Addendum to Chapter 2: Multilayer, Dynamic, Hyper Networks

Michele Coscia (mcos@itu.dk)

August 3, 2018

Abstract

Here we introduce further extensions to the simple graph model to deal with more complex interaction patterns in the real world. In particular, these sections focus on multilayer networks, dynamic networks, and hypergraphs.

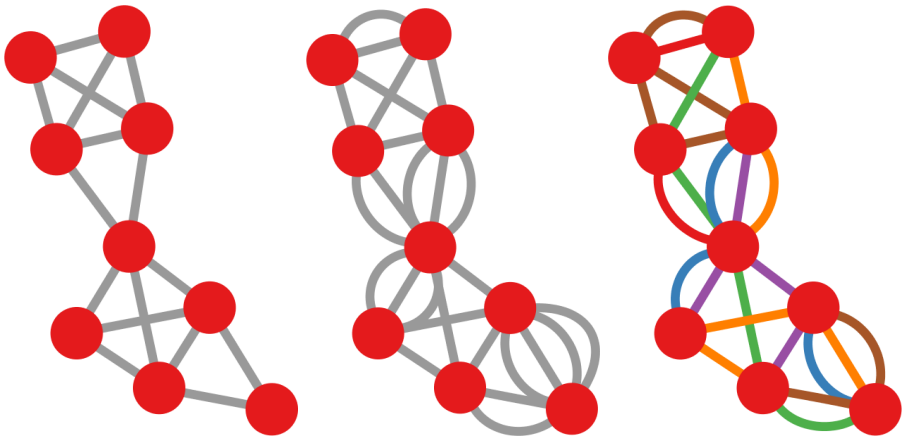
1 Multilayer Networks

Traditionally, network scientists try to focus on one thing at a time. If they are interested in analyzing your friendship patterns, they will choose one network that closely approximate your actual social relations and they will study that. For instance, they will download a sample of the Facebook graph. Or they will analyze tweets and retweets.

However, in some cases, that is not enough to really grasp the phenomenon one wants to study. If you want to predict a new connection on Facebook, something happening in another social media might have influenced it. Two people might have started working in the same company and thus first connected on Linkedin, and then became friends and connected on Facebook. Such scenario could not be captured by simply looking at one of the two networks. We invented multilayer networks to answer this kind of questions.

There are two ways to represent multilayer networks. The simpler is to use a multigraph. Differently from a simple graph (Figure 1a), in which every pair of nodes is forced to have at most one edge connecting them, in a multigraph (Figure 1b) we allow an arbitrary number of possible connections. We can add a "type" to each connection, distinguishing them: one for Facebook, one for Twitter, one for Linkedin (Figure 1c).

In practice, every edge type – or label – represents a different layer of the network. A pair of nodes can establish a connection in any layer, even at the same time. Each layer is a simple graph. In this course – and generally in computer science – the most used notation is to indicate a multilayer network as $G = (V, E, L)$. V and E are the sets of nodes and edges, as usual. L is the



(a) (b) (c)

Figure 1: (a) A simple graph. (b) A multigraph, allowing multiple edges between the same pair of nodes. (c) A simple multilayer network, a multigraph where each edge has a type (represented here by the edge's color).

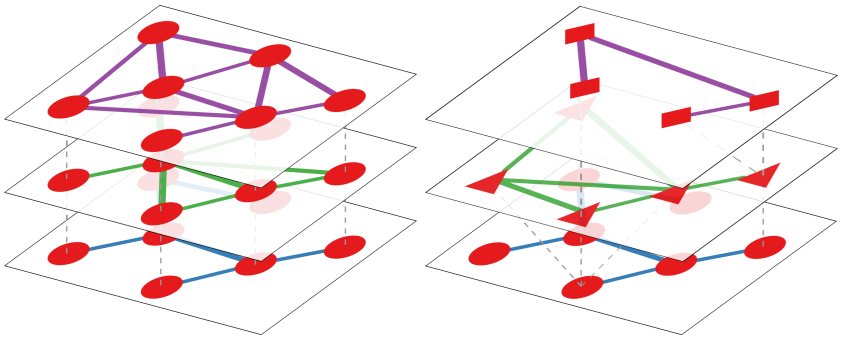
set of layers – or labels. An edge is now a triple $(u, v, l) \in E$, with $u, v \in V$ as nodes, and $l \in L$ as the layer.

