Generation of “probably” more likelihood models and data !

Adel Ferdjoukh

Eric Bourreau, Annie Chateau, Clémentine Nebut

Lirmm, Montpellier, France

SEKE, San Francisco, July 1st, 2016

Part I

Model Generation: a brief reminder

Synopsis

Context ? Model Driven Engineering

MDE

- Intensive use of models during software development process.
- A model is defined by (and conforms to) a meta-model.
- Meta-models come with other structural constraints written in OCL.
- Models are manipulated by programs called model transformations.

Problem ? Model Generation

Problem

- Are meta-models valid ? Do they really define correct models ?
- Are model transformations valid ? How should we test them ?

Solution

We need a model generation approach.

Problem ? Model Generation

Problem

- Are meta-models valid ? Do they really define correct models ?
- Are model transformations valid ? How should we test them ?

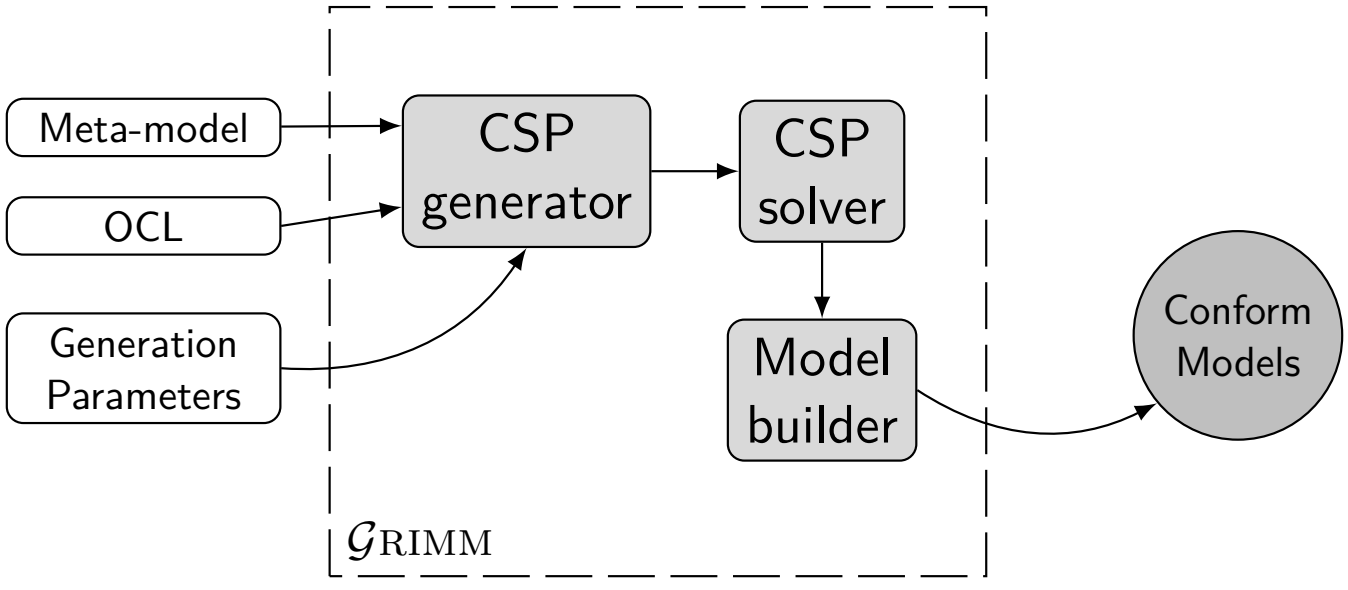
Solution

We need a model generation approach.

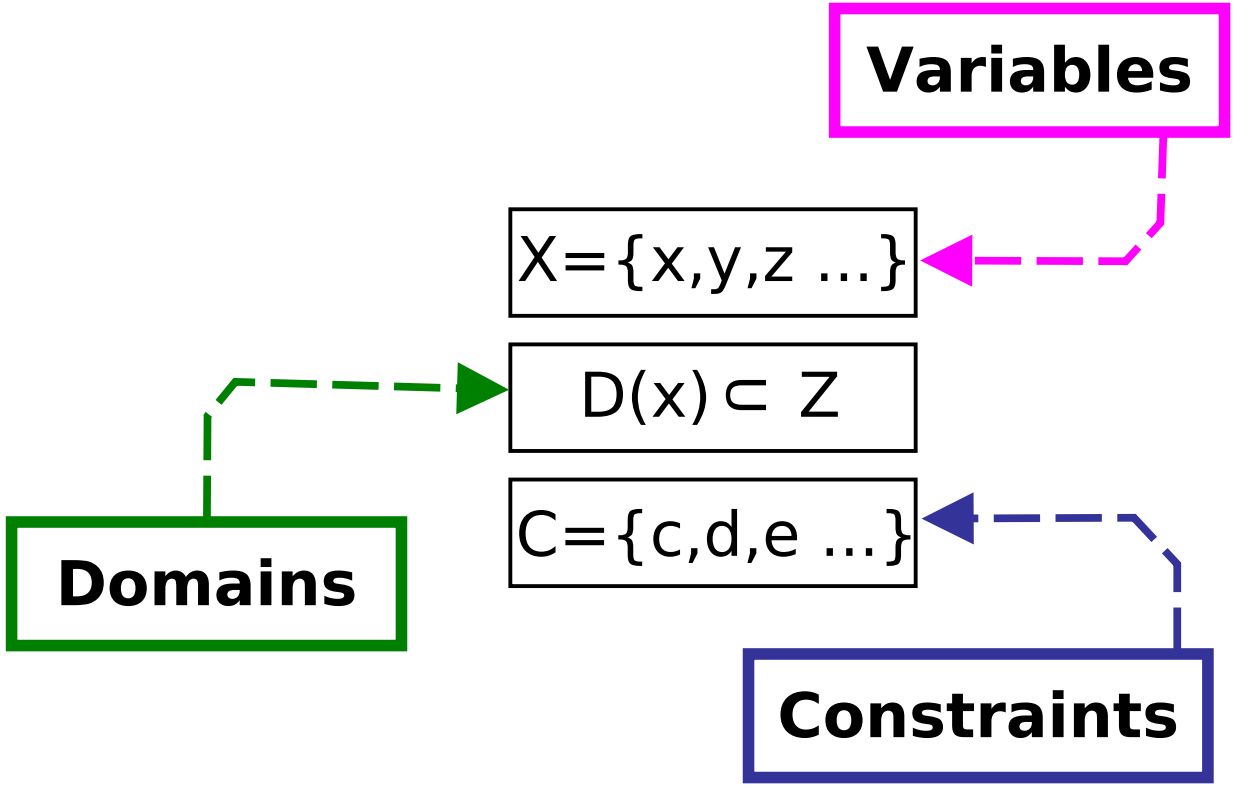
Model Generation: characteristics

- Automatisation
- Scalability
- Dealing with OCL
- Relevance
- Diversity

Model Generation: process



CSP: what is it ?



CSP: solver & solutions

CSP solver

A program that solves CSP.

