Synthesizing Benchmarks for Architecture Recovery Algorithms

Rodrigo Souza
rodrigorgs@gmail.com

Dalton Guerrero
dalton@dsc.ufcg.edu.br

Jorge Figueiredo
abrantes@dsc.ufcg.edu.br

Christina Chavez
flach@ufba.br

October 31, 2011

Abstract

As a software system evolves, its actual architecture may deviate from its original reference architecture. A better understanding of the actual architecture can be gained by applying architecture recovery algorithms to the source code of the system. Unfortunately, there is little knowledge regarding the quality of such algorithms, mostly because, in order to assess their accuracy, one needs updated, detailed architectures for a variety of software systems. To overcome such problem, we propose a simulation model that synthesizes implementation-level dependency graphs that conform to the module view of any given reference architecture. We show, using observations from network theory, that the graphs are very similar to dependency graphs extracted from the source code of real software systems. Then, we synthesize thousands of graphs, in a controlled way, and use them as benchmarks for six well-known architecture recovery algorithms: ACDC, Bunch, SL75, SL90, CL75, and CL90. By comparing the given reference architectures with those recovered by the algorithms, we conclude that ACDC and Bunch outperform the alternatives, specially if the architecture contains more than a couple of modules.

1 Introduction

Architectural drift

Reverse engineering / Architecture recovery algorithms

2 Background

Architecture recovery algorithms

Evaluation of architecture recovery algorithms.

3 The BCR+ model

TODO: Talk about network theory / complex networks / scale-free networks

When designing the architecture of a software system, the module viewpoint plays a major role, by specifying the modules of the system and their interrelationships. Constraining such dependencies benefits the maintenatibility, portability, and reusability of the software system.

In this section, we describe the BCR+ model for synthesizing graphs from a description of the module viewpoint of an arbitrary architecture — that is, the modules and dependencies between modules. The BCR+ model produces—by means of the mechanisms of growth and preferential attachment—directed graphs that are both scale-free and segmented in modules.

TODO: there are other models, but they don't take architecture as input

3.1 Overview

The BCR+ model is a generalization of a graph model proposed by Bollobás et al [1]. The original model generates directed graphs that are scale-free. Our model extends the original by generating graphs in which the nodes are organized in modules, according to an modular architecture given as input.

The BCR+ model takes the following parameters as input:

- number of vertices, n;
- three probability values, p_1, p_2 e p_3, com $p_1 + p_2 + p_3 = 1$;
- base in-degree, δ_{in};
- base out-degree, δ_{out}.
- an directed graph representing dependencies between modules, G;
- a constant, μ, with $0,0 \leq \mu \leq 1,0$;

The last two parameters are new in BCR+. The other ones are from the original model by Bollobás et al [1].

In the graph G, each vertex represents one module of the architecture. Each edge defines a relationship of dependency between modules. We say that a module M_1 depends on another module, M_2, if G contains an edge from the vertex that represents M_1 to the vertex that represents M_2. In the graph that is created, an edge from a vertex $v_1 \in M_1$ to another vertex, $v_2 \in M_2$, can be created only if M_1 depends on M_2 in the graph G or if M_1 and M_2 are the same module.

TODO: Say that these graphs contain both internal and external edges. Define internal/external.

The parameter μ controls the proportion of external edges in the graph—that is, edges connecting vertices in distinct modules. Lower values lead to graphs with fewer external edges.

The original model by Bollobás et al [1] is a particular case of BCR+ when $\mu = 0$ and G contains one single vertex, representing one single module.

As a computer model, BCR+ takes the parameters as inputs and, by running an algorithm, outputs a graph. The algorithm builds the output graph incrementally. It starts by creating a module for each vertex in the input graph G, and then adding one vertex to each module. After that, it creates all external edges that are allowed by G (see Figure 1). Then, the graph is modified according to three formation rules that are applied successively, in random order, until the graph grows to n vertices. At each algorithm step, the probability of the i-th being applied is given by the parameter p_i.

3.2 Formation Rules

There are three formation rules in BCR+. Each one modifies the output graph by adding or removing vertices or edges in the graph. Also, each formation rule has a probability of being applied, given by the parameters p_1, p_2 and p_3.

