in its main package, whereas the Google Ads library had 156 classes. This example demonstrates the large impact that TPLs can have when detecting similar applications, particularly in cases such as this where the TPL "outweighs" the application's code due to size and the number of classes.

Concerning the results when excluding obfuscated code, there are significant differences in all the cases when comparing to rankings of apps including obfuscated code. However, the magnitude of the differences is negligible, in most of the cases, according to the Cliff's d effect size. The negligible values, in terms of effect size, could be explained by the fact that only 1,458 obfuscated apps were identified in our dataset. There are improvements in the average rating (i.e., AVG_r) of the goldset apps. This is not a surprising result because removing the obfuscated apps reduces the size of the dataset, and obfuscated apps ranked in top positions are removed from the ranking list. Therefore, AVG_r results after removing obfuscated apps suggest that *CLANdroid* is able to find similar apps even including obfuscated apps because API calls, sensors, user permissions, and intents are part of the Android SDK (i.e., their calls/declarations in Android apps can not be obfuscated).

However, the behavior of TOP_r is different. The top ranked apps from the goldset, on average, lost positions in the ranking list when removing obfuscated code. This behavior is not evident when analyzing the AVG_r, and is opposite to the improvement of AVG_r. Upon manual inspection of the apps, we found a possible explanation for the case of TOP_r when excluding obfuscated applications. For instance, in our dataset we have four goldset applications for the *Smart AppLock (App Protector)* application [45]. However, when we removed obfuscated applications there were only two goldset applications remaining as the other two were marked as obfuscated applications. We noticed that in $CLANdroid_{Ident}$ for *Smart AppLock*, an obfuscated application (*AppLock* [46]) is ranked at the top position of 134, while the next highest ranked goldset application known as *App Lock (Smart App Protector)* [47] is at the position 7,220. However, after removing obfuscated apps from the dataset, *App Lock (Smart App Protector)* becomes the top-ranked app with its position changing to 4,460.

We noticed a similar case when detecting similar applications for *Infinite Racing* [48]. We found four applications in our dataset which are in the goldset, and when we removed the obfuscated applications we had two goldset applications remaining (i.e., there were two obfuscated applications and two non-obfuscated applications in the original goldset for this application). Similarly to the previous example, the ranking of the two non-obfuscated applications improved after the removal of the obfuscated apps. However, the new rankings of the non-obfuscated applications are not as good as the rankings of the obfuscated applications.

Summary of RQ$_3$. *The results demonstrate that the accuracy of CLANdroid is significantly (negatively) impacted by the inclusion of third-party libraries. Excluding third-party libraries moved the average rankings (AVG_r) up by, on average, a minimum of 490 positions. We found that without including third-party libraries, each CLANdroid instance improved both its top app rankings (TOP_r) and average rankings significantly. However, while we also found that there are differences in the rankings when excluding applications we detected as obfuscated, the magnitude of these differences is negligible in most cases. We also found that while the average rankings improved when we removed the obfuscated applications, the top rankings worsened due to obfuscated applications occupying the top ranks.*

TABLE II

RECENT STUDIES ON DETECTING SIMILAR APPS. WE USE M FOR MOBILE AND D FOR DESKTOP APPS. THE NEXT COLUMN LISTS THE NUMBER OF APPS IN THE DATASET, AND THE TPL COLUMN MARKS IF THE STUDY CONSIDERED THE IMPACT OF THIRD-PARTY LIBRARIES WITH A YES, NO, OR NA (NOT APPLICABLE). FINALLY, THE MARKET CATEGORY STATES WHERE THE APPS WERE ACQUIRED FROM- MM : MULTIPLE MARKETS, NR : NOT REPORTED, GP : GOOGLE PLAY, FB : FREEBSD, SF : SOURCEFORGE, E: ECLIPSE PLUGINS.

Study	Purpose	Information Type	Platform	#apps	TPL	Market
Michail and Notkin [14]	Detecting similar libraries	Library source code	D	NA	NA	NR
Kawaguchi *et al.* [49]	Automatic Categorization	Source code identifiers	D	41	NA	SF
Crussell *et al.* [50]	Detecting cloned and rebranded apps	Java bytecode	M	>265K	YES	MM
Li *et al.* [51]	Using similarities to address security	File directories	M	>58K	NO	MM
Bajracharya *et al.* [52]	Source code retrieval	API calls from source	D	346	NA	E
Chen *et al.* [17]	Detecting cloned apps to address security	Methods from SMALI code	M	>150K	YES	MM
Cubranic *et al.* [53]	Recommending Software Artifacts	Issue-tracking	D	1	NA	E
Moritz *et al.* [54]	API search engine	API methods	D	13K	NA	NR
Gorla *et al.* [55]	Finding unadvertised behavior in apps	API invocations from SMALI	M	>22K	YES	GP
Desnos *et al.* [56]	Detection of similar apps	Custom method signatures	M	2	NO	GP
Ye *et al.* [57]	Context-aware Browsing	Component repository	D	NR	NA	NR
McMillan *et al.* [58]	Finding relevant functions	Function call graph	D	> 18K	NA	FB
Thung *et al.* [59]	Detecting similar applications	Collaborative tagging	D	>100K	NA	SF
Wang *et al.* [10]	Detecting cloned apps	API invocations from SMALI	M	>100K	YES	MM
Shao *et al.* [60]	Detecting cloned apps	Statistical and Structural features	M	>169K	YES	MM

