A Graph-Based Intermediate Representation for Java Code and Developer Documentation

Anonymous Author(s)
Affiliation
Address
`email`

Abstract

Semantic information plays a key role in the code search and synthesis settings. In this work, we propose a graph-based representation for source code and natural language which incorporates semantic and relational features from both domains. We apply this graph to parsing a corpus of Java code and developer docs, and demonstrate the effectiveness of a common graph-based representation on three downstream tasks: code search, document recommendation and link prediction.

1 Background and motivation

In addition to its syntactic structure, source code contains a rich denotational and operational semantics [Henkel et al., 2018]. To effectively reason about code in semantically similar but syntactically diverse settings requires models which incorporate features from the call graph [Gu et al., 2016, Liu et al., 2019] and surrounding typing context [Allamanis et al., 2017]. Many semantic features, such as data and control flow [Si et al., 2018] can be represented as a directed acyclic graph (DAG), which admits linear-time solutions to a number of graph problems, including topological sorting, single-source shortest path and reachability queries.

Natural language also contains semantic information which can be represented using graphs. The NLP community has a rich set of graph based representations, including Reddy et al. [2016]'s and other typed attribute grammars which can be used to reason about syntactic and semantic relations. The pointer network architecture [Vinyals et al., 2015b,a] can be used to construct permutation-invariant semantic relations between natural language entities, and have important applications in dependency parsing [Ma et al., 2018], named-entity recognition [Lample et al., 2016], and other language modeling tasks where sequence-based representations fall short. Li et al. [2017] extend pointer networks with a copy-mechanism to handle out-of-vocabulary code tokens.

Content recommendation for doc-to-doc (D2D) and code-to-code (C2C) settings can be solved with existing link prediction [Zhang and Chen, 2018] and code embedding [Gu et al., 2018] techniques, but cross-domain transfer remains challenging. Robillard and Chhetri [2015], Robillard et al. [2017] first explore the task of predicting reference API docs from source code using manual annotation. Prior work also studies the association between comments and code entities [Panthaplackel et al., 2020] using machine learning, but only within source code.

Maintainers of widely-used software projects often publish web-based developer docs, typically stored in markup languages like HTML or Markdown. These files contain a collection of natural language sentences, markup, and hyperlinks to other documents. Both the link

Submitted to 34th Conference on Neural Information Processing Systems (NeurIPS 2020).

graph and the document tree contain important semantic information: the markup describes the text in relation to the other entities in the document hierarchy [Yang et al., 2016], while the link graph describes the relationship between the parent document and related documents or source code entities. Documents occasionally contain hyperlinks to source code, but source code rarely contains links to developer documents.

Some programming languages allow users to specify which type of values will inhabit a given variable at runtime. Types allow the compiler to reason about certain properties like nullity [Ekman and Hedin, 2007] and shape [Considine et al., 2019]. While types many not appear explicitly in source code, they can often be inferred from the surrounding context using a dataflow graph (DFG). The Java language recently introduced local variable type inference Liddell and Kim [2019], which allows variable types to be omitted, and later inferred by the compiler.

2 Proposed approach

Given a single token in either source code or developer documentation and its surrounding context, what are the most relevant source code or documentation entities related to the token? We would like to infer which entities are connected to a given token, based on the surrounding semantic context. Learning relationships between these two domains requires parsing both natural language and source code. Following Si et al. [2018], Gu et al. [2018], Liu et al. [2019], we use a node embedding on the dataflow graph and type environment, and following Yang et al. [2016], Zhang and Chen [2018], use the markup hierarchy and link graph to construct an embedding for code-like tokens used within documentation.

