26th International Conference on Software Analysis, Evolution, and Reengineering (SANER'19), ERA Track

# Knowledge Graphing Git Repositories: A Preliminary Study

Yanjie Zhao[1], Haoyu Wang[1], Lei Ma[2], Yuxin Liu[3], Li Li[3], and John Grundy[3]
[1] Beijing University of Posts and Telecommunications, Beijing, China
[2] Harbin Institute of Technology, Harbin, China
[3] Monash University, Melbourne, Australia

***Abstract*—Knowledge Graph, being able to connect information from a variety of sources, has become very famous in recent years since its creation in 2012 by Google. Researchers in our community have leveraged Knowledge Graph to achieve various purposes such as improving API caveats accessibilities, generating answers to developer questions, and reasoning common software weaknesses, etc. In this work, we would like to leverage the knowledge graph concept for helping developers and project managers to comprehend software repositories. To this end, we design and implement a prototype tool called GitGraph, which takes as input a Git repository and constructs automatically a knowledge graph associated with the repository. Our preliminary experimental results show that GitGraph can correctly generate knowledge graphs for Git projects and the generated graphs are also useful for users to comprehend the projects. More specifically, the knowledge graph, on one hand, provides a graphic interface that users can interactively explore the integrated artefacts such as commits and changed methods, while on the other hand, provides a convenient means for users to search for advanced relations between the different artefacts.**

## I. Introduction

Git, created by Linus Torvalds in 2005 for managing the source code of Linux kernel, has become the most popular version control system in the world for tracking changes in general computer files during the development of software systems. Unlike traditional client-server version control systems such as CVS and Subversion, where the complete history of software projects is only stored in the server, Git comes with a full-fledged repository (e.g., full version-tracking abilities) that stores the complete history both locally and centrally, making it a promising repository for mining insights.

With the explosion of open source projects, Git has become even more prevalent for both developers and researchers. This momentum has further been boosted by the introduction of web-based version control services ( e.g., GitHub and BitBucket), which operates Git as their default mechanism for managing software systems. Those web-based Git services, on one hand, simplify the process of using Git repositories while on the other hand provide a user-friendly interface for developers and project managers to easily comprehend the project. Because of these advantageous, the web-based version control systems have achieved great success. For example, GitHub has attracted over 28 million users who contribute in total over 57 million repositories, including popular ones such as the Android framework code base.

Despite the big success of web-based Git systems, the current mechanism does not provide sufficient information for developers and managers to quickly and easily comprehend the projects. Indeed, the current graphical user interface of web-based Git systems does not provide an interactive visualisation for users to intuitively explore the relationship of different software artefacts. Furthermore, the information provided by the current graphical interface is also limited. More specifically, the changes made to the project (or program files) are usually listed via commits without providing statistical data about the overall changes. For example, project managers cannot directly observe the mostly changed program files, which could have been the pain points where bugs are frequently introduced.

To address the aforementioned limitations, in this work, we are interested in associating knowledge graphs with Git projects so as to provide an alternative means for developers and project managers to comprehend the projects. Knowledge Graph (KG), a new term introduced by Google in 2012 when Google integrates KG in its search engine, consists of a set of interconnected typed entities and their attributes. The reason why KG is chosen in this work is that the graph database (of KG) goes beyond the traditional relational database by supporting frequent schema changes, real-time data updates and query responses, and as well as allowing users to infer indirect facts in the graph.

Towards supplementing existing Git representations, we design and implement a prototype tool called GitGraph, which takes as input a Git project and outputs a knowledge graph specifically constructed for the project. The correctness and usefulness of GitGraph are then empirically evaluated via experimental results. Specifically, regarding the usefulness of the generated knowledge graph, we empirically show that it is capable of providing (1) an interactive visual overview of the project on which users can explore. It turns the original loose text structure into an associated physical structure. For example, researchers can visually view details of changes to a program file, e.g., to what extent it has been changed overall? (2) an interface for supporting advanced (or fine-grained) Git queries (graph query languages supported by the KG database). For example, users can leverage dedicated query scripts to search for the modification history of a given method over the graph, including when it was changed, who changed it, and what was changed?