The rules are illustrated in Figure 2. Simplified rules:

- Rule 1: one vertex is added to some module, together with an outgoing edge to another vertex in the same module.
- Rule 2: one vertex is added to some module, together with an ingoing edge coming from another vertex in the same module.
- Rule 3: one edge is added between two preexisting vertices. There are two variations of this rule:
  - Rule 3a: choose vertices from distinct modules.
  - Rule 3b: choose vertices that are in the same module.

The rules 1, 2, and 3b come directly from the original model by Bollobás et al [1]. The rule 3a was created in BCR+ to account for inter-module dependencies.

The choice of vertices to which add edges to is done according to preferential attachment. When we say that a vertex "chosen according to f(x)", we mean that the probability of choosing the vertex x is proportional to f(x):

$$\mathrm{P}(x) \;=\; \frac{\mathrm{f}(x)}{\sum_i \mathrm{f}(i)}$$

The denominator is a normalization factor, such as the sum of probabilities P(x) is 1.

With this definition in mind, the rules can be fully specified:

- Rule 1: *Add a vertex with an outgoing edge.* An existing vertex, w, is chosen according to $\mathrm{f}(x) = \delta_{in} + \mathrm{g}_{in}(x)$ (that is, parameter δ_{in} added to the vertex in-degree). A new vertex, v, is added to the module that contains w, together with an edge from v to w.
- Rule 2: *Add a vertex with an ingoing edge.* An existing vertex, w, is chosen according to $\mathrm{f}(x) = \delta_{out} + \mathrm{g}_{out}(x)$. A new vertex, v, is added to the module that contains w, together with an edge from w to v.

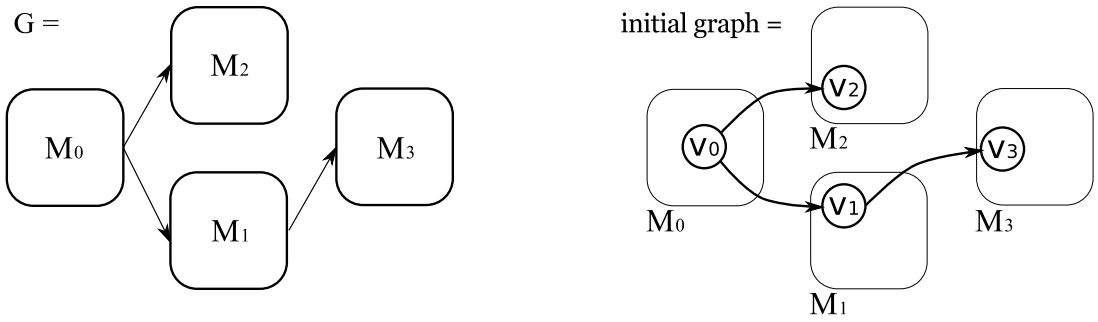

Figure 1: Initial graph synthesized by BCR+, given an input graph parameter G.

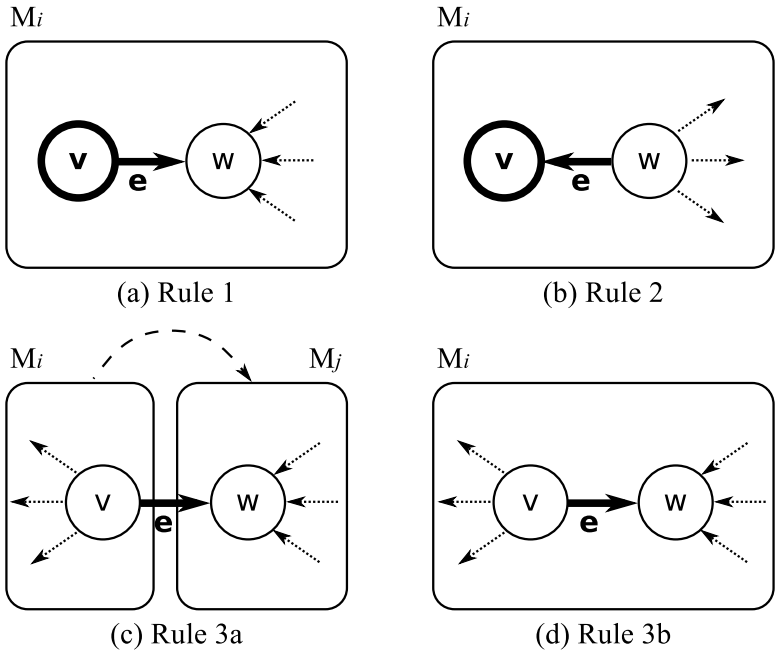

Figure 2: Formation rules for BCR+. M_i and M_j are distinct modules, such as M_i depends on M_j. In the diagram for each rule, thicker lines represent vertices and edges created when the rule is applied.