Solution

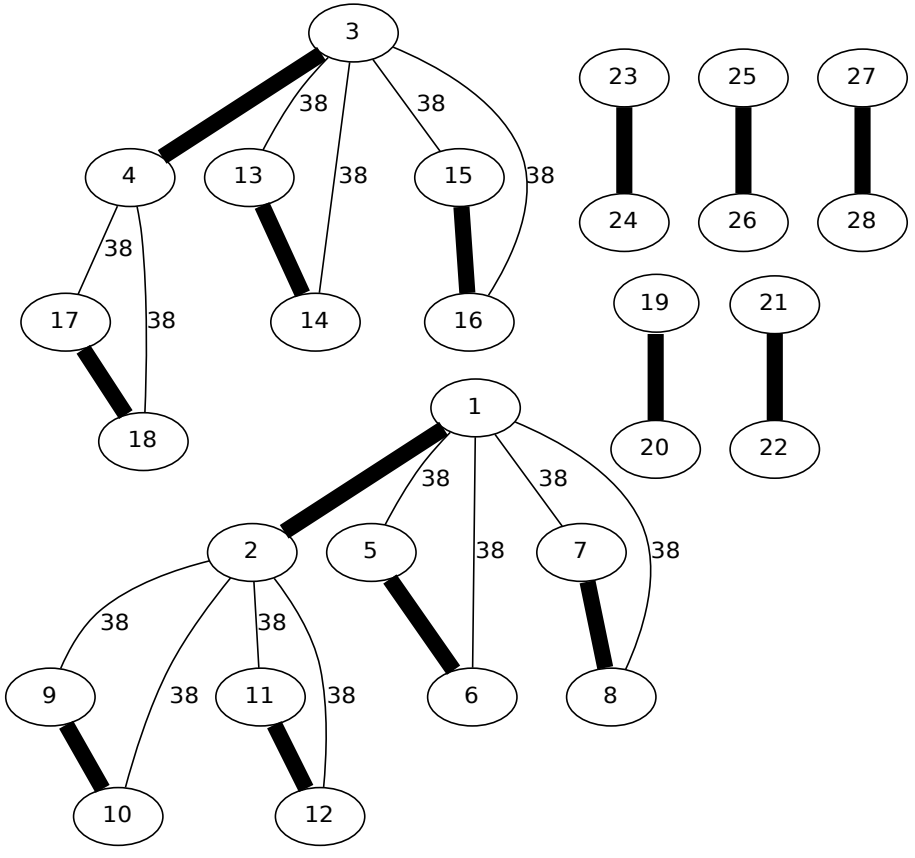
Assign values to all variables in order to satisfy all the constraints at the same time.

Part II

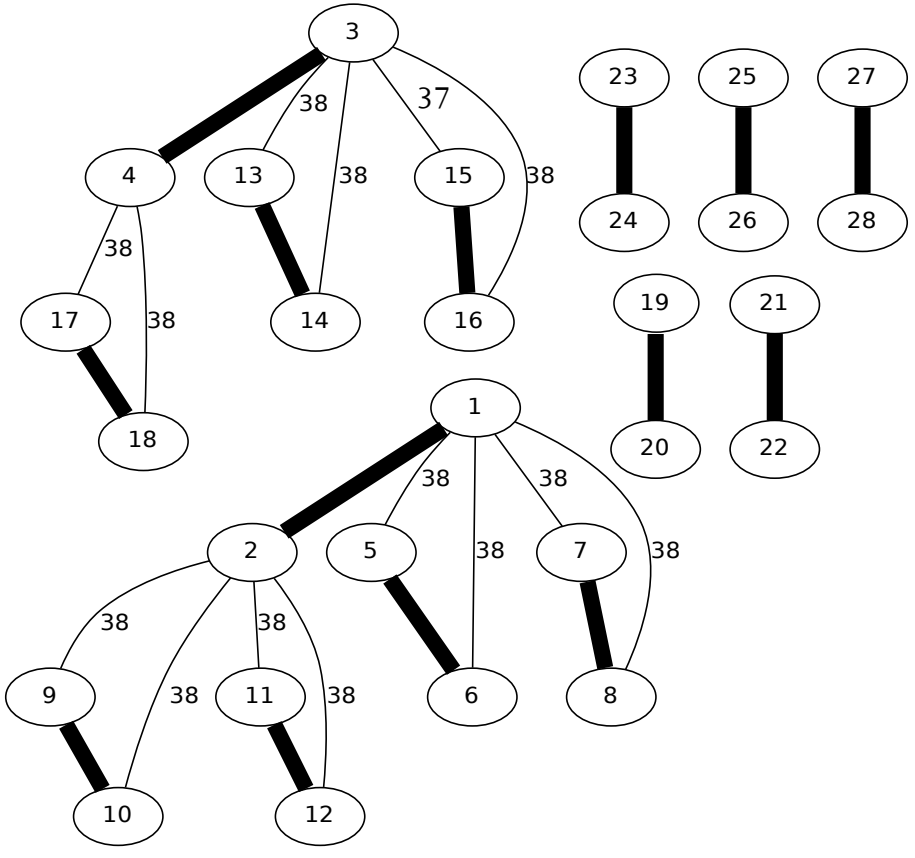
Generation of Relevant & Realistic Models