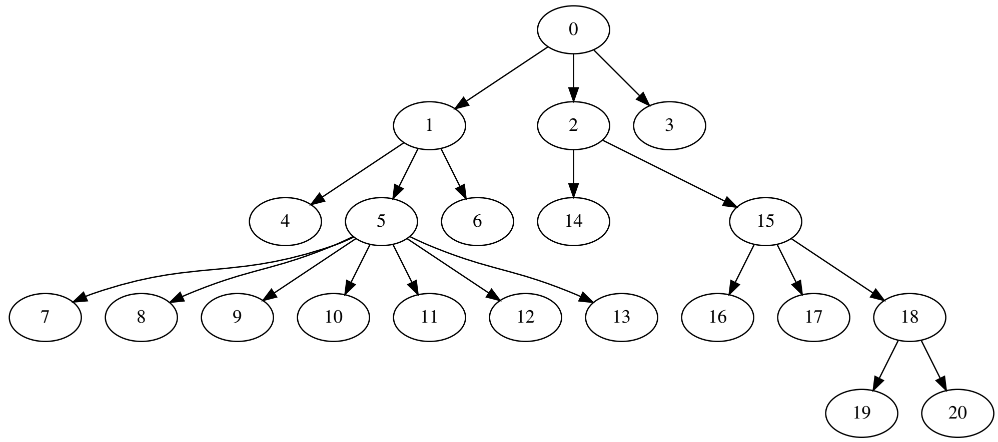
To compensate for the sparsity of hyperlinks between code and documentation, we need to construct bridge links between the documentation graph and source code entities. One heuristic which developers often use to discover relevant documents is plaintext search on a salient lexical string. Co-occurrence of an uncommon token is unlikely to be coincidence, and indicates that two entities are likely related, even though they may not share an explicit grammatical link. If we can recover this relationship using semantic and type-related information without observing the lexical token itself, this is a further indication our representation is providing useful information.

3 Data availability

Java, one of the most prolific programming languages on GitHub, is a statically typed language with a high volume of API documentation. With has a variety of tools source code [Parr, 2013, Hosseini and Brusilovsky, 2013, Kovalenko et al., 2019] and natural language [Manning et al., 2014, Grella and Cangialosi, 2018] parsing, it is a convenient language for both analysis and implementation. Our dataset consists of Java repositories on GitHub, and their accompanying developer documents. All projects in our dataset have a collection of source code files and multiple related repositories on GitHub.

We construct two datasets consisting of naturally-occurring links between developer docs and source code, and a surrogate set of links constructed by matching lexical tokens available in both domains. Our target is recovery of ground truth links in the test set and surrogate links in the lexical matching graph. By adding weighted edges between source code and docs, we evaluate our approach by predicting synthetic links between tokens contained in code fragments and markup entities which refer to the selected token. In addition, we evaluate our approach on both D2D and C2C link retrieval, as well as precision and recall on the surrogate link relations.

Our data consists of two complementary datasets: abstract syntax trees collected from Java source code and developer documentation. We use the astminer [Kovalenko et al., 2019] library to parse Java code, jsoup [Hedley, 2009] to parse HTML and Stanford's CoreNLP [Manning et al., 2014] library to parse dependency graphs from developer docs. Consider the following AST, parsed from the Eclipse Collections Java project:



The AST depicted above was generated by parsing the following code snippet:

```java
public void lastKey_throws() {
    new ImmutableTreeMap<>(new TreeSortedMap<>()).lastKey();
}
```

Now consider the following dependency graph, taken from a Javadoc in the same project:

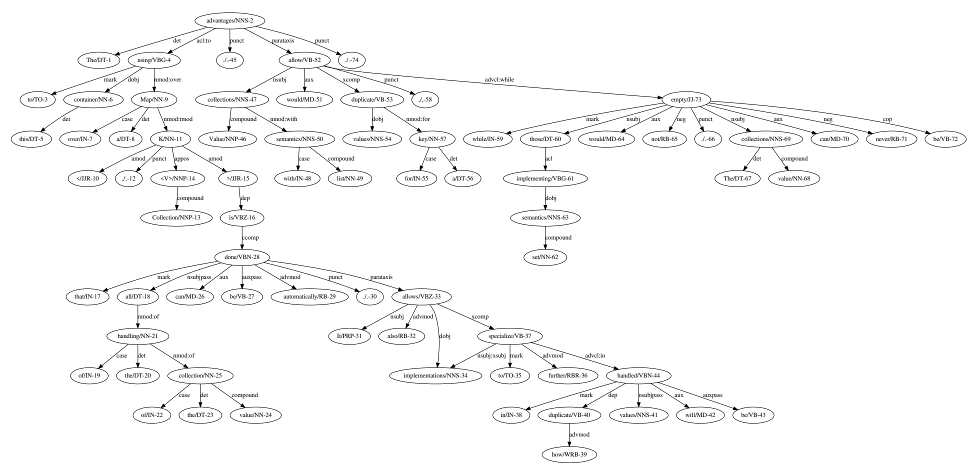


Figure 1: This graph was parsed from the following comment: "The advantages to using this container over a `Map<K, Collection<V>>` is that all of the handling of the value collection can be done automatically. It also allows implementations to further specialize in how duplicate values will be handled. Value collections with list semantics would allow duplicate values for a key, while those implementing set semantics would not. The value collections can never be empty."

