Mining Software Dependencies at Large Scale

A Preliminary Study on the Maven Central Repository

Motivations

- Existence of massive repositories with millions of software artifacts (e.g., Maven Central (MC) > 3M artifacts)
- Scarce research have been made in order to study such repositories at a large scale

- Vulnerable dependencies are a known problem in today’s OSS ecosystems

Objectives

- Explore the global structure of MC
- Analyze the degree of interrelation between artifacts •

Determine which are the most influential artifacts

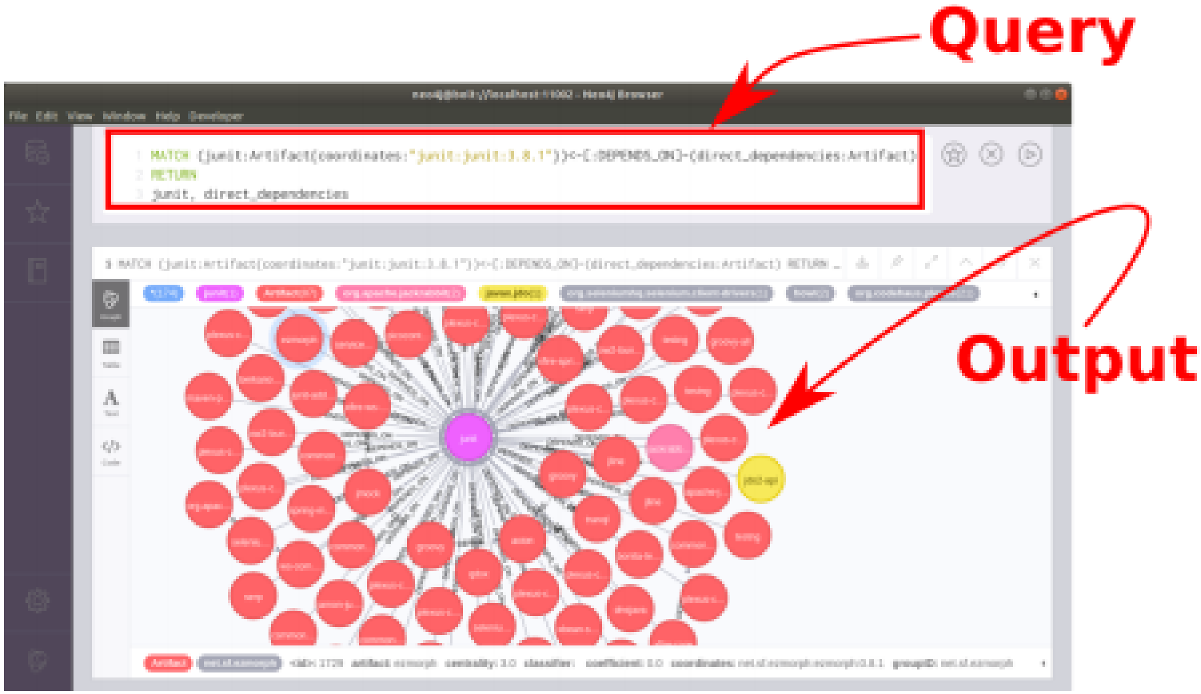
- Describe the historical evolution of popular OSS projects

Data collection

- Maven-miner (https://github.com/diverse-project/maven-miner) •

Neo4j (http://neo4j.com/)

- Cypher (https://neo4j.com/developer/cypher-query-language/)