Synopsis

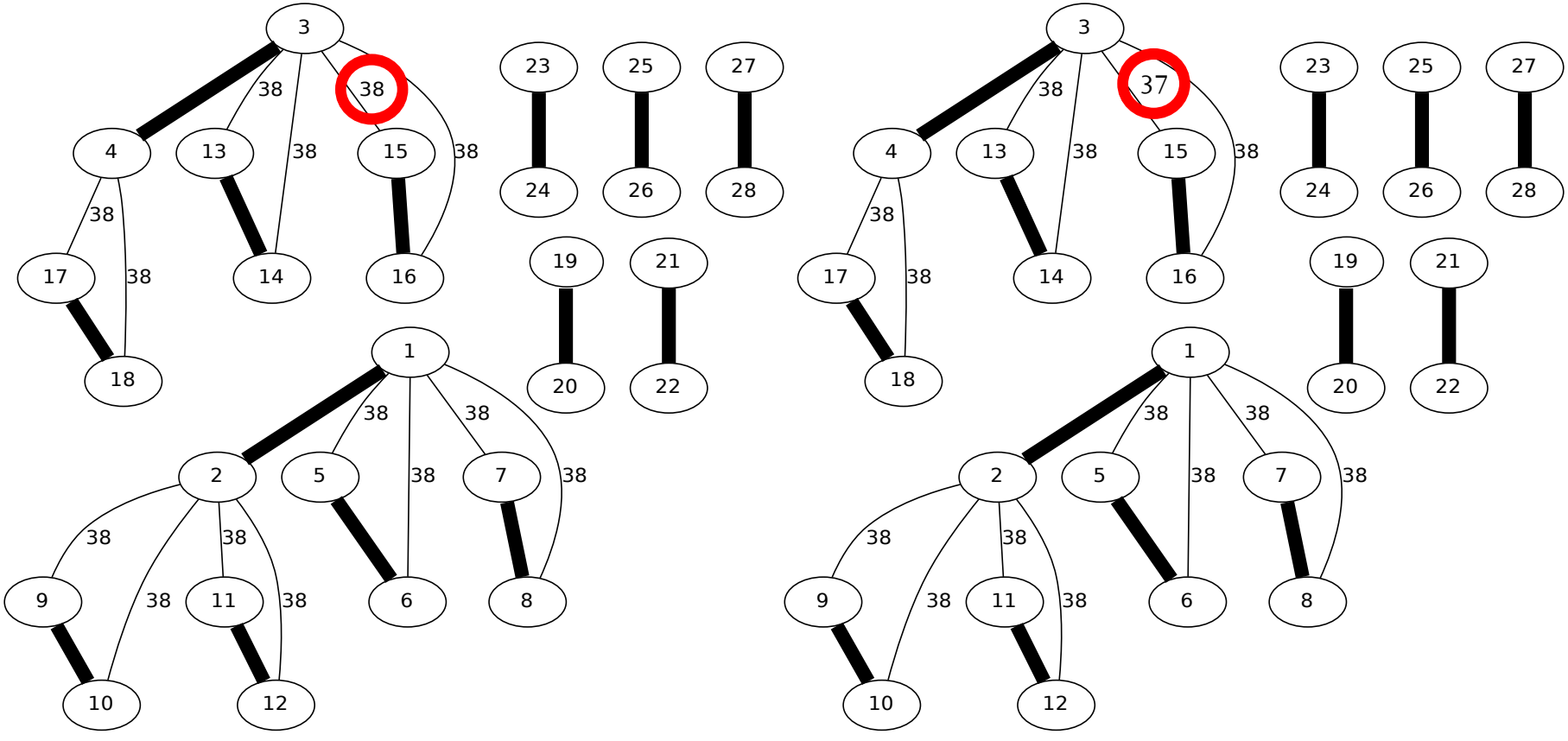
Pending problems



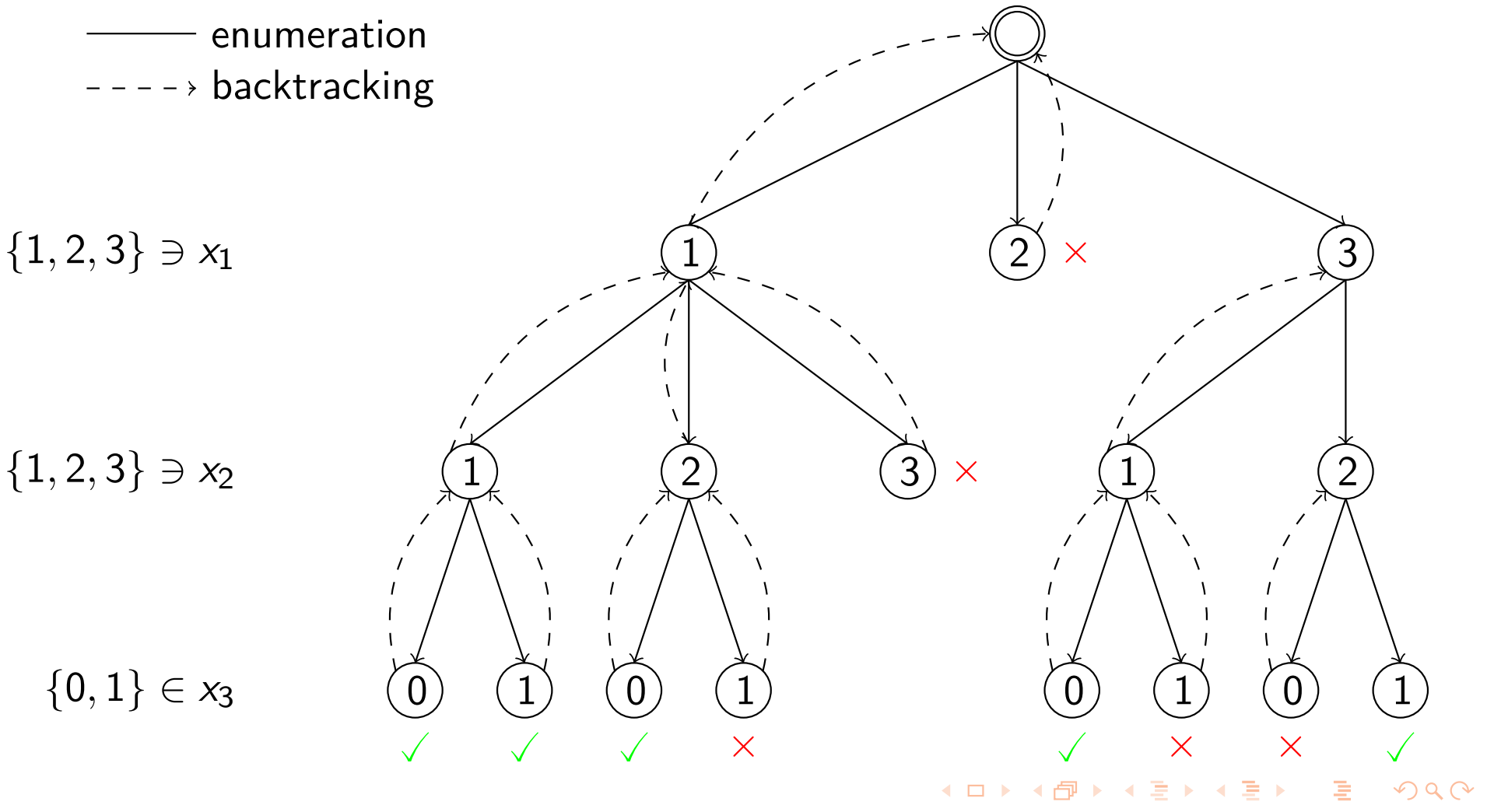
Pending problems



Pending problems



Pending problems



Pending problems

Observation

The generated are models are similar and ugly, *i.e.* useless.

Pending problems

Solution

We should improve their realism, relevance and internal diversity.

How ?

considered solutions

1. Go further in solver's solutions tree.

Analysis

1. Even the 1^{st} and the 100.000^{th} solutions are close to each.

How ?

considered solutions

1. Go further in solver's solutions tree.
2. Inject randomness inside solvers.

Analysis

1. Even the 1^{st} and the 100.000^{th} solutions are close to each.
2. Unfortunately, CSP is not designed for.

How ?

considered solutions

1. Go further in solver's solutions tree.
2. Inject randomness inside solvers.
3. Inject randomness *a priori*.

Analysis

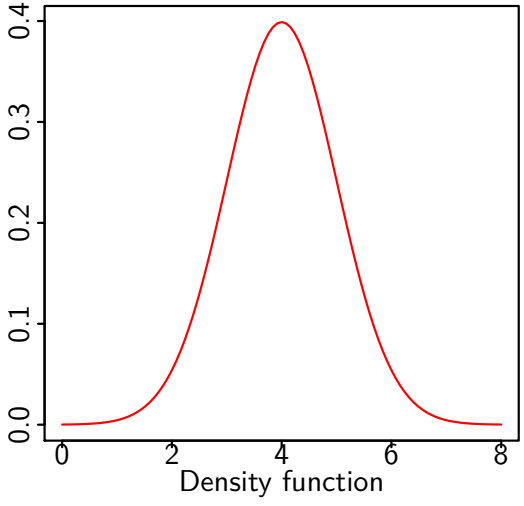
1. Even the 1^{st} and the 100.000^{th} solutions are close to each.
2. Unfortunately, CSP is not designed for.
3. Yes, why not !

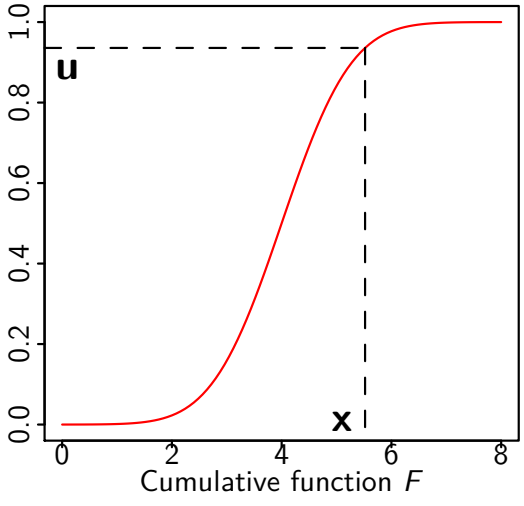
Usual probability distributions

Probability distributions

- We find them every where:
  - Social networks models (Exponential distribution).
  - Scaffold graph in biology.
  - So, why not in other software models ?
- Simulation is almost easy to implement.

Simulate a probability distribution





Sampling a probability distribution

- $X \rightsquigarrow \mathcal{N}(\mu = 4, \sigma = 1)$.
- $F(x) \in [0, 1] \rightarrow F^{-1}(u) = x$ is a sample of X.