We make available online our implementation, along with the scripts to replicate our experiments at

https://github.com/xuanyi531/GitGraph

## II. Methodology

Our objective in this work is to leverage the benefits of KG to supplement the representation of Git repositories so as to help developers and project managers better comprehend their Git projects. To this end, as our first attempt to achieve this goal, we design and implement a prototype tool called GitGraph, which takes as input a Git repository and outputs a knowledge graph associated to the repository and a set of web services that provide interfaces for researchers and practitioners to programmatically access the graph.

Fig. 1 illustrates the working process of GitGraph, which is mainly made up of three modules: (1) Metadata Extraction Module, (2) Graph Construction Module, and (3) Service Construction Module. We now introduce these three modules, respectively.

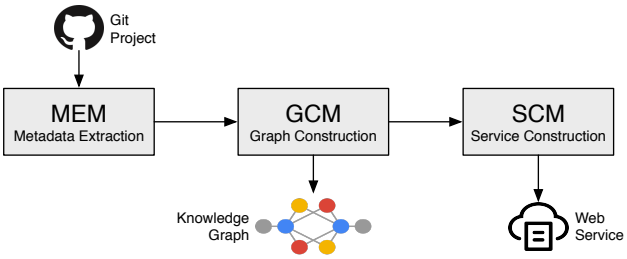


Fig. 1. GitGraph Overview.

### A. MEM: Metadata Extraction

Given a Git repository, GitGraph so far extracts automatically the following metadata:

- **Commits:** Since the evolution of the repository is mainly reflected by commits. GitGraph attempts to record commit-related metadata as detail as possible. More specifically, given a commit, GitGraph records its commit time, commit message, commit author, the changed files and the changed code (i.e., diff).
- **Files:** GitGraph accounts all the files presented in the Git repository, including not only program files but also other files such as libraries, XML configurations, images, etc.
- **Classes/Methods:** Regarding program files, in order to provide fine-grained changes, we also perform simple static analysis to those files, attempting to summarise the interior structure of the detailed implementation. For example, for Java files, we locate the classes, and their methods presented in the Java files. For each method, we further parse its code to highlight if certain (third-party) APIs are accessed into. We believe this information will be useful for understanding the usage of certain APIs (e.g., the official Java APIs or third-party library APIs).
- **Branches:** Like any other version control systems, Git also supports branch mechanism to facilitate teamwork. To avoid the possibility of messing with the mainline code, developers can create a branch diverging from the mainline code to implement their modules independently. In this work, we consider a branch of a Git repository as a small single-branch Git project and hence all the metadata mentioned above will be extracted for a branch.

### B. GCM: Graph Construction Module

After the extraction of metadata, the graph construction module (GCM) of GitGraph aims at connecting them to construct the knowledge graph. Fig. 2 illustrates the predefined schema of the knowledge graph that we plan to build. The shaded rectangle represents a node in the graph and, if applied, the property of the node is represented by a un-shaded box. Different nodes are connected via labelled edges where the labels are named based on the relationship of the nodes.

Because the files (especially program files) are continuously changed during the evolution of the Git repository, in this work, we consider the different versions of the same file as different nodes, which are differentiated by the commitID affix, representing the version when the file is changed. Similarly, classes and methods are presented following the same strategy. As long as the content of classes/methods is changed, GitGraph will generate a new node to record the change. Otherwise, the node from the previous commit will be used. Algorithm 1 presents the detailed working process of this strategy. The changed files and classes are integrated into the graph via new nodes (lines 5 and 10 respectively) while the unchanged files and classes are directly inherited from the previous commit (lines 18 and 12 respectively). For simplicity, the integration of method nodes is condensed into a single method *parseAndConnectMethods(cls)*, as shown in line 14.