Query

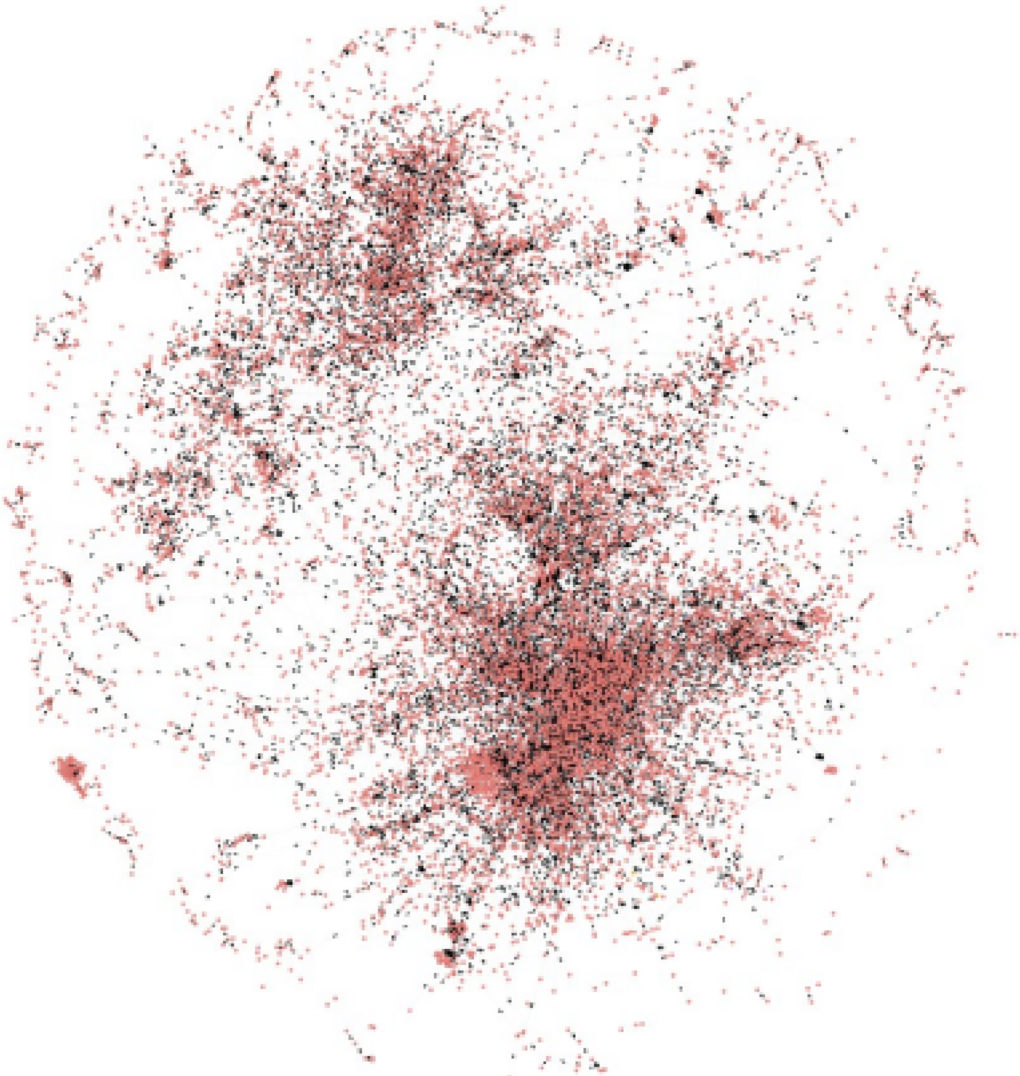
```
MATCH (junit:Artifact{coordinates:"junit:junit:3.8.1"})<-[:DEPENDS_ON]-(direct_dependencies:Artifact)
RETURN
junit, direct_dependencies
```

Output

```
$ MATCH (junit:Artifact{coordinates:"junit:junit:3.8.1"})<-[:DEPENDS_ON]-(direct_dependencies:Artifact) RETURN ...
```

```
Artifact  net.sf.ezmorph  <id>: 1729  artifact: ezmorph  centrality: 3.0  classifier:  coefficient: 0.0  coordinates: net.sf.ezmorph:ezmorph:0.8.1  groupID: net.sf.ezmorph
```

neo4j@bolt://localhost:11002 - Neo4j Browser
File Edit View Window Help Developer
junit



The “big picture”