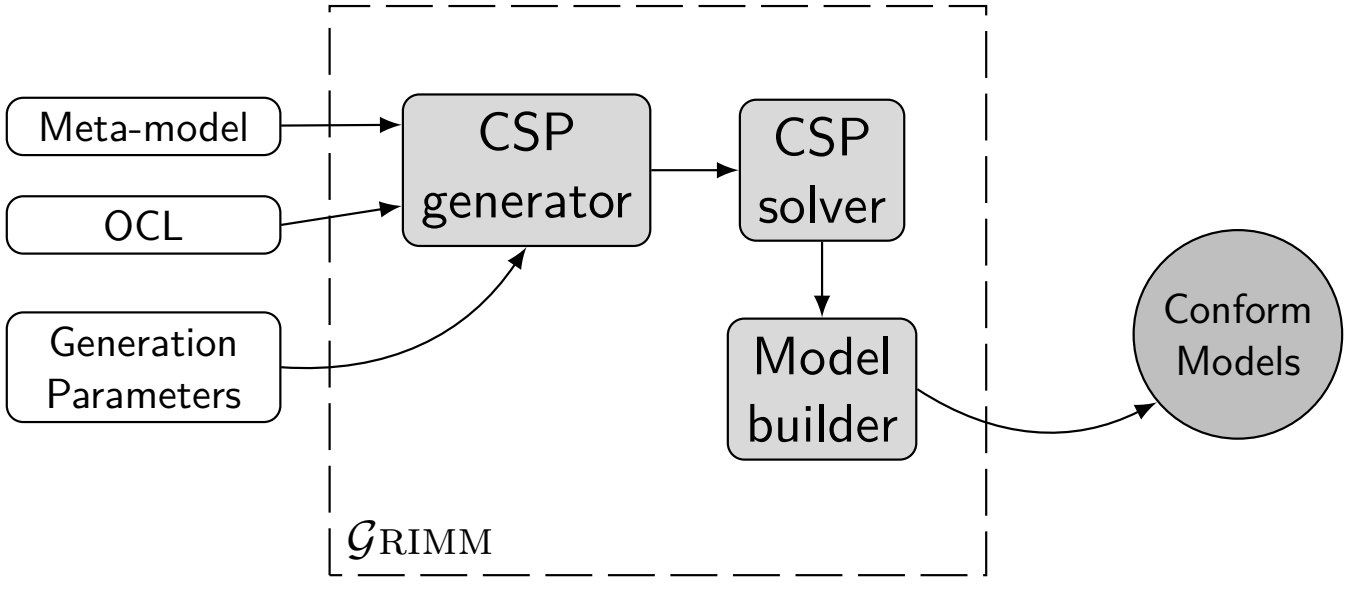
Integration into $\mathcal{G}$RIMM

Note

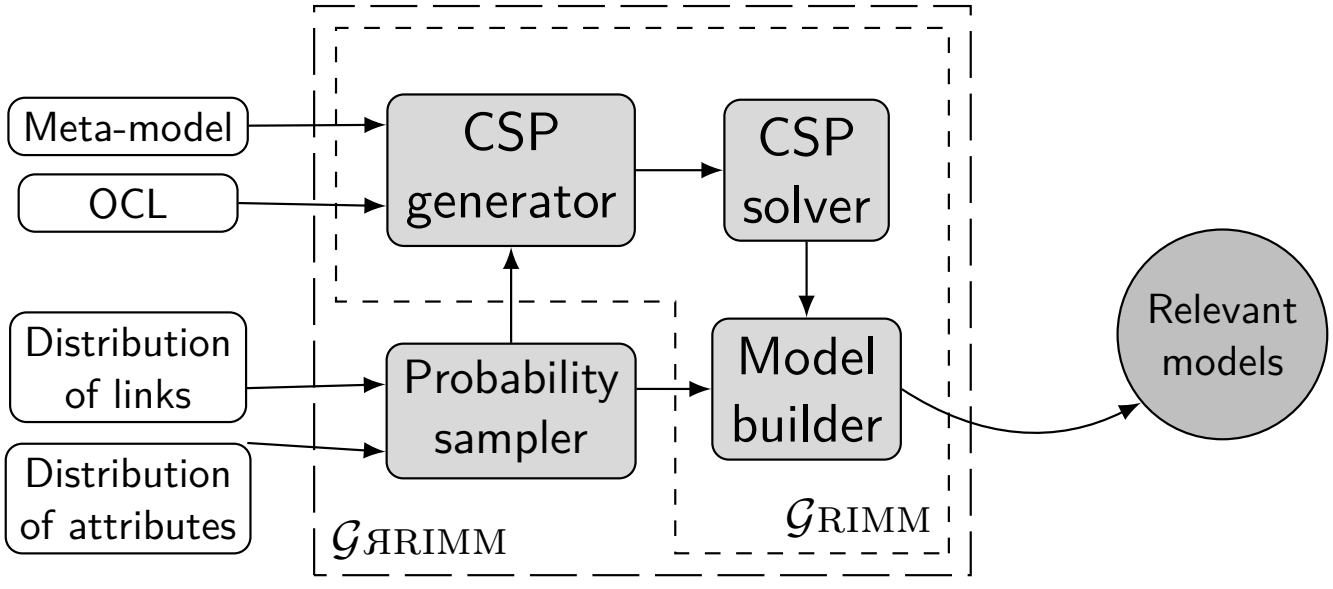
- Domain experts have more information about models than expressed in meta-models.
- Some elements lead to usual probability distributions with known parameters.

In $\mathcal{G}$RIMM, its concerns associations between classes and attributes' values.

$\mathcal{G}\mathcal{R}$RIMM: randomness in $\mathcal{G}$RIMM



$\mathcal{G}\mathcal{R}$RIMM: randomness in $\mathcal{G}$RIMM



Part III

Case studies

Case study: Java projects

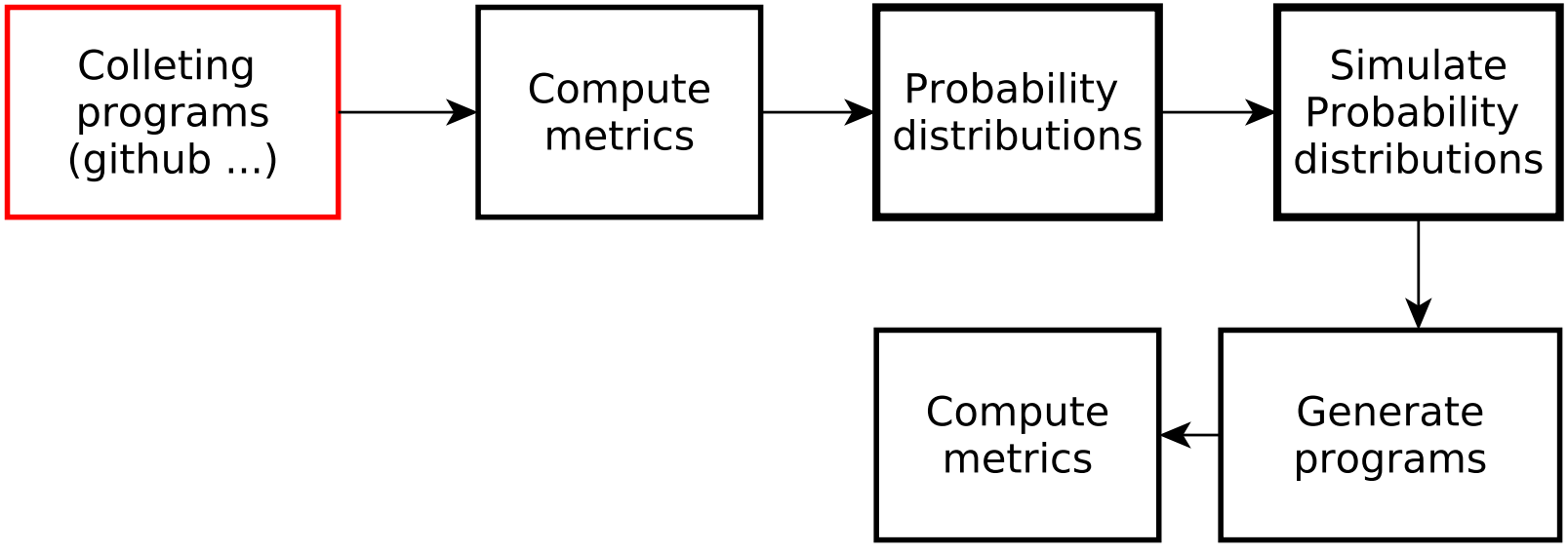
Case study

- Apply this method to OOP, a domain where many metrics exist.

Usefulness

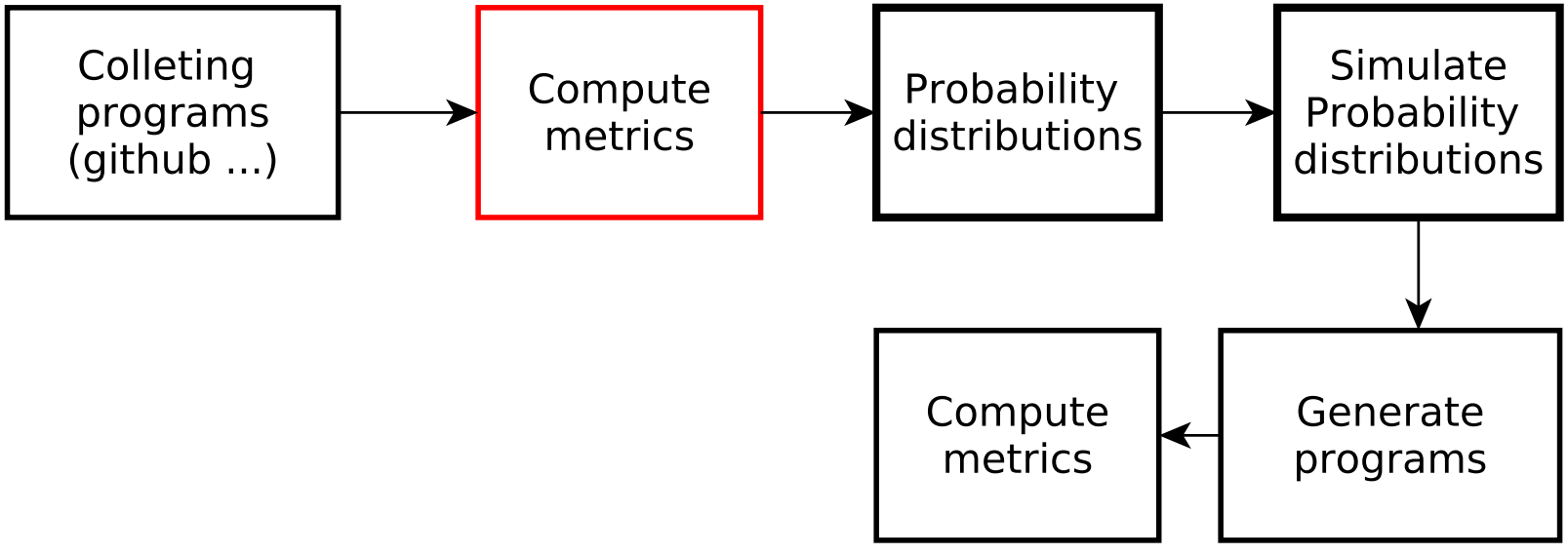
- Get models with characteristics:
  - Chosen by users.
  - Close to reality.

