Dynamics of networks: generation, dimensionality reduction, and coarse-grained evolution of graphs

Proposed by Alexander Holiday

under the supervision of

Professor Yannis Kevrekidis

11/01/2013

1 Introduction

From collaborations among movie stars [?] to gene interactions in C. elegans[?], network science has found a vast array of applications in the past two decades. This is largely a result of the generality of the network framework: myriad systems are readily adapted to description as interacting bodies; the body (e.g. a city, person, or protein) forms a node in the network, and the interactions between them (e.g. via highways, Facebook friendships, or biological suppresion) create the edges. Thus, one may usefully apply the same abstraction to study such disparate topics as coupled oscillators and the spread of opinions in a society.

However, too often this merely leads to a recasting of the original problem in a new light. [Some examples]. While this, in itself, is useful, it fails to exploit the true power in such a formulation. There are several reasons for this, a selection of which will be the focus of this proposal. Broadly speaking, the generality of this construct is in some sense its undoing: while many problems may be addressed through it, they require many different types of analysis. Thus, a researcher in neural networks may use none of the same tools as a civil engineer designing tranporation networks. Certainly, the dynamics between proteins in a cell and drivers in rush-hour traffic may share almost no similarties; however, this should not prevent them from having similar analytical tools at their disposal.

One of these tools may be considered a foundation upon which all others rest: the ability to computationally construct networks with desired properties. There are several examples in which, during the course of analysis of a system, incorrect conclusions have been drawn due to improper modeling of the network itself [?, ?, ?]. For instance, early network research into the internet tended to model its structure in the following, simple manner: each webpage (a node of the network), was randomly connected to each other webpage with some given probability p (creating what is known as an Erdős-Rényi random graph, detailed below) [?]. The simple structure lent itself to easier calculations, but as future papers emerged detailing the web's actual structure, these early results were dismissed [?, ?]. As the intimate link between a network's underlying structure and its resulting dynamics becomes increasingly evident [?, ?], it becomes difficult to justify the use of simple models in lieu of the ability to construct more accurate versions (or at the very least one cannot fully trust any subsequent results). Unfortunately, there is currently a lack of methods available to researchers for the construction of networks with desired properties. The algorithms that do exist tend to address one property of the network (e.g. the average number of connections a node has), leaving others unspecified. In fact, in collaboration with the Floudas lab, a general network generation method capable of simultaneously stipulating multiple properties has recently been developed; but, while a promising step forward, it fails to construct larger networks in feasible times [?]. Therefore, there remains a need for a general algorithm capable of generating a network with arbitrary specifications.

Even if one succeeds in accurately modeling a network's structure, the resulting data can be unwieldly and overwhelming. Consider a simple example: the religious affiliation of Facebook members. Our data consists of a list of all Facebook users, the links between them ("friendships"), and the declared religion of each member (let us assume everyone makes such information public). Given this informaiton, an obvious question would be: what is the driving factor in an individual's religious association. Initially, reasoning that those with many friends of a certain relgion will tend to follow suit, one might correlate religion with the fraction of friends who share the same. However, it could also be reasoned children tend to align themselves similarly with their parents. Studying this effect in isolation is feasible, but a method of combining this effect with that of the individuals friends isn't obvious. Perhaps, too, there are certain highly connected individuals who vocally advocate their religion, causing many of their friends to convert (e.g. pastors, rabbis, priests, etc.). Here, even the proper identification of such individuals is unclear. We have a static data set, in which only one property has been measured at each node, yet there are many possible methods of analysis (some easier than others), each potentially leading to very different conclusions. Considering most data sets evolve in time (are dynamic) and many specify more than one property at each node, a systematic method of analyzing such data would be a great help in guiding research efforts.

Many tools designed for general dimensionality reduction of high-dimensional data can be applied here (for examples, see [?] [?]), but there certainly remains much room for improvement. A popular method of "community detection" (grouping like nodes) is currently "quality scoring". This has been successfully applied to several systems ([?] [?]), but it requires certain assumptions about a proper "null network" for the system. Other methods require the specification of certain algorithmic constants whose relevance to the underlying problem is unclear. The search for a general method of clustering continues. An even more general pattern recognition approach would be ideal.