**Algorithm 1** The Knowledge Graph Construction Process.

**Require:** $commit_{i+1}$ *//The subsequent commit of* $commit_i$
1: $commitNode = createCommitNode(commit_{i+1})$
2: $changedFiles = extractChanges(commit_{i+1})$
3: **for** $file \in changedFiles$ **do**
4: $fileNode = createFileNode(file.name, commit_{i+1})$
5: $connect(commitNode, fileNode)$
6: $classes = extractClasses(file)$
7: **for** $cls \in classes$ **do**
8: **if** $(md5(cls.body) != getClassMD5(commit_i, cls)$ **then**
9: $classNode = createClassNode(cls.name, commit_{i+1})$
10: $connect(fileNode, classNode)$
11: **else**
12: $connect(fileNode, getClassNode(commit_i)$
13: **end if**
14: $parseAndConnectMethods(cls)$
15: **end for**
16: **end for**
17: **for** $file \in getFiles(commit_i)$ & & $file \notin changedFiles$ **do**
18: $connect(commitNode, getFileNode(commit_i)$
19: **end for**

The output of this module is a knowledge graph that graph users can interactively play with. Fig. 3 presents an simplified example of a graph constructed from YahooNewsOnboarding [5]. Also, with the help of graph query languages (e.g., Cypher, a declarative, SQL-inspired language for describing patterns in graphs visually using an ascii-art syntax), users

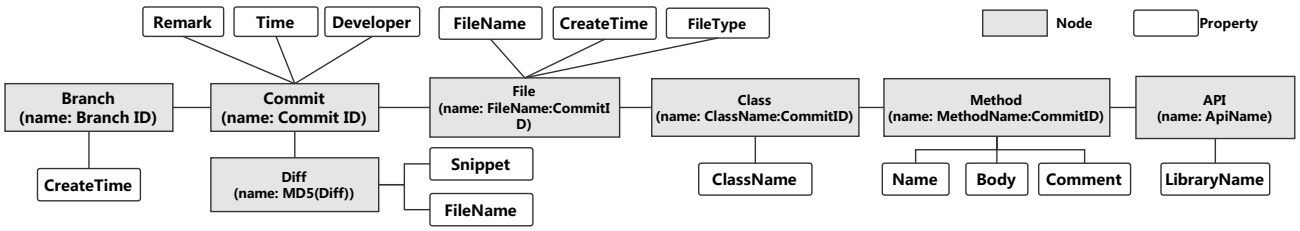


Fig. 2. The Schema of GitGraph's Graph Database

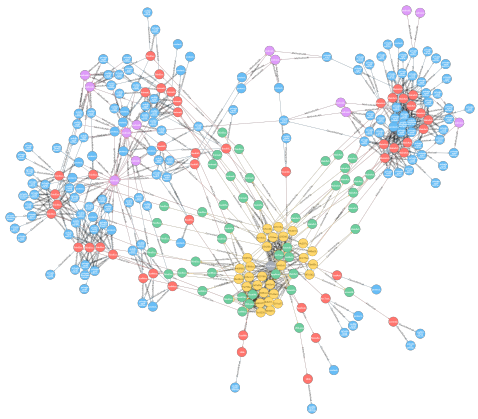

Fig. 3. An Example of Simplified Knowledge Graph.

can achieve complicated queries such as enquiring the top ten modified classes during the evolution of the Git repository (cf. Code 1).

**Code 1**

```
MATCH (c:Class)
RETURN c.name, count(*) AS cnt
ORDER BY cnt DESC LIMIT 10
```

### C. SCM: Service Construction Module

As discussed in the previous section, graph query languages can be leveraged to achieve advanced enquiries from knowledge graphs. Despite that graph query languages such as Cypher are relatively easy to write, users still need to spend some time to learn the language before mastering it, resulting in a user-unfriendly strategy, where impatient users may drop the usage of knowledge graphs.