Process



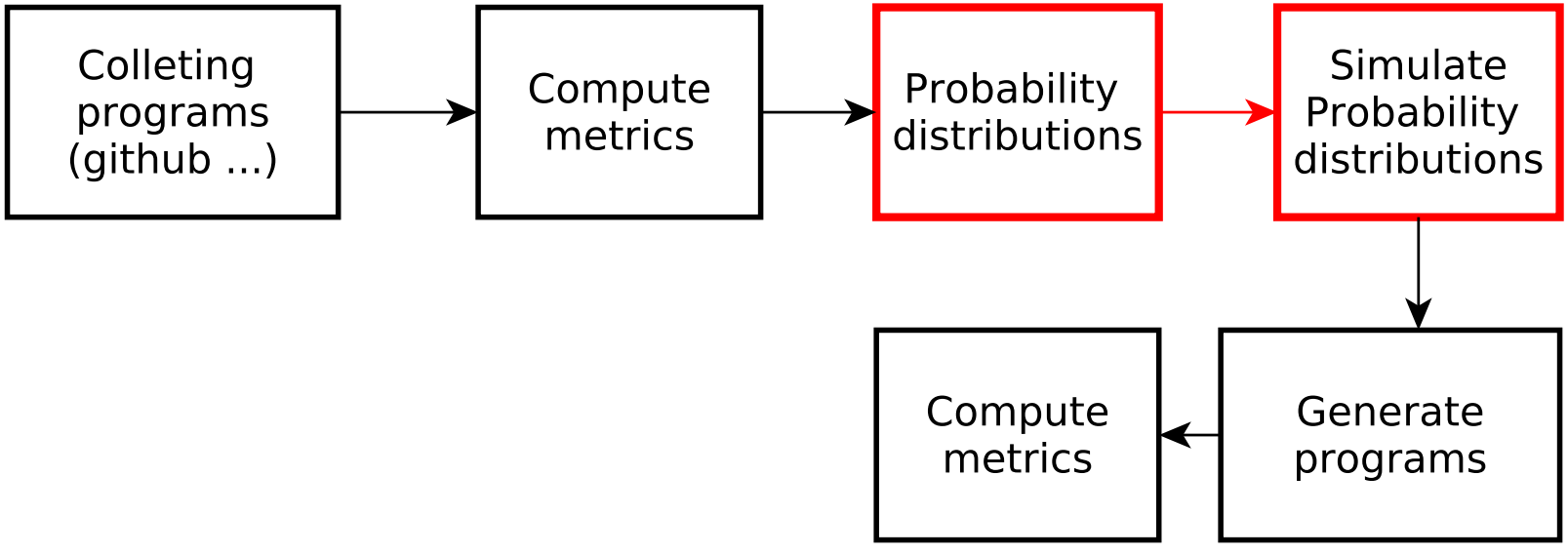
- 200 Java projects where analysed.

Process



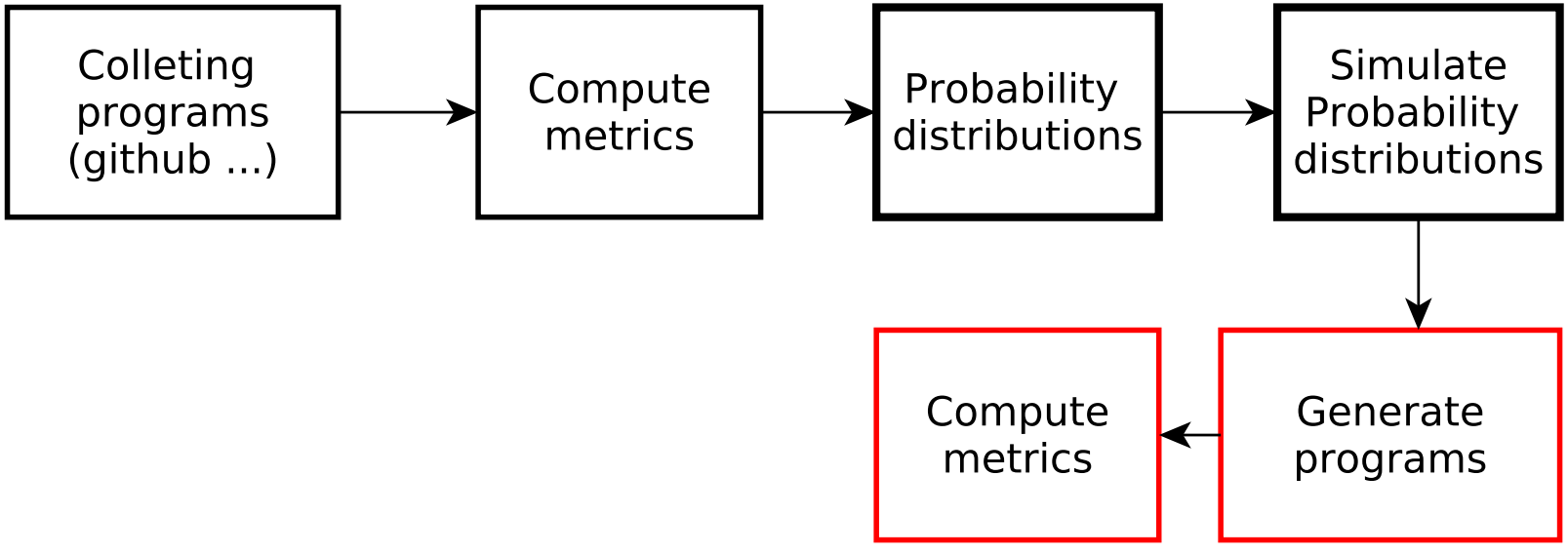
- Metrics: attributes / class, constructors / class, method visibility . . .

Process



- Add them to the CSP generator or to the model builder.

