Navigating and Exploring Software Dependency Graphs using Goblin

Damien Jaime, Joyce El Haddad, Pascal Poizat

Navigating and Exploring Software Dependency Graphs using Goblin

Damien Jaime
Sorbonne Université, CNRS, LIP6, Université Paris Nanterre
F-75005, Paris, France
damien.jaime@lip6.fr

Joyce El Haddad
Université Paris Dauphine-PSL, CNRS, LAMSADE
CF-75016, Paris, France
joyce.elhaddad@lamsade.dauphine.fr

Pascal Poizat
Sorbonne Université, CNRS, LIP6, Université Paris Nanterre
F-75005, Paris, France
pascal.poizat@lip6.fr

***Abstract*—Using package managers is a simple and common method for reusing code through project dependencies. However, these, direct, dependencies can themselves rely on additional packages, resulting in indirect dependencies. It may then become complex to get a grasp of the whole set of dependencies of a project. Beyond studying individual projects, a deep understanding of software ecosystems is also a critical prerequisite for achieving sustained success in software development. This paper presents the 2025 edition of the MSR conference mining challenge. This year's mining challenge focuses on dependencies and dependency ecosystem analysis using the Goblin framework that has been presented at the previous edition of the MSR conference. Goblin is composed of a Neo4j Maven Central dependency graph and a tool called Weaver for on-demand metric weaving into dependency graphs. As a whole, Goblin is a customizable framework for ecosystem and dependency analysis.**

***Index Terms*—software ecosystem, dependency graph, dataset, mining software repositories, maven central**

I. Dataset and tooling

The Goblin framework (see Figure 1) is organized around a Neo4j database of the whole Maven Central dependency graph. This database can be created and updated incrementally using Goblin Miner. The database can be queried directly using Cypher (the Neo4j query language) or through the Goblin Weaver tool. More generally, Goblin Weaver comes with an API for the on-demand weaving of user-programmed metrics of interest (added values) into the dependency graph. Several such metrics are already available: CVEs, popularity, freshness and release speed. As a whole, Goblin aims to be a customizable framework for working on software dependencies at the ecosystem level.

A. Neo4j Maven Central ecosystem dependency graph

Figure 2 shows the structure of the dependency graph database.

This database is mostly composed of two node types (for libraries, also called artifacts, and for their releases) and two edge types (from releases to their dependencies, and from libraries to their releases). The nodes for libraries (type Artifact) contain the Maven id (g.a with g the group id, and a the artifact id) and a boolean found. This found boolean is used to know whether the library has been found in the ecosystem or not. This allows to provide dependency information even if the library is not found on Maven Central. The nodes for releases (type Release) contain the Maven id (g.a.v with g the group id, a the artifact id, and v the version), the release timestamp, and the version information. The edges for dependencies (type dependency) are from Release nodes to Artifact nodes and contain target version (which can be a range) and scope (compile, test, etc). The edges for versioning

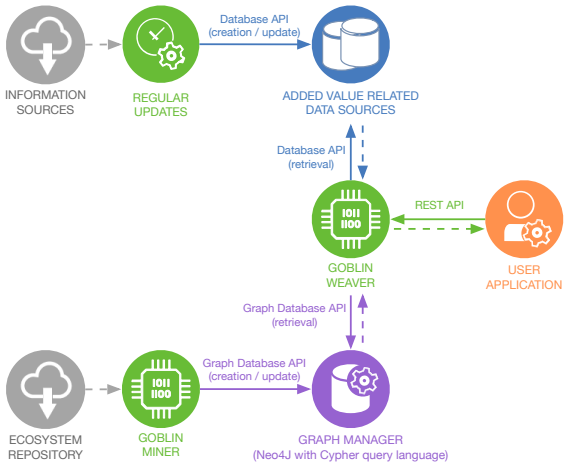

Fig. 1. Goblin framework architecture.

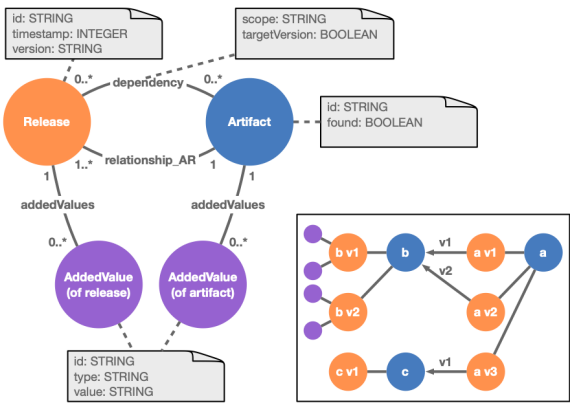

Fig. 2. Ecosystem dependency graph model and illustrative example.

