SoftwareX 7 (2018) 273–280

Contents lists available at ScienceDirect

SoftwareX

journal homepage: www.elsevier.com/locate/softx

Original software publication

Decremental dynamic algorithm to trace mutually connected clusters

Deokjae Lee [a], S. Hwang [b], S. Choi [c], B. Kahng [a,*]

[a] CCSS, CTP and Department of Physics and Astronomy, Seoul National University, Seoul 08826, Republic of Korea
[b] Institute for Theoretical Physics, University of Cologne, Cologne, 50937, Germany
[c] Michigan Center for Theoretical Physics, Randall Laboratory of Physics, University of Michigan, Ann Arbor, MI 48109, USA

ARTICLE INFO

Article history:
Received 25 August 2017
Accepted 13 August 2018

Keywords:
Interdependent networks
Mutually connected clusters
Percolation
Hybrid phase transition
Euler tour tree

ABSTRACT

The structure and dynamics of interdependent networks model catastrophic failures in complex systems that are interdependent. Percolation transitions on these networks exhibit hybrid phase transitions, which have significant practical implications for the early detection of large-scale failures. While the computer simulation of the percolation transitions and related dynamics can effectively be reduced to the computation of mutually connected clusters, such a computation is nontrivial, and several algorithms to handle the task have been proposed. Here we introduce a C++ implementation of one of the algorithms. This implementation uses intrusive data structures and thus provides a greater flexibility for applications in which efficient memory access is critical. The data structures, which we provide as a part of the library, are also useful for general percolation problems.

Code metadata

Current code version	v1.0
Permanent link to code/repository used for this code version	https://github.com/ElsevierSoftwareX/SOFTX-D-17-00066
Legal Code License	MIT
Code versioning system used	git
Software code languages, tools, and services used	C++
Compilation requirements, operating environments & dependencies	Any OS with a compiler supporting C++11 standard. A compile time dependency on Boost.Optional (http://www.boost.org/).
If available Link to developer documentation/manual	
Support email for questions	lee.deokjae@gmail.com

1. Introduction

Catastrophic failures stemming from heavy mutual dependencies between complex systems are critical problems in modern society. For example, the mutual dependency between the power grid and the computer network caused a large abrupt blackout in Italy in 2003 [1]. Statistical physics of interdependent networks has been considered the primary paradigm to understand such phenomena since its recent introduction [1–5]. An important aspect of such catastrophic failures is that a small failure in a system can propagate to a large fraction of the system with a low but not negligible probability. More precisely, the size distribution of the total failures in a cascade follows a power law with an outlier whose size is a finite fraction of the entire system [2,6,7]. Percolation transitions in interdependent networks and the associated avalanche dynamics capture the essence of this phenomenon.

A percolation transition in interdependent networks is a hybrid phase transition (HPT) [2,6]. This has significant theoretical and practical implications. In HPTs, the order parameter exhibits a discontinuity at the transition point, whereas the system also shows critical phenomena in the vicinity of the transition point, manifested by divergent physical quantities. Thus, they possess the characteristics of both the first- and second-order phase transitions. HPTs have only been noticed recently by researchers and thorough theoretical understanding of HPTs is an ongoing and active area of research. In particular, for some practical applications,

* Corresponding author.
E-mail address: bkahng@snu.ac.kr (B. Kahng).

https://doi.org/10.1016/j.softx.2018.08.002

the divergent quantities may be used as precursors of catastrophic failures [8,9,2]. Studying such quantities therefore gives access to information that allows one to predict or perhaps even prevent such failures, making it a significant attribute of the paradigm.

Using computer simulation to study the percolation and avalanche dynamics of interdependent networks is a challenging task. It requires a dynamic (online) algorithm to trace the size distribution of the mutually connected clusters, as defined in the following section, while nodes or edges are being removed from the networks. Brute-force approaches based on previous algorithms used for ordinary percolation problems are not efficient enough for this process. Therefore, several algorithms have been proposed to specifically tackle this problem [10–12]. In this paper, we introduce an implementation of the algorithm developed in [11].

Besides its main purpose of dealing with interdependent networks, this implementation has some noteworthy properties that are useful for general percolation problems. First of all, it provides several efficient data structures, including an implementation of a fully dynamic algorithm for graph connectivity (HDT algorithm) developed in [13]. Here, the term connectivity denotes information about the connected components of a graph, such as the existence of a path between any pair of nodes. For the incremental percolation processes during which nodes or edges are added gradually, the disjoint-set forest is a simple and very efficient solution for simulations [14,15]. However, for the decremental percolation processes during which nodes or edges are removed gradually, an efficient solution is not trivial. The HDT algorithm is a recent advance in the computer science discipline, and is directly applicable to various decremental percolation processes.