Process

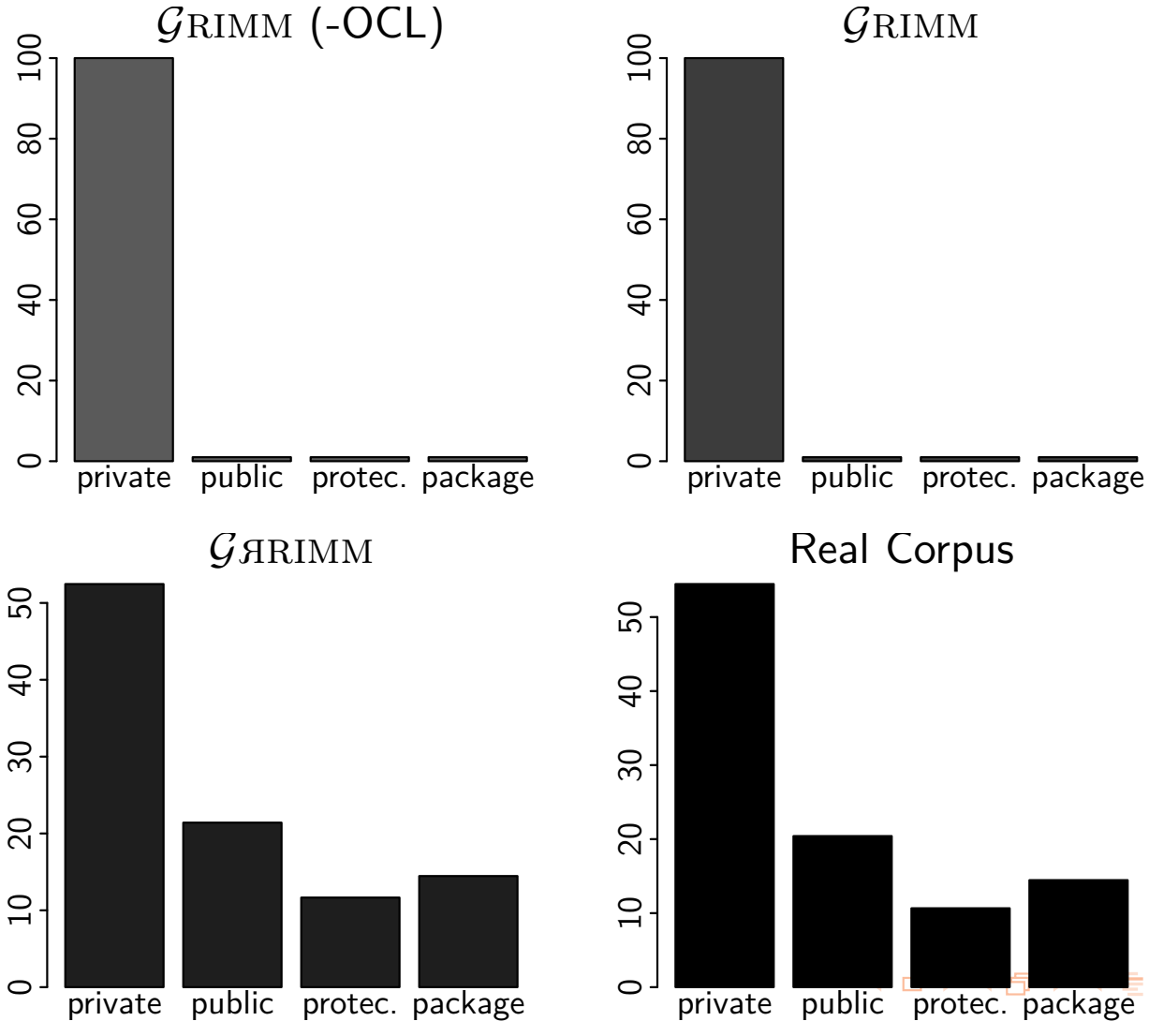


- Compare “generated” models’ metrics to “real” projects’ metrics.

Results: theoretical distributions

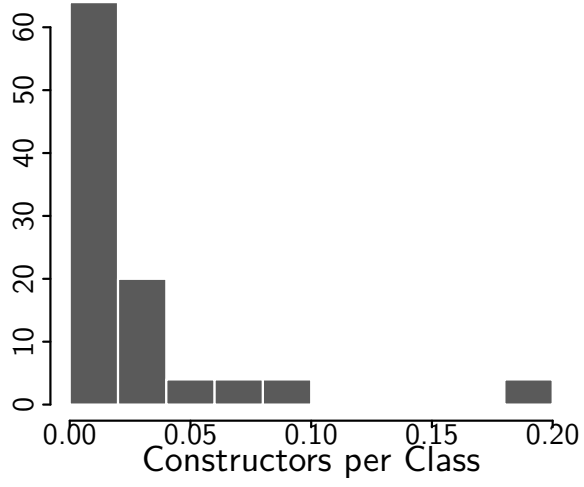
Metric	$\rightsquigarrow$	**Theoretical distrib.**
Class/Package	$\rightsquigarrow$	$\varepsilon(\frac{1}{8.387})$
Methods/Type	$\rightsquigarrow$	$\varepsilon(\frac{1}{7.06})$
Attributes/Type	$\rightsquigarrow$	$\mathcal{N}(3.46, 2.09)$
Constructor/Type	$\rightsquigarrow$	$\mathcal{N}(0.73, 0.26)$
Sub-Classes/Classe	$\rightsquigarrow$	$\varepsilon(\frac{1}{0.22})$
% Interface/Package	$\rightsquigarrow$	$\varepsilon(\frac{1}{8.001})$
Parameters/Methods	$\rightsquigarrow$	$\mathcal{N}(0.87, 0.25)$

Results: Java

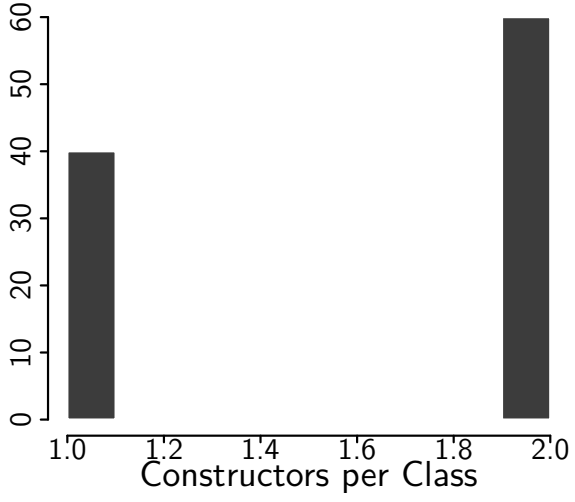


Results: Java

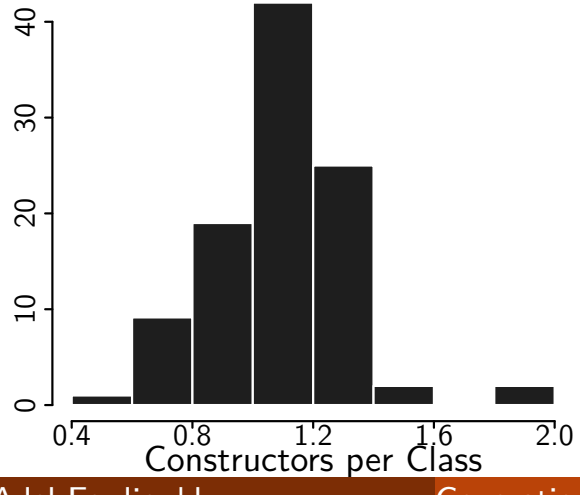
$\mathcal{G}$RIMM (-OCL)



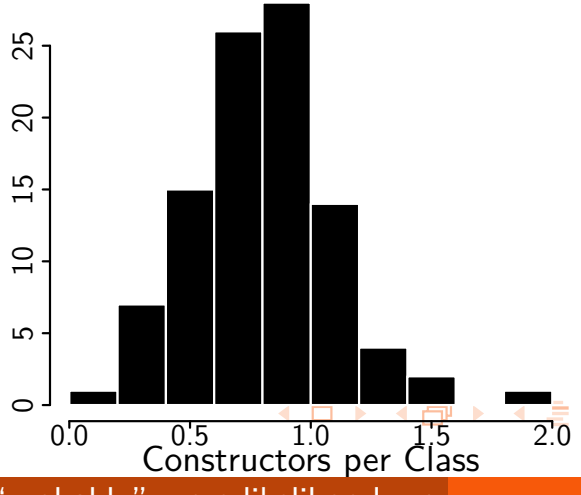
$\mathcal{G}$RIMM



$\mathcal{G}\mathfrak{R}$RIMM



Real Corpus



Case study: Scaffold Graph

What is it ?