To this end, as the last module of GitGraph, we aim at providing a means for end users to enquiry the graph without knowing any query languages (i.e., no need to learn and consequently write query scripts). To achieve this purpose, we implement the last module as a configuration-based web service generator, which automatically generates online web services wrapping the actual implementation of query scripts (configurable). As shown in Fig. 4, client apps only need to send HTTP requests, which can further be optimised with dedicated SDKs, to achieve advanced enquiries.

Now, to enquiry the top modified classes during the evolution of the Git repository, instead of using the script presented

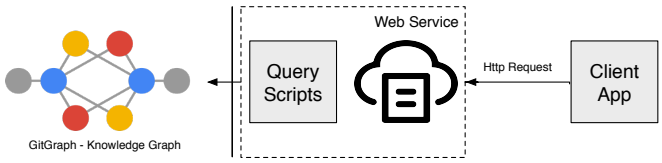


Fig. 4. Expected Usage Scenerio of GitGraph Knowledge Graph.

in Code 1, one can achieve it via the following HTTP request (cf. Code 2), where query languages are no longer needed.

**Code 2**

```
HTTP Get /topChangedClasses?num=10
```

## III. Evaluation

Towards verifying the correctness and usefulness of our approach, in this preliminary work, we aim to answer the following two research questions:

- **RQ1:** Can GitGraph correctly construct knowledge graphs for Git repositories?
- **RQ2:** Are GitGraph useful for comprehending Git repositories?

### A. Correctness

Since there is no automated way to verify if the generated knowledge graph is correct, we resort to manual verification in this work to confirm the correctness of our approach. More specifically, we select five Android app (relatively small) projects from F-Droid, which has their source code been made available on Github. We then manually build the knowledge graph based on the schema and rules introduced in the previous section. After that, we launch GitGraph on these five projects separately for knowledge graphs construction. Our experimental results show that all the automatically generated knowledge graphs are respectively and perfectly matched with the ones we manually build, demonstrating that GitGraph can indeed correctly construct knowledge graphs for Git repositories.

### B. Usefulness

To evaluate the usefulness of GitGraph, we select 10 popular and active Android app projects from F-Droid, as listed in Table I. These projects have 132 to 2,636 commits. The constructed KG for each of them has thousands of nodes and edges. For example, the app "suntimesWidge" has more than 17K nodes and roughly 2 million edges. It is not easy to comprehend such a large project with millions of changes.

TABLE I
EXPERIMENTAL RESULTS OF GitGraph for 10 Android App Git repositories.