Second, the implementations of the data structures are intrusive [16]. Intrusive data structures are widely used in the development of performance- and resource-critical software such as operating systems and games [17,18]. An intrusive implementation of a data structure does not concern the allocation and release of the memory to store the data. The users must allocate and release the memory explicitly and provide hooks to be used by the implementation (see Section 4.4). Whereas this choice imposes more boilerplates in general, the explicit control of the memory comes with the flexibility and opportunity to optimize memory access. Being able to experiment and control the data cache miss rate is crucial for improving runtime performance in current computer architectures.

2. Percolation and avalanche process in interdependent networks

Let us consider two networks $G_1 = (V_1, E_1)$ and $G_2 = (V_2, E_2)$, where V_1 and V_2 are the sets of nodes in each network, and E_1 and E_2 are the sets of edges in each network. The mutual dependencies between these networks are dictated by 'special edges' that link a node of one network to a node of the other. We will refer to them as 'interdependency edges,' and to the two networks as 'interdependent networks' [1]. Here, both the edges in a graph and the interdependency edges between graphs are assumed to be undirected.

To describe the interdependency structure, it is convenient to introduce the set of nodes interdependent on a node n as $I(n)$, i.e., every node $m \in I(n)$ is linked to n by an interdependency edge. Furthermore, by abuse of notation, we define $I(W) = \bigcup_{n\in W} I(n)$ for any subset W of either V_1 or V_2.

A mutually connected cluster (MCC) M is a pair (H_1, H_2) of subgraphs H_is of G_is for $i = 1, 2$ with the following conditions, given the subgraphs $H_i = (W_i, F_i)$:

1. Each pair of nodes in W_i is connected by at least one path in H_i.
2. $I(W_i) \neq \emptyset$ and $I(W_1) = W_2$ and $I(W_2) = I(W_1)$.
3. M is maximal in the sense that any addition of nodes or edges to H_i violates 1 or 2.

We refer to H_1 and H_2 as the projection of M on G_1 and G_2 respectively. Each forms a connected component in the respective network. If $|W_1| + |W_2| = O(|V_1| + |V_2|)$, M is called the giant mutually connected cluster (GMCC). As a special case, if each node has one and only one interdependency edge, each network is partitioned by the set of all projections, and the projections of any MCC to the two networks have the same size (number of nodes). Otherwise, some nodes may not belong to any MCC, and projections of one MCC to the two networks may have different sizes.

To model the propagation of failures in mutually dependent complex systems, let us assume that a node is functional if and only if (a) it belongs to the giant cluster of its network, and (b) each of its interdependent node also belongs to the giant cluster of the respective network. This criterion of functionality defines a connected component of functional nodes in each network, and it coincides with the projection of the GMCC [1,19].[1]

We can model the failure of a node by deleting or separating it from the GMCC. By the nature of the GMCC, all interdependent neighbors of a failed node will simultaneously fail, causing some nodes to be separated from the largest cluster of the network. These separated nodes will cause additional failures, which in turn trigger more failures, and so on. The process iterates until no more nodes fail. This propagation of failures is called an avalanche. The algorithm we present here is essentially a formalization of this avalanche process for MCCs.

With the use of our fast implementation, one can efficiently study percolation problems by tracing the size of the GMCC and controlling some aspects of the networks, such as the mean degree, number of initially failed nodes, degree distributions, and so on [1,19,21,22,12]. Various properties of the avalanche dynamics, for example the mean size and the duration of the propagation, are also quantities of interest [1,2,23,12,6].

3. Algorithm to trace MCCs dynamically

In Algorithm 1, we present a high-level description of the algorithm that traces the MCCs while nodes or edges are being deleted one by one. The description is slightly more general than the description in [11], in the sense that in our current setting we allow each node to have zero or multiple interdependencies. To start, the algorithm requires every network to have only one connected component. If not, we add auxiliary edges to make each network connected, and they will later be removed by applying the algorithm. The algorithm is essentially a process checking the following two assertions when edges are deleted: (a) If two nodes have no path in a network, then they cannot be in the same MCC. (b) If a node has two interdependent nodes that are in different connected components, then the node cannot belong to any MCC. Hence, an efficient way to maintain the information on the connectivity during the process is crucial for ensuring the performance of the algorithm.

For this purpose, the algorithm maintains a spanning forest for each network. In contrast to the common usage of spanning forests in which they represent the connected components of the networks, we employ them here to store the projections of MCCs. By construction, initially there is only one MCC, and each spanning

[1] One may assume that each node must have at least one interdependent node that belongs to the giant cluster instead of assuming all interdependent nodes belong to the giant cluster [20]. This leads to a definition of MCC different from ours.