Our goal is to connect these two graphs using a common representation for source code and natural language. Absent any explicit `@link` or `@see` annotations, in order to relate these two graphs, we must somehow infer the shared semantic entities, which we can do for a subset using a simple lexical matching procedure.

4 Experiment

We then train four link prediction models on the following datasets:

- Code graph (CG)
- Documentation graph (DG)
- Both code and documentation graphs separately (CDG)

- Single graph containing code and documentation entities related by lexical matching (LEX)

For each model, we evaluate our trained model on link prediction in the same setting and every other setting.

References

Miltiadis Allamanis, Marc Brockschmidt, and Mahmoud Khademi. Learning to represent programs with graphs. *arXiv preprint arXiv:1711.00740*, 2017. URL `https://arxiv.org/pdf/1711.00740.pdf`.

Breandan Considine, Michalis Famelis, and Liam Paull. Kotlin∇: A shape-safe eDSL for differentiable programming. `https://github.com/breandan/kotlingrad`, 2019.

Torbjörn Ekman and Görel Hedin. Pluggable checking and inferencing of nonnull types for Java. *Journal of Object Technology*, 6(9):455–475, 2007. URL `http://www.jot.fm/issues/issue_2007_10/paper23.pdf`.

Matteo Grella and Simone Cangialosi. Non-projective dependency parsing via latent heads representation LHR. *arXiv preprint arXiv:1802.02116*, 2018. URL `https://arxiv.org/pdf/1802.02116.pdf`.

Xiaodong Gu, Hongyu Zhang, Dongmei Zhang, and Sunghun Kim. Deep API learning. In *Proceedings of the 2016 24th ACM SIGSOFT International Symposium on Foundations of Software Engineering*, pages 631–642, 2016. URL `https://arxiv.org/pdf/1605.08535.pdf`.

Xiaodong Gu, Hongyu Zhang, and Sunghun Kim. Deep code search. In *2018 IEEE/ACM 40th International Conference on Software Engineering (ICSE)*, pages 933–944. IEEE, 2018. URL `https://guxd.github.io/papers/deepcs.pdf`.

Jonathan Hedley. jsoup: Java HTML parser. 2009. URL `https://jsoup.org`.

Jordan Henkel, Shuvendu K Lahiri, Ben Liblit, and Thomas Reps. Code vectors: Understanding programs through embedded abstracted symbolic traces. In *Proceedings of the 2018 26th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering*, pages 163–174, 2018. URL `https://arxiv.org/pdf/1803.06686.pdf`.

Roya Hosseini and Peter Brusilovsky. JavaParser: A fine-grain concept indexing tool for Java problems. In *CEUR Workshop Proceedings*, volume 1009, pages 60–63. University of Pittsburgh, 2013.

Vladimir Kovalenko, Egor Bogomolov, Timofey Bryksin, and Alberto Bacchelli. PathMiner: a library for mining of path-based representations of code. In *Proceedings of the 16th International Conference on Mining Software Repositories*, pages 13–17. IEEE Press, 2019. URL `https://github.com/JetBrains-Research/astminer`.

Guillaume Lample, Miguel Ballesteros, Sandeep Subramanian, Kazuya Kawakami, and Chris Dyer. Neural architectures for named entity recognition. *arXiv preprint arXiv:1603.01360*, 2016. URL `https://arxiv.org/pdf/1603.01360.pdf`.

Jian Li, Yue Wang, Michael R Lyu, and Irwin King. Code completion with neural attention and pointer networks. *arXiv preprint arXiv:1711.09573*, 2017. URL `https://www.ijcai.org/Proceedings/2018/0578.pdf`.