| Project Name | # Commits | # Graph Nodes | # Graph Edges | Top 3 Changed Files | Top 3 Changed Classes | Top 3 Changed Methods |
|---|---|---|---|---|---|---|
| AFWall+ [6] | 1474 | 13779 | 617637 | main.java.../Api.java<br>main.java.../MainActivity.java<br>dev.../Api.java | Api<br>MainActivity<br>G | public void onCreate(...)<br>protected void onCreate(...)<br>public void onReceive(...) |
| BookWorm [4] | 510 | 5153 | 58522 | Main.java<br>BookEntryResult.java<br>DataHelper.java | Main<br>BookEntryResult<br>DataHelper | public void onCreate(...)<br>public void onPause()<br>public void onCreate(...) |
| Calendula [7] | 1383 | 18552 | 822798 | /.../MedicinesActivity.java<br>/.../CalendulaApp.java<br>/.../Med...Fragment.java | MedicinesActivity<br>Medicine...Fragment<br>CalendulaApp | protected void onCreate(...)<br>public View onCreateView(...)<br>public boolean on...Selected(...) |
| Flavordex [8] | 182 | 3628 | 59565 | main/.../EditCatFragment.java<br>main/.../ExportDialog.java<br>main/.../ViewFlavorsFragment.java | EditCatFragment<br>ViewPhotosFragment<br>ExportDialog | public View onCreateView(...)<br>public Loader onCreateLoader(...)<br>public Dialog onCreateDialog(...) |
| Kolab Notes [9] | 532 | 5817 | 217767 | main/.../OverviewFragment.java<br>main/.../DetailFragment.java<br>main/.../MainPhoneActivity.java | OverviewFragment<br>DetailFragment<br>MainPhoneActivity | protected void onCreate(...)<br>public void onActivityCreated(...)<br>public boolean on...Selected(...) |
| MaterialFBook [10] | 132 | 3011 | 21073 | main/.../MainActivity.java<br>main/.../NotificationsService.java<br>main/.../MainActivity.java | MainActivity<br>SettingsActivity<br>NotificationsService | protected void onCreate(...)<br>public boolean on...Selected(...)<br>public void onCreate(...) |
| Network Monitor [11] | 956 | 10127 | 421478 | jraf/.../NetMonService.java<br>jraf/.../LogActivity.java<br>main/.../Advanced...Activity.java | LogActivity<br>NetMonService<br>NetMonPreferences | protected void onCreate(...)<br>public ... getContentValues()<br>public Dialog onCreateDialog(...) |
| Padland [12] | 202 | 2169 | 29162 | main/.../PadViewActivity.java<br>main/.../PadLandDataActivity.java<br>main/.../PadListActivity.java | PadViewActivity<br>PadLandDataActivity<br>PadListActivity | protected void onCreate(...)<br>private WebView _makeWebView()<br>public void onCreate(...) |
| SuntimesWidge [13] | 2636 | 17826 | 1910041 | main/.../SuntimesActivity.java<br>main/.../SuntimesUtils.java<br>main/.../WidgetSettings.java | SuntimesActivity<br>SuntimesUtils<br>WidgetSettings | protected void initViews(...)<br>public void themeViews(...)<br>protected void updateViews(...) |
| Uber-ride [14] | 152 | 2394 | 27650 | main/.../LoginManager.java<br>test/.../LoginManagerTest.java<br>main/.../LoginActivity.java | LoginManager<br>LoginManagerTest<br>RequestDeeplink | public void setup()<br>protected void onCreate(...)<br>public void onLoad...Agent() |

In this section, we elaborate several use cases as examples to demonstrate the usefulness of our approach.

**Interactive graphical interface.** With GUI-based navigation provided by GitGraph, one can review the project more quickly and clearly, making it easy to understand the evolution of the corresponding project. For example, as shown in Fig. 5, one can quickly observe the relevant changes involved by a bug-fix commit. Furthermore, we believe this graphical interface can be also useful by researchers to quickly confirm their empirical findings when they need to manually verify the results obtained by mining Git repositories.

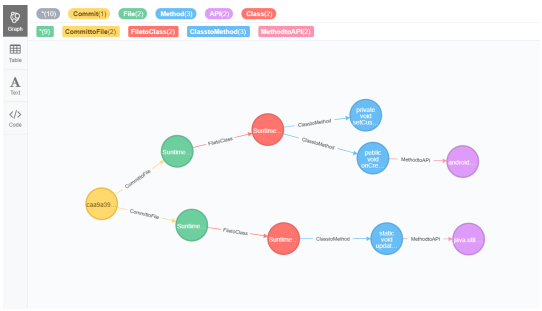

Fig. 5. Graphical representation of the changes involved by a Bug-fix commit.

**Advanced Git Query.** The knowledge graph generated by GitGraph not only provides a graphical representation for Git projects but also provides a means that goes beyond traditional Git management systems such as GitHub for supporting advanced searches. Researchers/users are able to query knowledge using graph query language scripts or simply using HTTP request as mentioned in Subsection II-C. The extracted knowledge can then be leveraged to better comprehend the projects so as to support the decision of future investigations.