V. RELATED WORK

$CLANdroid$ is related to previous work on (i) source code engines, and (ii) approaches for detecting similar desktop and mobile apps, which are summarized in Table II. However, in this section we will only describe the approaches for detecting similar Android apps. For example, the tool AnDarwin by Crussell *et al.* [50] is a scalable approach for detecting similar Android apps using semantic information. AnDarwin extracts semantic vectors from source code methods in the apps; the main idea is that the methods can be combined in semantic blocks, therefore, if two semantic blocks are code clones, then the semantic vectors representing these blocks are considered as similar. The directory structure in mobile apps has been also used to detect similar apps; for instance, DStruct [51] decompiles and APK and walks through the directories and files of the app to construct a tree, which represents the directory structure. DStruct computes the percent difference between two trees to represent the similarity between two applications. Thus, the smaller the percent difference the more similar the apps are based on their directory structures.

Other approaches have proposed the usage of centroids, topics, and method signatures to detect similar apps. Chen *et al.* [17] detect similar apps by comparing centroids created from dependency graphs at method level. However, these similarity measures are used to draw a boolean value conclusion on the app's core functionality cloning. That is, either two apps are marked as clones or not, which prevents partial similarity detection. Chen *et al.* evaluated their approach across multiple different Android markets, yet did not use Google Play. Gorla *et al.* [55] applies Latent Dirichlet Allocation on the descriptions of over 32K applications; the *k*-means algorithm is then used to cluster the apps (by using the topics generated with LDA) and, thus, provide the ability to identify groups of apps with similar descriptions. Desnos [56] used method signatures to detect similar Android apps, where the signatures were composed of string literals, API calls, control flow structures, and exceptions. Wang *et al.* [10] proposed a two-phase approach that first removes code of third-party libraries from the APKs, and then uses fingerprints containing API calls to detect repackaged/cloned apps across different markets. Another work on detecting repackaged apps in two-phases is the one by Shao *et al.* [60], which clusters the apps using resources (e.g., strings and images) and statistical features initially, and then performs a second clustering stage using structural features. Finally, the work by Thung *et al.* [59] is also similar to $CLANdroid$, because they used an approach based on CLAN for detecting similar software systems, but instead of using API calls the authors used the tags for the systems in SourceForge [61].

VI. CONCLUSION AND LESSONS LEARNED

In this paper we present an approach for detecting *Closely reLated applications in ANdroid (CLANdroid)* that helps users find similar mobile apps. Our main contribution is a novel application of Android-specific features as semantic anchors for detecting similar Android apps. We extracted similar apps for our dataset from Google Play and evaluated six instances of $CLANdroid$ based on API calls, identifiers, sensors, intents, permissions, and the combination of APIs and identifiers. Evaluations provided by participants in an online survey suggest that $CLANdroid$ is able to detect similar apps, except when using detection mechanism based on sensors. In addition, conversely to Google Play, $CLANdroid$ is able to detect similar apps that belong to different categories.

These results can be exploited for designing search engines or recommendation systems. For instance, in the case of Android apps, high-level features are mostly described by API calls, identifiers in the code, intents, and by the permissions registered within the apps. Although sensors also contribute to defining the functionalities provided by the app, they should be combined with the other semantic anchors, yet assigning them lower weights (i.e., sensors) in the ranking functions. Our future work will be devoted to developing hybrid solutions for finding high-level similarities (e.g., features) and low-level implementation similarities (e.g., sensors or APIs).

In this paper, we also explored the impact of third-party libraries and obfuscated code when finding similar apps using several instances of *CLANdroid*. We found that excluding third-party libraries helps increasing the ranks of similar applications, thus corroborating previous findings on the impact of third-party libraries on the results [3]. However, while we also found that there are differences in the rankings when

excluding apps we detected as obfuscated, the magnitudes of these differences are negligible in most cases.

REFERENCES

[1] I. Mojica, M. Nagappan, B. Adams, and A. Hassan, "Understanding reuse in the Android market," in *ICPC'12*, 2012, pp. 113–122.