The model that we introduce in Figure 1c is but the simplest way to represent multilayer networks. This strategy rests on the assumption that there is a one-to-one node mapping between the layers of the network. In other words, the entities in each layer are always the same: you are always you, whether you manage your Facebook account or your Linkedin one. However, this is not always true in all scenarios: sometimes a node in one layer can map to multiple nodes – or none! – in another.

To fix this problem we extend the model. We introduce the concept of "interlayer coupling". In this scenario, the node is split into the different layers to which it belongs. In this case, you are not you anymore: you are the union of the "Facebook you", the "Linkedin you", the "Twitter you". Figure 2a shows the visual representation of this model: each layer is a slice of the network. There are two types of edges: the intra-layer connections – the traditional type: we're friends on Facebook, Linkedin, Twitter –, and the inter-layer connections. The inter-layer edges run between layers, and their function is to establish that the two nodes in the different layers are really the same node: they are *coupled* to – or *dependent* on – each other.

Formally, now our network is $G = (V, E, C)$. V is still the set of nodes, but now we split the set of edges in two: E is the set of classical edges, the intra-layer one – connections between different people on a platform –; and C is the set of coupling connections, the inter-layer one, denoting dependencies between nodes in different layers.

Having a full set of coupling connections enables an additional degree of



(a) (b)

Figure 2: The extended multilayer model. Each slice represents a different layer of the network. Dashed grey lines represent the inter-layer coupling connections. (a) A multilayer network with trivial one-to-one coupling. (b) A multilayer network with complex interlayer coupling.

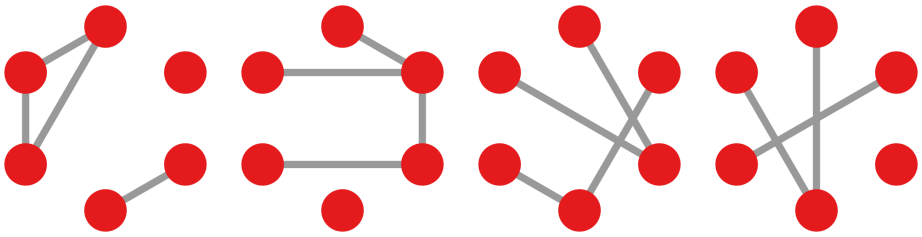
freedom. We can now have nodes in one layer expressing coupling with multiple nodes in other layers. In our social media case, we are now allowing you to have multiple profiles in one platform that sill map on your single profile in another. For instance, you can run as many different Twitter accounts as you want, and they are still coupled with your Facebook account. To get a visual sense on what this means, you can look at Figure 2b.

2 Dynamic Networks

Most networks are not crystallized in time. Relationships evolve: they are created, destroyed, modified over time by all parties involved. Every time we use a network without temporal information on its edges, we are looking at a particular slice of it, that may or may not exist any longer.

For many tasks, this is ok. For others, the temporal information is a key element. Imagine that your network represents a road graph. Nodes are intersections, and edges are stretches of the street connecting them. Roadworks might cut off a segment for a few days. If your network model cannot take this into account, you would end up telling drivers to use a road that is blocked, creating traffic jams and a lot of discomfort. That is why you need dynamic networks.

Consider Figures 3a to 3d as an example. Here, we have a social network. People are connected only when they are actually interacting with each other. We have four observations, taken at four different time intervals. Suppose that you want to infer if these people are part of the same social group – or community. Do they? Looking at each single observation would lead us to say *no*. In each time step the nodes are disconnected in multiple components. Adding the observations together, though, would create a clique, where all nodes are



(a) $t = 1$ (b) $t = 2$ (c) $t = 3$ (d) $t = 4$

Figure 3: An example of dynamic network. Each figure represents the same network, observed at different points in time.

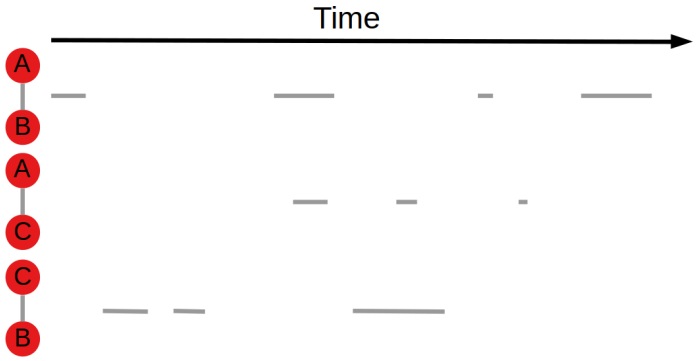


Figure 4: An example of dynamic edge information. Time flows from left to right. Each row represent a possible potential edge between nodes A, B, and C. The moments in time in which each edge is active are represented by gray bars.