Clayton Liddell and Donghoon Kim. Analyzing the adoption rate of local variable type inference in open-source Java 10 projects. *Journal of the Arkansas Academy of Science*, 73(1):51–54, 2019. URL `https://scholarworks.uark.edu/cgi/viewcontent.cgi?article=3346&context=jaas`.

Bohong Liu, Tao Wang, Xunhui Zhang, Qiang Fan, Gang Yin, and Jinsheng Deng. A neural-network based code summarization approach by using source code and its call dependencies. In *Proceedings of the 11th Asia-Pacific Symposium on Internetware*, Internetware '19, New York, NY, USA, 2019. Association for Computing Machinery. ISBN 9781450377010. doi: 10.1145/3361242.3362774. URL `https://doi.org/10.1145/3361242.3362774`.

Xuezhe Ma, Zecong Hu, Jingzhou Liu, Nanyun Peng, Graham Neubig, and Eduard Hovy. Stack-pointer networks for dependency parsing. *arXiv preprint arXiv:1805.01087*, 2018. URL https://arxiv.org/pdf/1805.01087.pdf.

Christopher D Manning, Mihai Surdeanu, John Bauer, Jenny Rose Finkel, Steven Bethard, and David McClosky. The stanford coreNLP natural language processing toolkit. In *Proceedings of 52nd annual meeting of the association for computational linguistics: system demonstrations*, pages 55–60, 2014. URL https://nlp.stanford.edu/pubs/StanfordCoreNlp2014.pdf.

Sheena Panthaplackel, Milos Gligoric, Raymond J. Mooney, and Junyi Jessy Li. Associating natural language comment and source code entities. In *AAAI*, 2020. URL https://arxiv.org/pdf/1912.06728.pdf.

Terence Parr. *The definitive ANTLR 4 reference*. Pragmatic Bookshelf, 2013.

Siva Reddy, Oscar Täckström, Michael Collins, Tom Kwiatkowski, Dipanjan Das, Mark Steedman, and Mirella Lapata. Transforming dependency structures to logical forms for semantic parsing. *Transactions of the Association for Computational Linguistics*, 4: 127–140, 2016. URL https://www.mitpressjournals.org/doi/pdf/10.1162/tacl_a_00088.

Martin P Robillard and Yam B Chhetri. Recommending reference API documentation. *Empirical Software Engineering*, 20(6):1558–1586, 2015. URL https://www.cs.mcgill.ca/~martin/papers/cr2014a.pdf.

Martin P Robillard, Andrian Marcus, Christoph Treude, Gabriele Bavota, Oscar Chaparro, Neil Ernst, Marco Aurélio Gerosa, Michael Godfrey, Michele Lanza, Mario Linares-Vásquez, et al. On-demand developer documentation. In *2017 IEEE International Conference on Software Maintenance and Evolution (ICSME)*, pages 479–483. IEEE, 2017.

Xujie Si, Hanjun Dai, Mukund Raghothaman, Mayur Naik, and Le Song. Learning loop invariants for program verification. In *Advances in Neural Information Processing Systems*, pages 7751–7762, 2018. URL https://papers.nips.cc/paper/8001-learning-loop-invariants-for-program-verification.pdf.

Oriol Vinyals, Samy Bengio, and Manjunath Kudlur. Order matters: Sequence to sequence for sets. *arXiv preprint arXiv:1511.06391*, 2015a. URL https://arxiv.org/pdf/1511.06391.pdf.

Oriol Vinyals, Meire Fortunato, and Navdeep Jaitly. Pointer networks. In *Advances in neural information processing systems*, pages 2692–2700, 2015b. URL https://arxiv.org/pdf/1506.03134.pdf.

Zichao Yang, Diyi Yang, Chris Dyer, Xiaodong He, Alex Smola, and Eduard Hovy. Hierarchical attention networks for document classification. In *Proceedings of the 2016 conference of the North American chapter of the association for computational linguistics: human language technologies*, pages 1480–1489, 2016. URL https://www.cs.cmu.edu/~./hovy/papers/16HLT-hierarchical-attention-networks.pdf.

Muhan Zhang and Yixin Chen. Link prediction based on graph neural networks. In *Advances in Neural Information Processing Systems*, pages 5165–5175, 2018. URL https://papers.nips.cc/paper/7763-link-prediction-based-on-graph-neural-networks.pdf.