As shown in Table I, the 5th, 6th, and 7th columns enumerate the top 3 changed files, classes, and methods, respectively. Such rankings go beyond traditional Git repositories as well as popular web-based Git systems such as GitHub to provide useful statistics for project managers and maintainers to quickly comprehend the projects. For example, based on the ranking of frequently changed classes, project maintainers can drill down the possible reasons so as to decide whether a thorough refactoring process is needed to avoid further changes if the previous changes are all resulted by a bad designing of the class in the first place. With the knowledge graph built by GitGraph, in addition to the three rankings shown in Table I, users can actually easily observe much more rankings such as the top contributing developers w.r.t. code qualities (i.e., with fewer changes after their commits).

We believe that the knowledge graph constructed by GitGraph can be leveraged to support various mining software repository studies. Just as an example towards demonstrating this possibility, we present in this work a preliminary study of identifying frequent buggy methods. In this work, we rely on a naive approach to identify buggy methods: as long as the method is changed by bug-fix commits, for which both "bug" and "fix" keywords are presented in their commit messages. Table II further enumerates the total number of bug-fix commits and the top 3 involved buggy methods for the 10 Git projects, which can be respectively obtained through a single script query or a single web service call.

TABLE II
TOP 3 CHANGED BUGGY METHODS.

| Project Name | # Bug Fix Commits | Top 3 Buggy Methods |
|---|---|---|
| AFWall+ | 393 | MainActivity.onCreate(...)<br>Connect...Receiver.onReceive(...)<br>Api.getApps(...) |
| BookWorm | 16 | BookSearch.onCreate(...)<br>BookDAO.insert(...)<br>BookDAO.build...Cursor(...) |
| Calendula | 211 | CalendulaApp.onCreate()<br>Confirm...Activity.saveSchedules()<br>Schedule...Activity.updateSchedule() |
| Flavordex | 18 | ViewFlavorsFragment.onCreate(...)<br>CatListDialog.onCreateLoader(...)<br>PhotoUtils.getTakePhotoIntent(...) |
| Kolab Notes | 161 | OverviewFragment.onActivityCreated(...)<br>DetailFragment.onActivityCreated(...)<br>OverviewFragment.onResume() |
| MaterialFBook | 8 | NotificationsService.notifier(...)<br>MainActivity.searchToolbar()<br>MainActivity.onPause() |
| Network Monitor | 123 | LogActivity.loadHTMLFile()<br>NetMonDatabase.onUpgrade(...)<br>HTMLExport.writeHeader(...) |
| Padland | 39 | PadViewActivity._makeWebView()<br>PadLandDataActivity.menu_group(...)<br>PadListAdapter.getGroup(...) |
| SuntimesWidge | 214 | Sun...Activity.onCreate(...)<br>GetFixHelper.getFix()<br>SuntimesUtils.calendar...String(...) |
| Uber-ride | 41 | LoginManager. val...on(...)<br>Legacy...Handler.checkValidState(...)<br>Legacy...HandlerTest. handle...Dialog() |

## IV. FUTURE WORK

In this work, we have presented GitGraph for constructing knowledge graphs for Git projects. Although so far GitGraph works for Git projects separately, it can be also leveraged to construct a KG for multiple Git projects. Indeed, different KGs generated for different Git projects can be seamlessly integrated via some common features such as their leveraged APIs and their presented common vulnerabilities/bugs.

Furthermore, we are not only interested in representing Git repositories but are also interested in going beyond simple representation to offer more capabilities. To this end, we attempt to also harvest metadata from other sources such as question-answer websites and bug tracking systems. Unlike traditional relative databases, which have a fixed schema that is difficult to extend after the database is created, graph databases come with a flexible schema that can be easily extended. Hence, the aforementioned metadata (e.g., extracted from question-answer websites and bug tracking systems) and new metadata, which supplements the comprehension of Git repositories, can be easily integrated into existing graphs.

## V. RELATED WORK

**Understanding and Visualizing Git Repositories.** Some essential and useful tools have been developed to summarise and visualise a Git repository. Focusing on the contribution of each developer, Gitential [1] helps to get valuable insights from a team and developers by analysing Git repos and coding hours, etc. GitVis3D [3], a visualisation tool for git communities, visualises a versioning graph as 3D animation. Furthermore, GitUp [2], a new Git interaction model, provides a layer on top of a reusable generic Git toolkit called GitUpKit to help developers build their own Git UI.