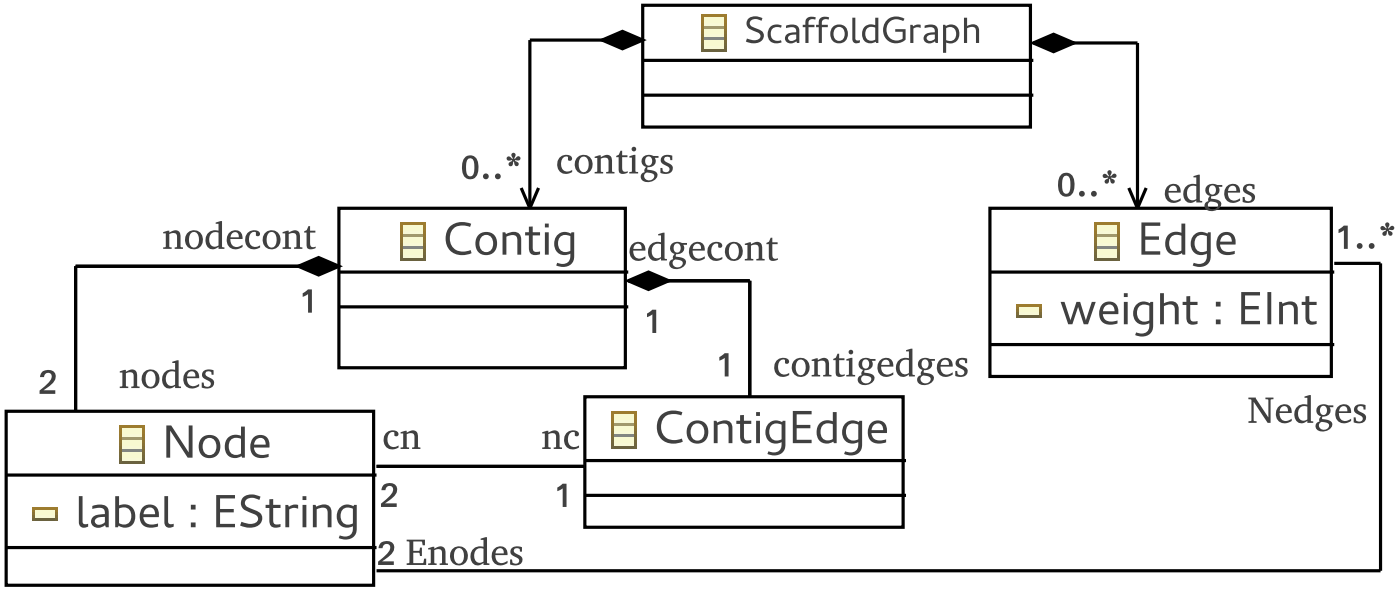
- A graph used in bio-informatics to read genomes.
- There is a technical difficulty to read them directly.
- Linearisation algorithms build genomes through scaffold graphs.

- Need data to test these algorithms.
- We want to generate them.

Process

Process

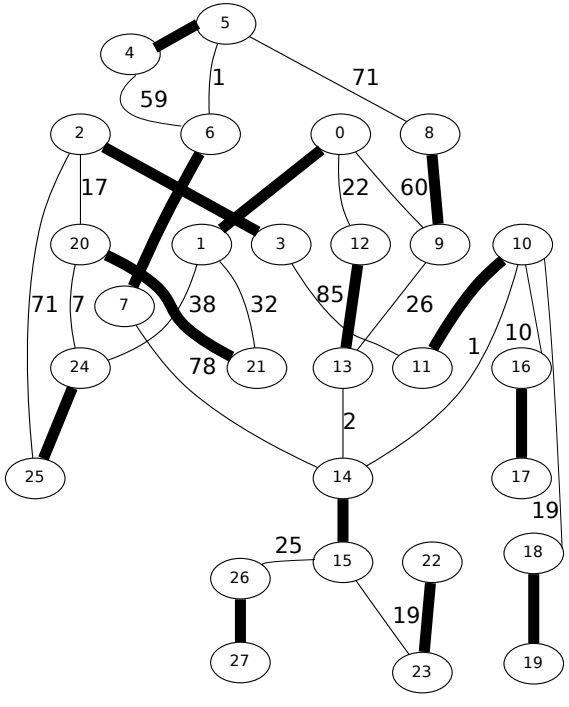
- Modelled as a meta-model (discussion with an expert),
- She also gives OCL constraints,



Process

Process

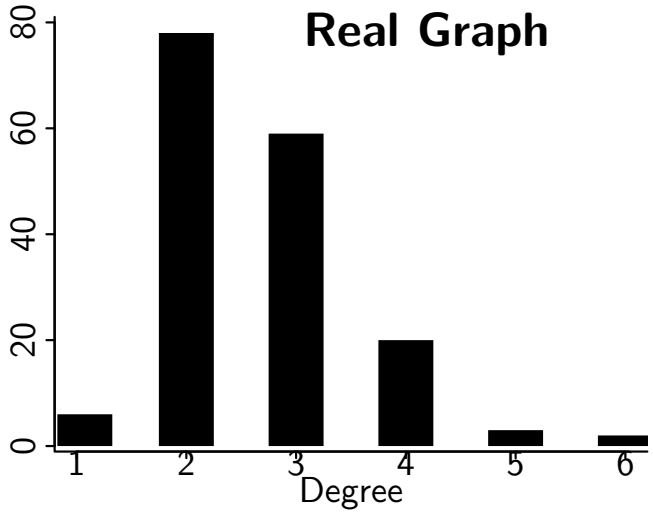
- Other characteristics: strong connectivity.



Process

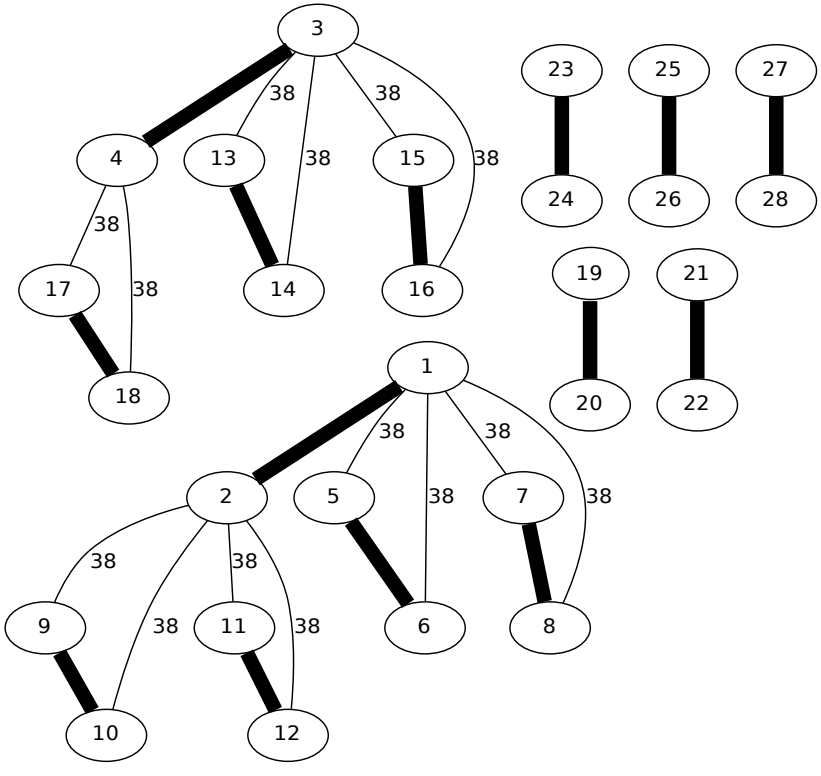
Process

- Information on degree distribution.