- Only 1% of MC
- 31877 nodes
- 57227 edges

Key concepts

- B is a **dependency** of

A
- A **directly uses** B
- A **transitively uses** C

Descriptive statistics

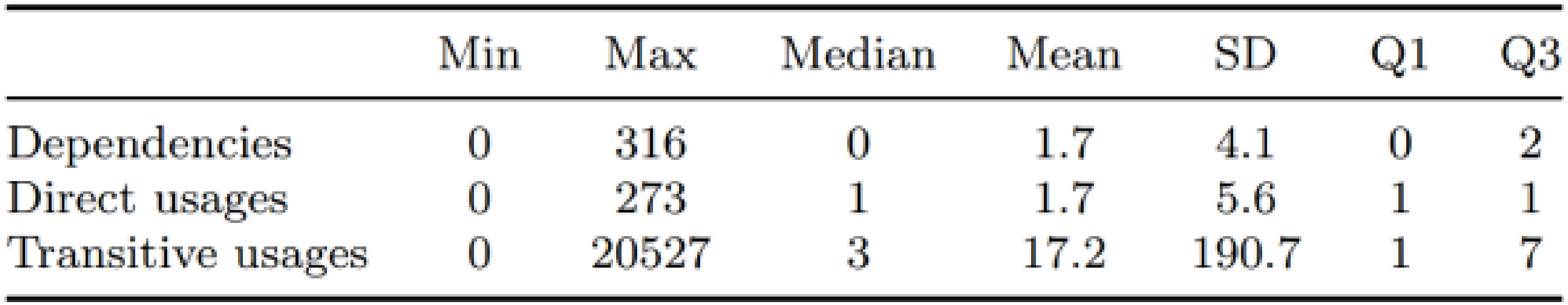
- According to the studied dataset:
  - Artifact with the Max # of dependencies: *org.jboss.as:jboss-as-build:7.1.2*

- Artifact with the Max # of direct usages:

  org.slf4j:slf4j-api:1.6.1

- Artifact with the Max # of transitive usages:

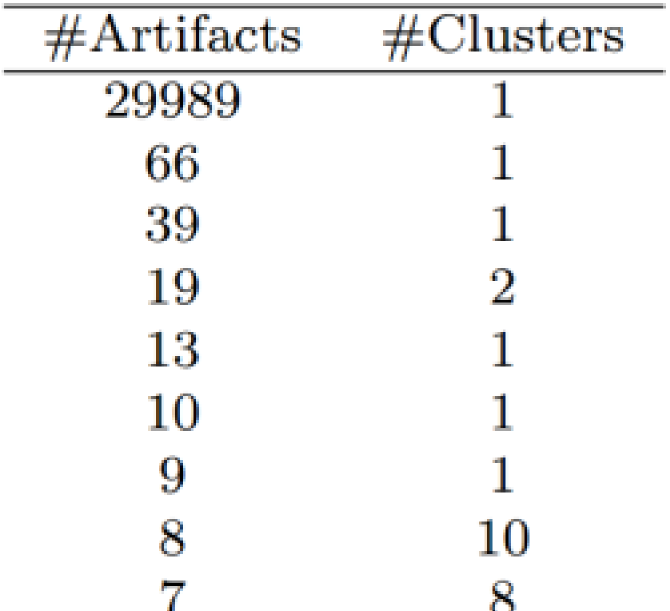
  commons-logging:commons-logging:1.1.1



	Min	Max	Median	Mean	SD	Q1	Q3
Dependencies	0	316	0	1.7	4.1	0	2
Direct usages	0	273	1	1.7	5.6	1	1
Transitive usages	0	20527	3	17.2	190.7	1	7