**Application of Knowledge Graph.** In recent years, researchers have demonstrated that knowledge graphs are useful for various purposes such as constructing question-answering websites [15], [16], [18], [20], [21], representing API caveats from the API documentation [17], performing semantic rankings [19], etc. Our work is the first attempt to apply knowledge graph to understand Git repositories.

## VI. CONCLUSION

This paper presents our preliminary work on knowledge graphing Git repositories, which provides an interactive graphical representation that is expected to help developers and project managers better comprehend Git projects. Moreover, as experimentally demonstrated in this work, the automatically generated knowledge graph is also useful for supporting advanced Git queries such as observing the top changed buggy methods, and supporting mining software repository studies.

## ACKNOWLEDGEMENT

This work is supported by the National Key Research and Development Program of China (grant No.2017YFB0801903), the National Natural Science Foundation of China (grants No.61702045) and the National Key Program for Basic Research of China (grant No.2017-JCJQ-ZD-043).

## REFERENCES

[1] Gitential. https://gitential.com/.
[2] GitUp. https://gitup.co/.
[3] GitVis3D. https://github.com/kofujimura/gitVis3D.
[4] BookWorm, 2011. https://github.com/charlieCollins/and-bookworm.
[5] Yahoonewsonboarding, 2016. https://github.com/rahulrj/YahooNewsOnboarding.
[6] AFWall+, 2018. https://github.com/ukanth/afwall.
[7] Calendula, 2018. https://github.com/citiususc/calendula.
[8] Flavordex, 2018. https://github.com/ultramega/flavordex.
[9] Kolab Notes, 2018. https://github.com/konradrenner/kolabnotes-android.
[10] MaterialFBook, 2018. https://github.com/ZeeRooo/MaterialFBook.
[11] Network Monitor, 2018. https://github.com/caarmen/network-monitor.
[12] Padland, 2018. https://github.com/mikifus/padland.
[13] SuntimesWidge, 2018. https://github.com/forrestguice/SuntimesWidget.
[14] Uber Rides Android SDK, 2018. https://github.com/uber/rides-android-sdk.
[15] Abdalghani Abujabal, Mohamed Yahya, Mirek Riedewald, and Gerhard Weikum. Automated template generation for question answering over knowledge graphs. In *WWW 2017*, pages 1191–1200.
[16] Sutanay Choudhury, Khushbu Agarwal, Sumit Purohit, Baichuan Zhang, Meg Pirrung, Will Smith, and Mathew Thomas. Nous: Construction and querying of dynamic knowledge graphs. In *ICDE*, 2017.
[17] Hongwei Li, Sirui Li, Jiamou Sun, Zhenchang Xing, Xin Peng, Mingwei Liu, and Xuejiao Zhao. Improving api caveats accessibility by mining api caveats knowledge graph. In *ICSME 2018*.
[18] Guozhu Meng, Yinxing Xue, Jing Kai Siow, Ting Su, Annamalai Narayanan, and Yang Liu. Androvault: Constructing knowledge graph from millions of android apps for automated computing. *arXiv preprint arXiv:1711.07451*, 2017.
[19] Chenyan Xiong, Russell Power, and Jamie Callan. Explicit semantic ranking for academic search via knowledge graph embedding. In *WWW 2017*, pages 1271–1279, 2017.
[20] Yuyu Zhang, Hanjun Dai, Zornitsa Kozareva, Alexander J Smola, and Le Song. Variational reasoning for question answering with knowledge graph. *arXiv preprint arXiv:1709.04071*, 2017.
[21] Xuejiao Zhao, Zhenchang Xing, Muhammad Ashad Kabir, Naoya Sawada, Jing Li, and Shang-Wei Lin. Hdskg: Harvesting domain specific knowledge graph from content of webpages. In *SANER*, 2017.