[2] I. Mojica, B. Adams, M. Nagappan, S. Dienst, T. Berger, and A. Hassan, "A large scale empirical study on software reuse in mobile apps," *IEEE Software Special Issue on Next Generation Mobile Computing*, 2013.

[3] M. Linares-Vásquez, A. Holtzhauer, C. Bernal-Cárdenas, and D. Poshyvanyk, "Revisiting android reuse studies in the context of code obfuscation and library usages," in *MSR'14*, 2014, pp. 242–251.

[4] C. McMillan, N. Hariri, D. Poshyvanyk, J. Cleland-Huang, and B. Mobasher, "Recommending source code for use in rapid software prototypes," in *ICSE'12*, 2012, pp. 848–858.

[5] M. Harman, Y. Jia, and Y. Zhang, "App store mining and analysis: Msr for app stores," in *MSR'12*, 2012, pp. 108–112.

[6] R. Minelli and M. Lanza, "Software analytics for mobile applications: Insights and lessons learned," in *CSMR*, 2013.

[7] M. Linares-Vásquez, G. Bavota, C. Bernal-Cárdenas, M. Di Penta, R. Oliveto, and D. Poshyvanyk, "Api change and fault proneness: A threat to the success of android apps," in *ESEC/FSE'13*, pp. 477–487.

[8] W. Martin, M. Harman, Y. Jia, F. Sarro, and Y. Zhang, "The app sampling problem for app store mining," in *MSR'15*, 2015, pp. 123–133.

[9] G. Bavota, M. Linares-Vasquez, C. Bernal-Cardenas, M. Di Penta, R. Oliveto, and D. Poshyvanyk, "The impact of api change- and fault-proneness on the user ratings of android apps," *Software Engineering, IEEE Transactions on*, vol. 41, no. 4, pp. 384–407, April 2015.

[10] H. Wang, Y. Guo, Z. Ma, and X. Chen, "Wukong: A scalable and accurate two-phase approach to android app clone detection," in *ISSTA'15*, 2015, pp. 71–82.

[11] Y. Tian, M. Nagappan, D. Lo, and A. Hassan, "What are the characteristics of high-rated apps? a case study on free android applications," in *ICSME'15*, 2015, pp. 301–310.

[12] K. Kontogiannis, "Program representation and behavioural matching for localizing similar code fragments," in *CASCON '93*. IBM Press, 1993, pp. 194–205.

[13] C. Liu, C. Chen, J. Han, and P. S. Yu, "Gplag: Detection of software plagiarism by program dependence graph analysis," in *KDD'06*. ACM Press, 2006, pp. 872–881.

[14] A. Michail and D. Notkin, "Assessing software libraries by browsing similar classes, functions and relationships," in *ICSE '99*, 1999, pp. 463–472.

[15] T. Sager, A. Bernstein, M. Pinzger, and C. Kiefer, "Detecting similar java classes using tree algorithms," in *MSR '06*, 2006, pp. 65–71.

[16] D. Schuler, V. Dallmeier, and C. Lindig, "A dynamic birthmark for java," in *ASE '07*, 2007, pp. 274–283.

[17] K. Chen, P. Liu, and Y. Zhang, "Achieving accuracy and scalability simultaneously in detecting application clones on android markets," in *ICSE'14*, 2014.

[18] E. Hull, K. Jackson, and J. Dick, *Requirements Engineering*. SpringerVerlag, 2004.

[19] W. Liu, K.-Q. He, J. Wang, and R. Peng, "Heavyweight semantic inducement for requirement elicitation and analysis," *Semantics, Knowledge and Grid*, vol. 0, pp. 206–211, 2007.

[20] J. Howison and K. Crowston, "The perils and pitfalls of mining Sourceforge." in *MSR*, 2004.

[21] M. Gabel and Z. Su, "A study of the uniqueness of source code," in *FSE'10*, 2010, pp. 147–156.

[22] C. McMillan, M. Grechanik, and D. Poshyvanyk, "Detecting similar software applications," in *ICSE'12*, 2012, pp. 364–374.

[23] S. C. Deerwester, S. T. Dumais, T. K. Landauer, G. W. Furnas, and R. A. Harshman, "Indexing by latent semantic analysis," *JASIS*, vol. 41, no. 6, pp. 391–407, 1990.

[24] M. Linares-Vásquez, A. Holtzhauer, and D. Poshyvanyk, "Clandroid online appendix," http://www.cs.wm.edu/semeru/data/clandroid/.

[25] "Qualtrics," http://www.qualtrics.com/.

[26] S. D.J., *Handbook of Parametric and Nonparametric Statistical Procedures*. Chapman & All, 2007.