This work is funded by PhD grant 2021/0047 from ANRT.

TABLE I
DATASET STATISTICS

Description	Dataset	Enriched dataset
Nodes	15,117,217	59,152,712
libraries	658,078	658,078
releases	14,459,139	14,459,139
added value	n/a	44,035,495
Edges	134,119,545	178,155,040
dependencies	119,660,406	119,660,406
versioning	14,459,139	14,459,139
added values	n/a	44,035,495

(type relationship_AR) are from Artifact nodes to Release nodes.

The latest version of our dataset, dated August 30$^{\text{th}}$, 2024, has statistics given in Table I. We also provide a second version of this dataset enriched with Weaver-computed metrics, which (see below in Section I-C) has the effect of creating new AddedValue nodes in the database for CVE (dated September 4$^{\text{th}}$, 2024), freshness, popularity, and speed.

B. Goblin Miner

The Goblin Miner allows one to update the dependency graph database or recreate it from scratch. First, it retrieves all releases in the Lucene Maven Central Index archive,[1] extracts the data from it, and creates the library and the release nodes, together with the version edges, in a Neo4j database. Then, it goes through all releases to retrieve their direct dependencies with the `org.eclipse.aether` library. Information on create and update computation times are given in [1].

C. Goblin Weaver

The Goblin Weaver REST API is available as an alternative to the direct access to the database using the Cypher language, and for on-demand enrichment of the dependency graph with new information. A memoization principle has been implemanted to avoid re-computing enrichments, as soon as the base graph itself is not re-computed or updated. For this, new kinds of nodes (type AddedValue) and edges (type addedValues from an Artifact or Release node to an AddedValue node) are used in the graph database. One should be careful, as the graph is large, computing metrics (especially aggregate ones) for the whole graph can be time-consuming. When the Weaver computes a metric, it adds a new node type AddedValue with edge type addedValues to the graph. Consequently, once a metric has been computed, it becomes directly accessible in the graph, and the Weaver is no longer required for its retrieval.

Added values are assigned to a certain type of node (Artifact or Release). Currently, the Weaver API can be used to compute and add the following metrics.

1) Release nodes added values: The added values of a release can be computed either locally (*e.g.*, the CVEs of a release) or aggregated (*e.g.*, the CVEs of a release and of all its direct and indirect dependencies). To use the aggregated value of a metric, add _AGGREGATE after the metric name. For example, to use the aggregated value of the CVE metric, use CVE_AGGREGATE.

CVE: Common Vulnerabilities and Exposures,[2] is a dictionary of public information on security vulnerabilities. We use the osv.dev[3] dataset to get CVE information. Our added value contains for each CVE its name, its CWE (type of vulnerability), and its severity (low, moderate, high, critical).

FRESHNESS: This corresponds, for a specific release, to the number of more recent releases available and to the time elapsed in milliseconds between it and the most recent release. More information about freshness is given in [2].

POPULARITY_1_YEAR: To compute the popularity of a release, we compute the number of dependants of the version of the library over a one year window (back from the date of the dependency graph). This corresponds for a release r to the question: "How many releases declared a dependency to r in the year before the dependency graph date?". There are many other ways to calculate the popularity of a release [3], and you can extend the Weaver to create your own, or modify the one-year window we have defined.

2) Artifact nodes added values: For now, only one metric is available for library nodes.

SPEED: This corresponds to the average number of releases per day of a library. More information is given in [4].

II. POTENTIAL RESEARCH QUESTIONS

The analysis of a software ecosystem graph presents numerous research opportunities, allowing for the investigation of various questions in areas such as structural analysis, community detection, dependency optimization, and risk assessment.

The following suggested questions outline potential research directions. Questions in groups 1 to 5 can be addressed with the dependency graph database alone, questions in group 6 require the additional use of the Weaver (or the enriched dataset), and questions in group 7 illustrate examples of inquiries that would require an extension of the Weaver.