Connectivity

- Union Find algorithm



#Artifacts	#Clusters
29989	1
66	1
39	1
19	2
13	1
10	1
9	1
8	10
7	8

Cluster #1

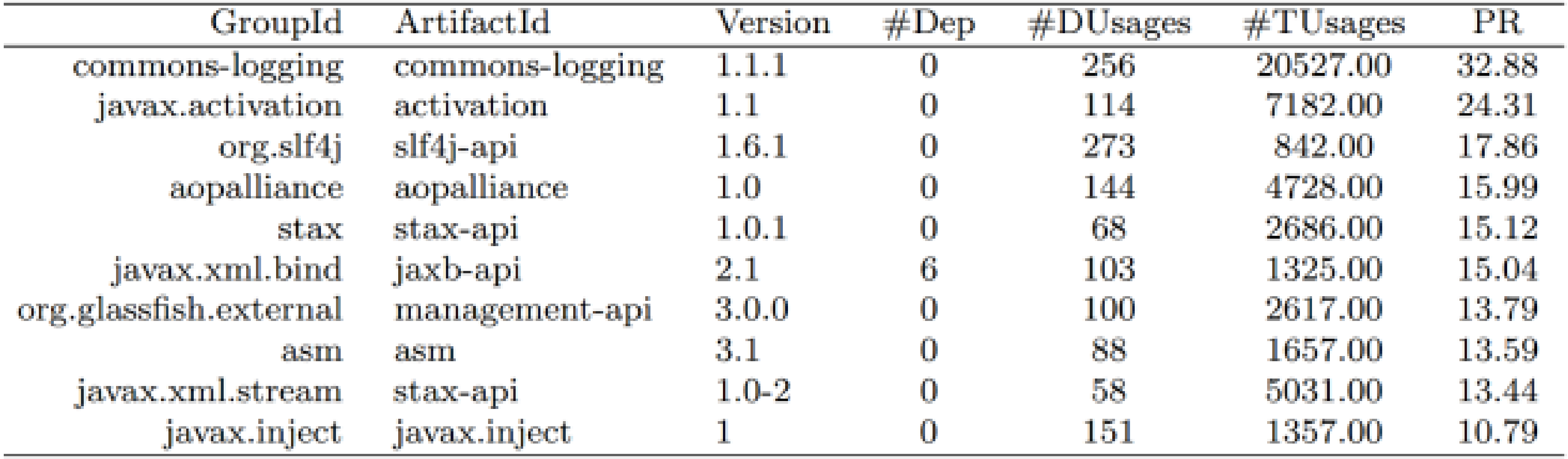
Cluster #2

Artifacts’ impact

It’s not only the number of dependencies what is important,

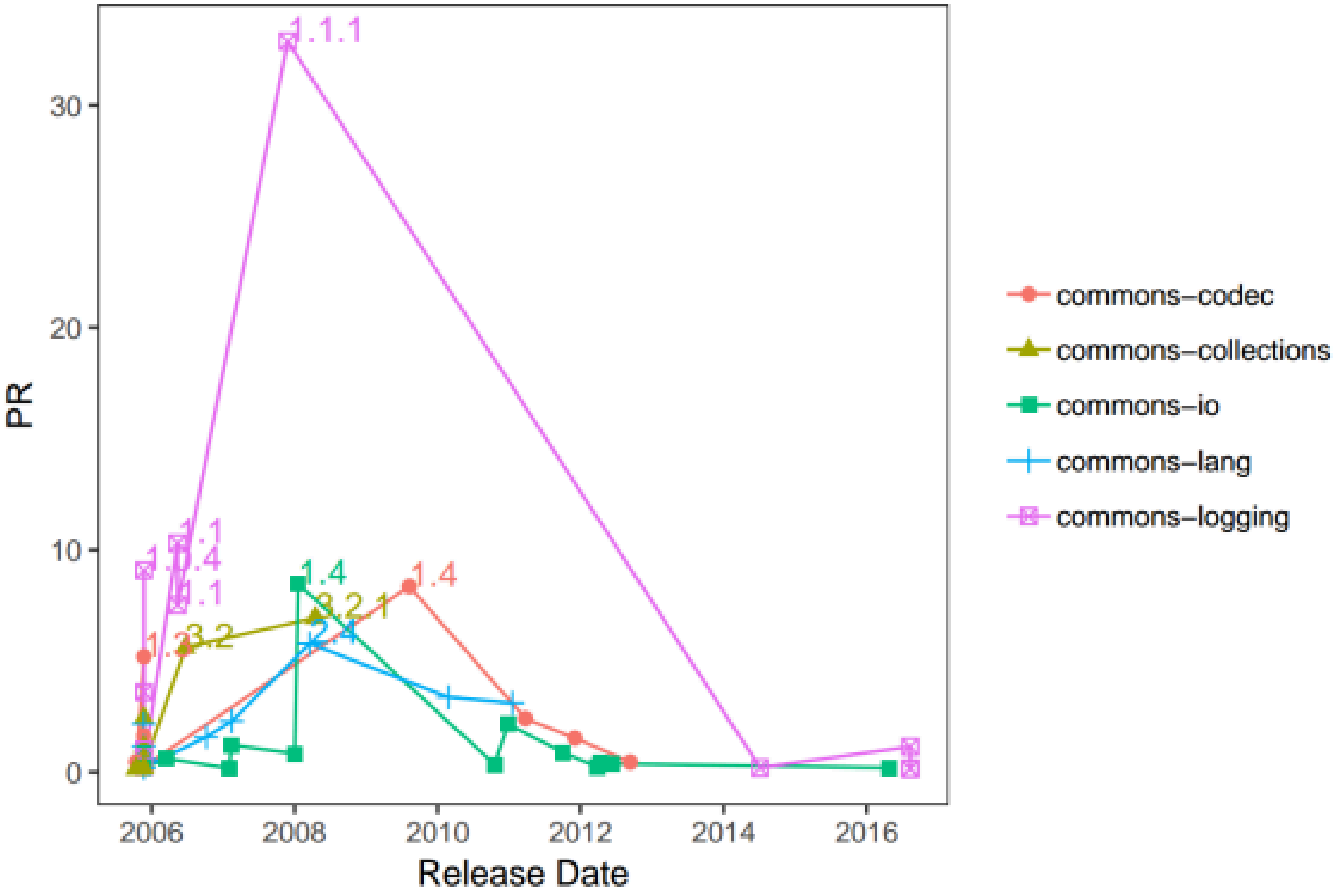
- **Page Rank** algorithm:
  but also the importance of the artifacts

behind those dependencies



GroupId	ArtifactId	Version	#Dep	#DUsages	#TUsages	PR
commons-logging	commons-logging	1.1.1	0	256	20527.00	32.88
javax.activation	activation	1.1	0	114	7182.00	24.31
org.slf4j	slf4j-api	1.6.1	0	273	842.00	17.86
aopalliance	aopalliance	1.0	0	144	4728.00	15.99
stax	stax-api	1.0.1	0	68	2686.00	15.12
javax.xml.bind	jaxb-api	2.1	6	103	1325.00	15.04
org.glassfish.external	management-api	3.0.0	0	100	2617.00	13.79
asm	asm	3.1	0	88	1657.00	13.59
javax.xml.stream	stax-api	1.0-2	0	58	5031.00	13.44
javax.inject	javax.inject	1	0	151	1357.00	10.79

Projects’ evolution



Conclusions

- A graph-based representation of MC brings new opportunities to perform large scale analyzes on software evolution and dependencies usage
- For the portion of data studied:
  - We found that the graph is nearly fully connected, with 94% of artifact belonging to a single large cluster
  - We identified the most influential artifacts through the use of different graph algorithms (*commons-logging:commonslogging:1.1.1*)

Future work

- Explore the full MC graph of artifacts and its dependencies

- Quantify how much of each dependency is actually used by each artifact (#classes? #methods?)
- Identify such parts in order to **introduce diversity** (e.g., to change a method for another with the same functionality but belonging to a different library)

KTH ROYAL INSTITUTE OF TECHNOLOGY

Mining Software Dependencies at Large Scale

A Preliminary Study on the Maven Central Repository