[27] R. Grissom and J. Kim, *Effect sizes for research: Univariate and multivariate applications*. New York, NY: Taylor & Francis, 2012.

[28] "Race, stunt, fight, lite!" https://play.google.com/store/apps/details?id=ac.lite.

[29] "Unity3d," https://unity3d.com/.

[30] "Monogame," http://www.monogame.net/.

[31] M. Nagappan, T. Zimmermann, and C. Bird, "Diversity in software engineering research," in *ESEC/FSE'13*. ACM, August 2013.

[32] Google, "Android sensors," http://developer.android.com/guide/topics/sensors/sensorsoverview.html.

[33] "Android permissions." http://developer.android.com/reference/android/Manifest.permission.html.

[34] "Slots royale - slot machines," https://play.google.com/store/apps/details?id=com.mw.slotsroyale.

[35] "Tennis score," https://play.google.com/store/apps/details?id=RobotMoose.TennisScore.

[36] "Slots free (5 slot machines)," https://play.google.com/store/apps/details?id=com.viaden.slotsfree.

[37] "Angry birds," https://play.google.com/store/apps/details?id=com.rovio.angrybirds.

[38] "Angry birds space," https://play.google.com/store/apps/details?id=com.rovio.angrybirdsspace.ads.

[39] "Hamster attack," https://play.google.com/store/apps/details?id=com.backflipstudios.android.hamsterattack.

[40] "The sims free play," https://play.google.com/store/apps/details?id=com.ea.games.simsfreeplay_na.

[41] "Angry monkey," https://play.google.com/store/apps/details?id=com.tgb.kingkong.

[42] "Ninjump," https://play.google.com/store/apps/details?id=com.bfs.ninjump.

[43] "Night vision cam," https://play.google.com/store/apps/details?id=androix.com.android.NightVisionCam.

[44] "Livekey camera," https://play.google.com/store/apps/details?id=com.sonyericsson.androidapp.appkey.

[45] "Smart applock (app protector)," https://play.google.com/store/apps/details?id=com.sp.protector.free.

[46] "App lock," https://play.google.com/store/apps/details?id=com.domobile.applock.

[47] "Applock 2 (smart app protect)," https://play.google.com/store/apps/details?id=com.thinkyeah.smartlockfree.

[48] "Inifinite racing," https://play.google.com/store/apps/details?id=com.Sailfish.InfinityRacing.

[49] S. Kawaguchi, P. K. Garg, M. Matsushita, and K. Inoue, "Mudablue: an automatic categorization system for open source repositories," *J. Syst. Softw.*, vol. 79, no. 7, pp. 939–953, 2006.

[50] J. Crussell, C. Gibler, and H. Chen, "Scalable semantics-based detection of similar android applications," in *ESORICS'13*, 2013.

[51] S. Li, S. Hanna, L. Huang, E. Wu, C. Chen, and D. Song, "Juxtapp and dstruct: Detection of similarity among android applications," Department of Computer Science, The University of Auckland, Tech. Rep. UCB/EECS-2012-111, 2012.

[52] S. K. Bajracharya, J. Ossher, and C. V. Lopes, "Leveraging usage similarity for effective retrieval of examples in code repositories," in *FSE*, 2010, pp. 157–166.

[53] D. Čubranić and G. C. Murphy, "Hipikat: recommending pertinent software development artifacts," in *ICSE '03*, 2003, pp. 408–418.

[54] E. Moritz, M. Linares-Vaśquez, D. Poshyvanyk, M. Grechanik, C. McMillan, and M. Gethers, "Export: Detecting and visualizing api usages in large source code repositories," in *ASE'13*, 2013.

[55] A. Gorla, I. Tavecchia, F. Gross, and A. Zeller, "Checking app behavior against app descriptions," in *ICSE'14*, 2014.

[56] A. Desnos, "Android : Static analysis using similarity distance," in *HICSS'12*, 2012, pp. 5394–5403.

[57] Y. Ye and G. Fischer, "Supporting reuse by delivering task-relevant and personalized information," in *ICSE '02*, 2002, pp. 513–523.

[58] C. McMillan, M. Grechanik, D. Poshyvanyk, Q. Xie, and C. Fu, "Portfolio: Finding relevant functions and their usages," in *ICSE'11*, 2011.

[59] F. Thung, D. Lo, and L. Jiang, "Detecting similar applications with collaborative tagging," in *ICSM'12*, 2012.

[60] Y. Shao, X. Luo, C. Qian, P. Zhu, and L. Zhang, "Towards a scalable resource-driven approach for detecting repackaged android applications," in *ACSAC '14*, 2014, pp. 56–65.

[61] "Sourceforge. http://sourceforge.net/."