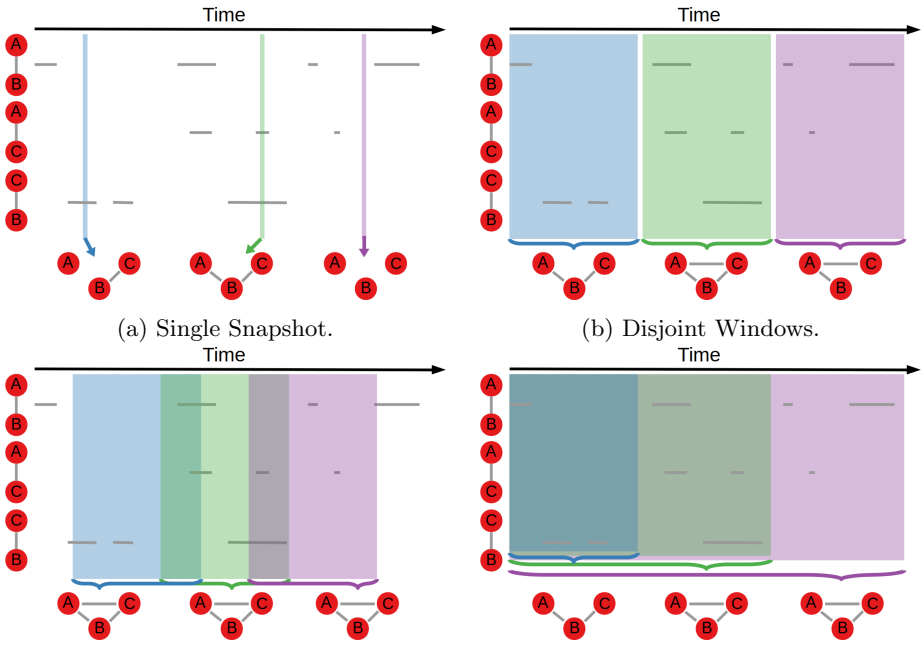
connected to each other. Taking into account the dynamic information allows us to make the correct inference. Yes, these nodes form a tightly connected group.

In practice, we can consider a dynamic network as a network with edge attributes. The attribute tells us when the edge is active – or inactive, if the connection is considered to be "on" by default, like the road graph. Figure 4 shows a basic example of this, with edges between three nodes.

More formally, our graph can be represented as $G = (G_1, G_2, ..., G_n)$, where each G_i is the i-th snapshot of the graph. In other words, $G_i = (V_i, E_i)$, with V_i and E_i being the set of nodes and edges active at time i.

How do we deal with this dynamic information when we want to create a static view of the network? There are a four standard techniques.

- *Single Snapshot* – Figure 5a. This is the simplest technique. You choose a moment in time and your graph is simply the collection of nodes and edges active at that precise instant. This strategy works well when the edges



(a) Single Snapshot. (b) Disjoint Windows.

(c) Sliding Windows. (d) Cumulative Windows.

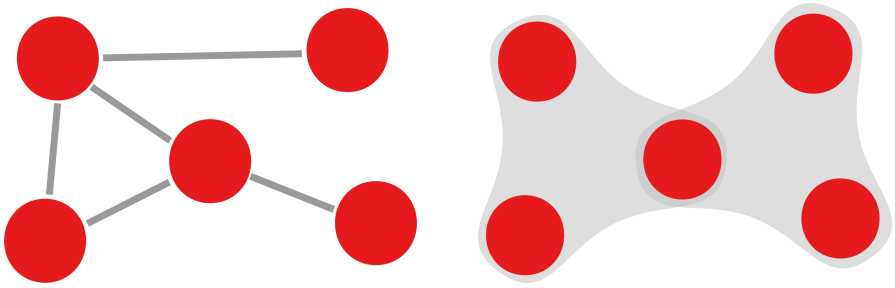
Figure 5: Different strategies for converting dynamic edges into a graph view.

in your network are "on" by default. It risks creating an empty network when edges are ephemeral and/or there are long lulls in the connection patterns, for instance in telecommunication networks at night.