- Rule 3: *Add an edge between pre-existing vertices.* A vertex, v, is chosen according to $\mathrm{f}(x) = \delta_{out} + \mathrm{g}_{out}(x)$. Then, an edge is added from v to another vertex, w, chosen according to $\mathrm{f}(x) = \delta_{in} + \mathrm{g}_{in}(x)$. The vertex w is not chosen among the set of all vertices. In fact, there are two rules, and the choice of the rule to apply is probabilistic and depends on the parameter μ:
  - Rule 3a: with proability μ, w is chosen among the vertices that are in modules on which the module of v depends, according to the parameter G.
  - Rule 3b: with probability $1 - \mu$, w is chosen among the vertices that are in the same module as v.

TODO: BCR+ is a growth model, can simulate the evolution of a software system subject to constraints in module interaction. See CSMR paper (extended version), pg 2, col 2, just before section B.

3.3 Example

TODO: Do we need an Example section?

4 Software-Realism (aka "softwareness")

If synthetic graphs are to be used as surrogates to dependency graphs extracted from the source code of software systems, it is expected that they resemble such graphs. To capture this software quality of a graph, we define the concept of software-realism, that indicates to what extent a graph resembles software dependency graphs.

A metric for software-realism should be consistent, that is, it should present high values when measuring dependency graphs, and low values when measuring graphs from other domains—e.g., protein interaction networks, social networks etc.

First, we define a metric for software-realism, based on structural properties of a graph. Then, we show that the metric is consistent, by applying both to software dependency graphs and to graphs from non-software domains. Finally, we show that the BCR+ model is capable of synthesizing graphs with a high degree of software-realism.

4.1 Background

TODO: Write about triad concentration profiles [2]

TODO: Similarity between graphs:

$$\mathrm{sim}(a, b) \;=\; \mathrm{cor}(\mathrm{PCT}(a), \mathrm{PCT}(b)).$$

4.2 Software-Realism Metric

TODO: : Software-realism metric:

$$\mathrm{S}(x, R) \;=\; \frac{\sum_{s \in R} \mathrm{sim}(x, s)}{|R|}.$$

4.3 Metric Evaluation

TODO: Classification model:

$$\mathrm{m}(x, R, S_0) \;=\; \begin{cases} \text{sw-realistic}, & \text{se } \mathrm{S}(x, R) \geq S_0; \\ \text{non sw-realistic}, & \text{caso contrário.} \end{cases}$$

TODO: data collection; data processing (extracting dependency graphs); analysis (accuracy, precision, recall)

TODO: choice of S_0.

4.4 Software-Realism of BCR+

TODO: choice of parameters; graph synthesizing; graph classification (according to sw-realism)

5 Evaluation of Architecture Recovery Algorithms

TODO: Traditional experimental setup for evaluation of architecture recovery algorithms: given a dependency graph and a reference architecture, run the algorithm on the graph and compare the recovered architecture with the reference architecture.
TODO: MoJo metric.

TODO: Experiment 1: comparison between arch recovery algorithms **TODO:** Experiment 2: analysis of parameters **TODO:** Experiment 3: two-way vs. one-way dependencies

6 TODO: Other Possible Sections

Discussion

Limitations

References

[1] Béla BollobÁs, Christian Borgs, Jennifer Chayes, and Oliver Riordan. Directed scale-free graphs. In *Proceedings of the 14th Annual ACM-SIAM Symposium on Discrete Algorithms SODA '03*, pages 132–139. Society for Industrial and Applied Mathematics, 2003.

[2] R. Milo, S. Shen-Orr, S. Itzkovitz, N. Kashtan, D. Chklovskii, and U. Alon. Network motifs: simple building blocks of complex networks. *Science*, 298(5594):824–827, October 2002.