1) Ecosystem evolution
   a) What are the patterns in the growth of the Maven Central graph across different time periods?
   b) Do artifacts tend to use more dependencies than in the past?
   c) Is the rhythm of library release higher than in the past, and how has this rhythm evolved over time?
   d) Does the emergence of project management methods (*e.g.*, agile methods) have any impact on the release rhythm of artifacts?
   e) To what extent does the ecosystem contain unmaintained artifacts?
   f) How do projects with unmaintained dependencies cope with the challenges they face?
2) Clustering

[1] https://maven.apache.org/repository/central-index.html

[2] https://cve.mitre.org/

[3] https://osv.dev/

a) Can we deduce different clusters from Maven Central's complete dependency graph? How do these clusters interact with one another?
b) Can dependency-based clustering reveal domain-specific groupings, and how well do they align with known categorizations of projects?
c) How can clustering be used to identify high-risk clusters in the Maven Central ecosystem?
d) Which artifacts serve as the most crucial dependencies for the ecosystem (*i.e.*, most depended upon)?
e) How do these central nodes affect the overall health and stability of the ecosystem?

3) Dependency update
a) How often do projects update their dependencies, and what factors influence this frequency (*e.g.*, project size, popularity, type)?
b) When an artifact releases a new version, how do its dependents react?
c) How does the removal or failure of certain projects affect the overall network (*e.g.*, log4j Vulnerability)?
d) How do major versus minor dependency updates differ in frequency and impact?
e) Do projects tend to avoid major updates due to the potential for breaking changes?

4) Trends
a) Has (and how) the adoption of new frameworks (*e.g.*, Spring Boot, Microservices) changed the dependency structures in Maven Central?
b) What impact do modern dependency management tools (*e.g.*, Dependabot) have on the ecosystem?
c) How does the adoption of newer Java versions influence dependency graphs?
d) Does an artifact's number of dependents correlate with other popularity metrics such as GitHub stars?

5) Graph theory
a) How do metrics such as degree distribution, clustering coefficient, and average path length characterize the dependency graph?
b) Is the graph scale-free, small-world, or does it exhibit other known graph structures?
c) Are certain types of projects more likely to be central (hubs) or peripheral (leaves) in the graph structure?
d) Is the graph made up of connected components with no relationship between them?
e) How do shortest path lengths between projects vary, and what does this tell us about the overall connectivity of the ecosystem?

6) Vulnerability
a) How do vulnerabilities propagate through the dependency network, and which projects are most affected?
b) What proportion of releases have vulnerabilities? What is the proportion of releases directly and transitively impacted?
c) What is the average time taken to patch a vulnerability in a dependency?
d) How do users of an artifact react to the discovery of a vulnerability in that artifact?

7) Licensing and Compliance
a) Are there dominant license types, and how do they influence the usage and distribution of projects?
b) How does the choice of licenses affect the artifact graph structure?
c) What percentage of projects have conflicting licenses within their dependency trees?

III. Ressources

The Maven Central Neo4j dependency graph datasets are available on Zenodo.[4] To import a database dump into Neo4j, please set your project database system version to 4.x. The Goblin Weaver project is available on GitHub,[5] as for the Goblin Miner.[6] A basic containerized project integrating both the Neo4j database and the Goblin Weaver is also available at GitHub.[7] A tutorial dedicated to the dataset and set of tools is available online.[8] Questions are welcomed using the issue tracking system. A more in-depth presentation of our framework and related work is given in [1]. An example of using this framework for automatic dependency update is presented in [5].

References

[1] D. Jaime, J. El Haddad, and P. Poizat, "Goblin: A framework for enriching and querying the maven central dependency graph," in *21st International Conference on Mining Software Repositories (MSR)*, 2024.

[2] J. Cox, E. Bouwers, M. C. J. D. van Eekelen, and J. Visser, "Measuring dependency freshness in software systems," in *37th International Conference on Software Engineering (ICSE)*, 2015.

[3] A. Zerouali, T. Mens, G. Robles, and J. M. González-Barahona, "On the diversity of software package popularity metrics: An empirical study of npm," in *26th International Conference on Software Analysis, Evolution and Reengineering (SANER)*, 2019.

[4] D. Jaime, J. El Haddad, and P. Poizat, "A preliminary study of rhythm and speed in the maven ecosystem," in *21st Belgium-Netherlands Software Evolution Workshop (BNEVOL)*, 2022.

[5] D. Jaime, P. Poizat, J. El Haddad, and T. Degueule, "Balancing the quality and cost of updating dependencies," in *Proceedings of the 39th IEEE/ACM International Conference on Automated Software Engineering (ASE)*, 2024.

[4] https://zenodo.org/records/13734581
[5] https://github.com/Goblin-Ecosystem/goblinWeaver
[6] https://github.com/Goblin-Ecosystem/goblinDependencyMiner
[7] https://github.com/Goblin-Ecosystem/Neo4jWeaverDocker
[8] https://github.com/Goblin-Ecosystem/goblinTutorial