- *Disjoint Windows* – Figure 5b. Similar to single snapshot. Here we allow longer periods of time to accumulate information. Differently from the previous technique, no information is discarded: when a window ends, the next one begins immediately. Works well when it's not important to maintain continuity.

- *Sliding Windows* – Figure 5c. Similar to disjoint windows, with the difference that we allow the observation periods to overlap. That is, the next window starts before the previous one ended. Works well when it is important to maintain continuity.

- *Cumulative Windows* – Figure 5d. Similar to sliding windows, but here we fix the beginning of each window at the beginning of the observation period. Information can only accumulate: we never discard edge information, no matter how long ago it was firstly generated. Every windows always includes the information of all previous windows. Works well when the effect of an edge never expires, even after the edge has not been active for a long time.



(a) Classical Graph. (b) Hypergraph.

Figure 6

Note how these different techniques generate radically different "histories" for the network in Figure 5a to 5d, even when the edge activation times are identical.

3 Hypergraphs

In the classical definition, an edge connects two nodes – the gray lines in Figure 6a. Your friendship relation involves you and your friend. If you have a second friend, that is a different relationship. But there exists some cases in which connections bind together multiple people at the same time. For instance, consider team building: when you do your final project with some of your classmates, the same relationship connects you with all of them. When we allow the same edge to connect more than two nodes we call it a *hyperedge* – the gray area in Figure 6b. A collection of hyperedges makes a *hypergraph*.

It is difficult to work with hypergraphs. Specialized algorithms to analyze them directly exist, but they become complicated very soon. In the vast majority of cases, we will transform hyperedges into simpler network forms and then apply the corresponding simpler algorithms.

There are two main strategies to simplify hypergraphs. The first is to transform the hyperedge into the simple edges it stands for. If the hyperedge connects three nodes, we can change it into a triangle – a 3-clique – in which all three nodes are connected to each other. In the project team example, the new edge simply represent the fact that the two people are part of the same team. The advantage is a gain in simplicity, the disadvantage is that we lose the ability to know the full team composition by looking at its corresponding hyperedge: we need to explore the newly created cliques.

The second strategy is to turn the hypergraph into a bipartite network. Each hyperedge is converted into a node of type 1, and the hypergraph nodes are converted into nodes of type 2. If nodes are connected by the same hyperedge, they all connect to the corresponding node of type 1. In the project team example, the nodes of type 1 represent the teams, and the nodes of type 2

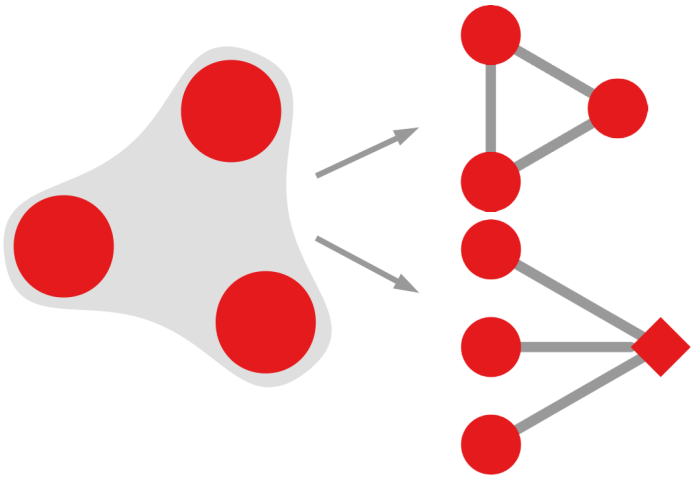


Figure 7: The two ways to convert a hyperedge into simpler forms. A hyperedge connecting three nodes can become a triangle (top right), or a bipartite network (bottom right).

the students. This is an advantageous representation: it is simpler than the hypergraph, but it preserves some of its abilities, for instance being able to reconstruct teams by looking at the neighbors of the nodes of type 1. However, the disadvantage with respect to the previous strategy is that there are fewer algorithms working for bipartite networks than with unipartite networks.

Figure 7 provides a simple example on how to perform these two conversion strategies on a simple hyperedge connecting three nodes.

When it comes to notation, the network is still represented by the classical node and edge sets: $G = (V, E)$. However, the E set now is special: its elements are not forced to be tuples any more. They can be triples, quartuplets, and so on. For instance, (u, v, w) is a legal element that can be in E, with $u, v, w \in V$.