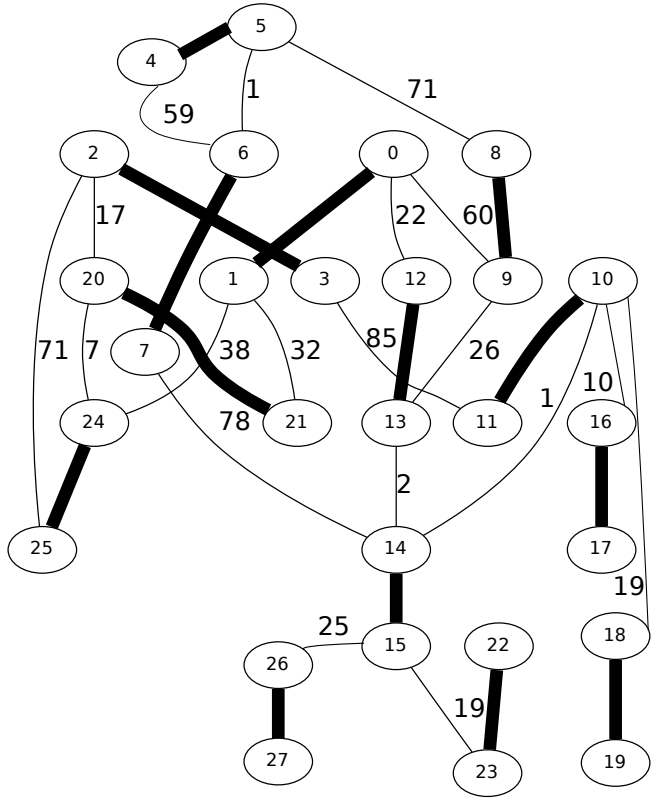


Results: Scaffold graphs

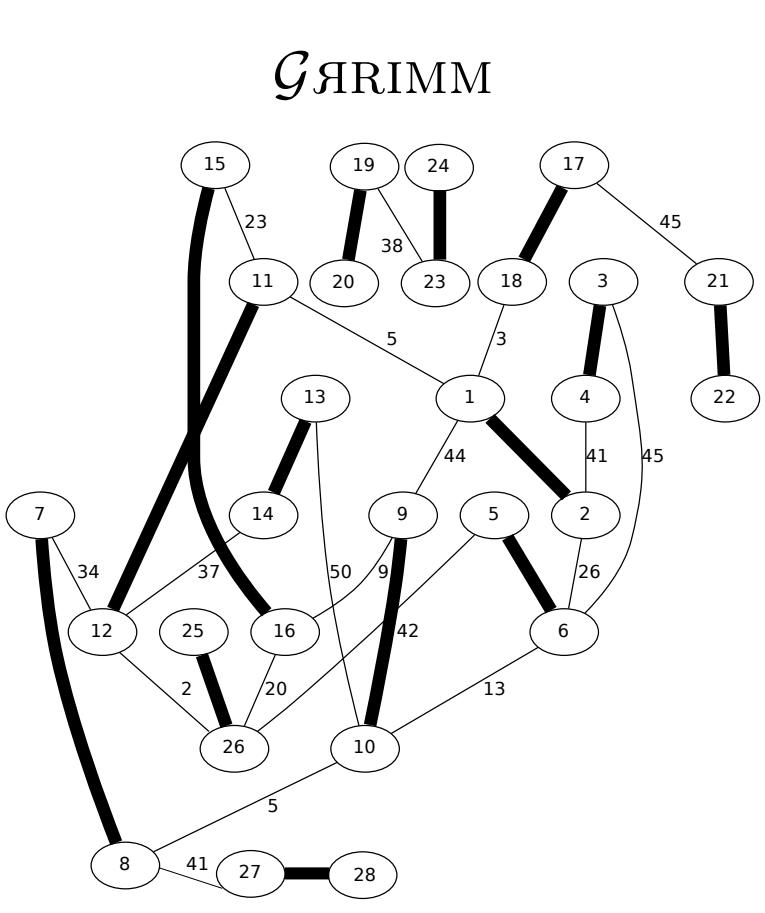
GRIMM

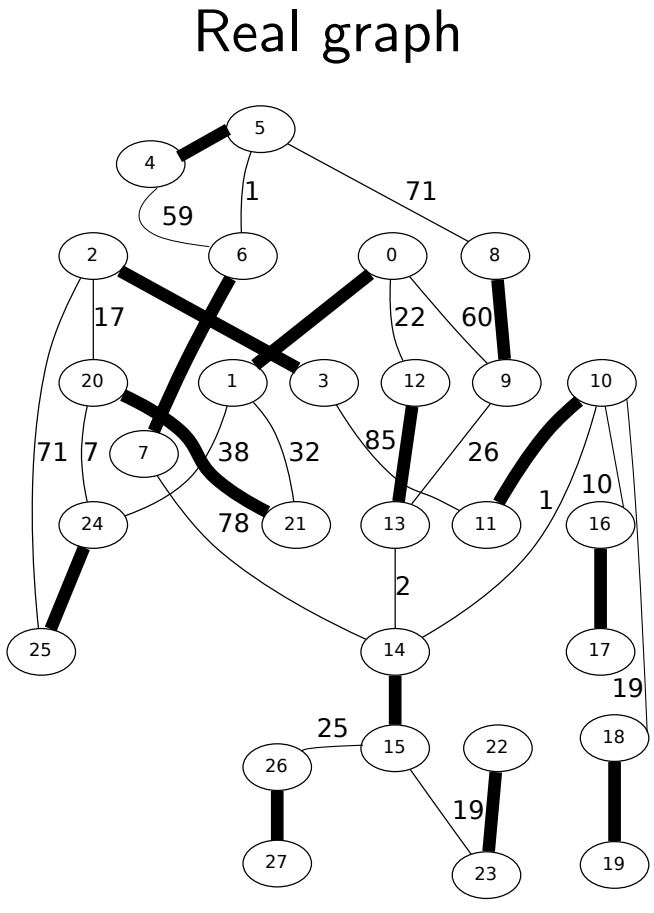


Real graph

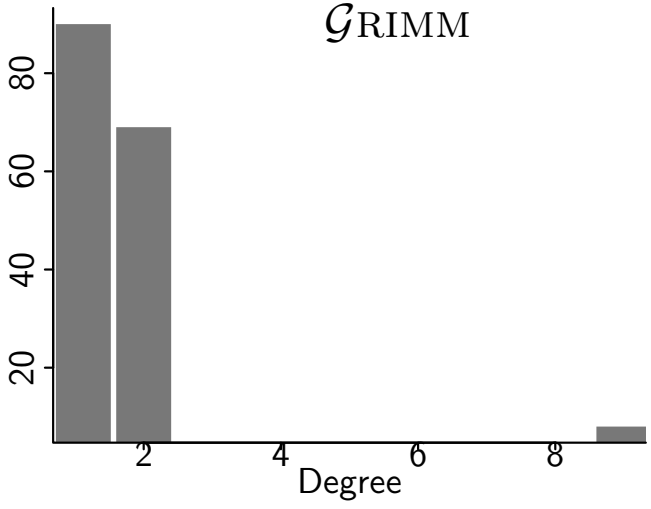


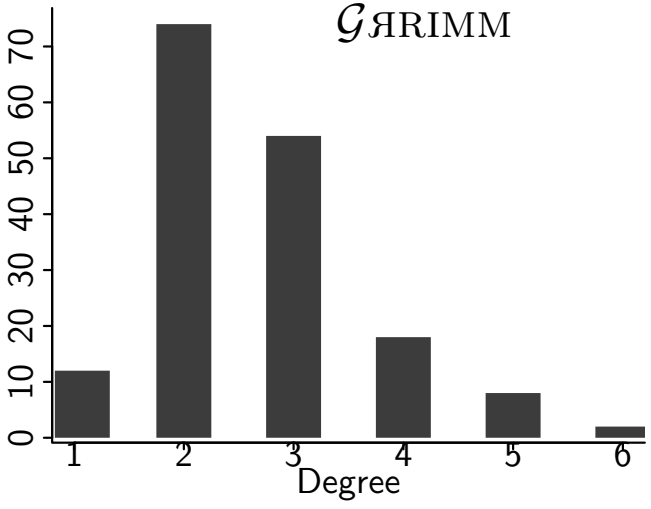
Results: Scaffold graphs

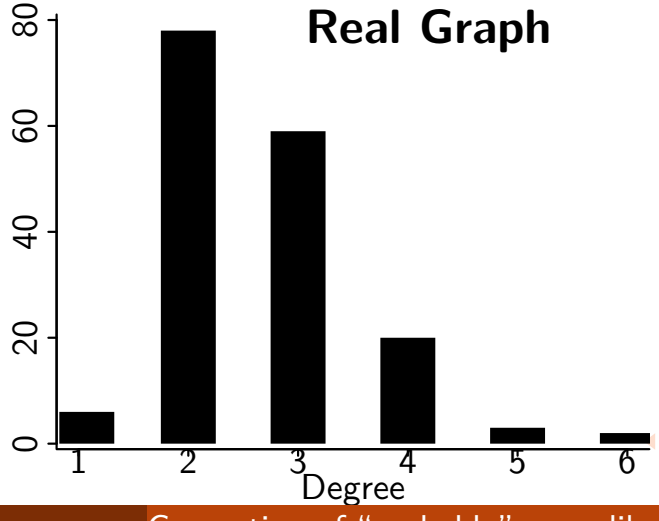




Results: Scaffold graphs







Conclusion

Summary

- An approach for model generation.
- It uses probability simulation for:
  - improve the relevance of models (their usefulness).
- Case study: generation of Java projects.
- Case study: generation of scaffold graphs.

The end