Finally, the ability to create specific networks on demand can be combined with a method of determining important network characteristics to perform coarse projective integration on an evolving system and accelerate simulation times. As the initial network evolves, the data mining technique would be used gather properties that describe the network in $m \ll n$ variables. If the evolution of these macroscopic variables was smooth enough, they could be projected forward, as shown in Schematic [?]. A new network adhering to the projected values would be created using the network generation algorithm, and evolution would continue on the microscopic scale. This process would be iterated until the system reached a desired state, likely some equilibrium configuration

The first two aspects of network simulation detailed above, network generation and dimensionality reduction, can be viewed as independent aims. However, the third, simulation acceleration through coarse projective integration, is very much tied to our ability to address these first two. The details of each will be elaborated below. First, the notation used throughout the proposal will be specified.

1.1 Notation

A graph, G, is defined by a set of vertices (or nodes), $V(G)$, and the connections (or edges) between them, $E(G)$. The size of the network, $n(= |V(G)|)$, is the total number of nodes, while the total number of edges will be represented by $m(= |E(G)|)$. A single vertex, v_i, is connected to another vertex, v_j if and only if the corresponding edge, e_{ij} is non-zero. An edge e_{ii} is called a loop. If $e_{ij} = e_{ji} \forall i, j$ then the graph is undirected (i.e. if i is connected to j, j must be connected to i, e.g. Facebook friendships), otherwise it is directed. In many cases, the edges take binary values, $e_{ij} \in 0, 1$, and we call the graph unweighted. Otherwise we deal with weighted graphs in which the edge value may take any positive value, typically signifying the strength of connection between v_i and v_j. No strict definition of a network exists. Some use the term when referring to weighted graphs; in the following, "graph" and "network" are used interchangeabley.

There are many ways of specifying a graph, the simplest being a list of all v_i and e_{ij} present. However, the most popular method of description is through an "adjacency matrix". This is a square, nxn, matrix in which $A_{ij} = e_{ij}$. This form is especially nice when the underlying graph is undirected ($e_{ij} = e_{ji}$) as the adjacency matrix is symmetric. The degree of a vertex is an important measure of its connectedness in the graph, and is given by $d_i = \sum\limits_{i=0}^{n} e_{ij}$, thus in an unweighted graph, the degree of a vertex is simply the number of edges connected to it (with loops counted twice). With this foundation, we now continue with the original material.

1.2 Network generation

Significant effort has gone into the production of algorithms that will create networks with different properties. By far the most popular is the Erdős-Rényi random graph model. Proposed in 1959 [**?**], it consists of the following simple procedure: given a probability of connection p, examine each possible edge in the graph (note there are $\binom{n}{2}$ possible edges in an undirected graph) and, with probability p, let $e_{ij} = e_{ji} = 1$, otherwise $e_{ij} = e_{ij} = 0$. Simply put, each edge exists with probability p. As previously mentioned, the simplicity of the method allows for a great deal of theoretical calculations [**?**], but, as one might guess, few real-life systems follow such a simplistic scheme. The need for new algorithms became clear.

To-date, many different methods of graph generation have been proposed. They largely fall into two categories: those which describe an evolution of the graph to its final state, and those which generate graphs with specified properties. A famous example of the former is the scale-free preferential attachment model [**?**]. The method is as follows: begin with a small number of disconnected vertices, n_0. Add a new vertex, and connect it to $m \leq n_0$ of the pre-existing vertices. Let the probability that the new vertex, v_{new} connects to one of the old vertices v_{old} be $p = \frac{d_{old}}{\sum\limits_i d_i}$. This method of attaching new edges leads to a rich-get-richer effect observed in many real-life situations. Note that no graph properties have been specified *apriori* (except, trivially, the size n). In contrast, consider the Erdős-Rényi model just described. Here, the average degree, $\bar{d}$ is specified beforehand (this is equivalent to setting p, as then $p = \frac{n\bar{d}}{2\binom{n}{2}}$), and a graph is constructed to match this stipulation. While the method of evolving a graph to its final form is useful when this growth mechanism is understood, these models are specialized to the system under investigation. When this information isn't available (as if often the case), it is more useful to measure certain features of the network in question, and use the property-specification class of methods to generate a graph with similar characteristics.

The number of such methods, and the variety of properties addressed by them, has grown steadily in recent years. From efficient algorithms to generate Erdős-Rényi random graphs, to the something-something algorithm that allows for the simultaneous assignment of degree distribution and degree-dependent clustering [?] [?], a wide range of variables are tunable in the current collection of methods. The shortcoming of all but one of the current methods in the current toolbox is that each deals with only a small number of specific properties. If the network you'd like to model has a certain degree distribution, and specific degree-triangle correlations, you must hope that an algorithm has been developed to deal exactly with the creation of a graph with a certain degree distribution and degree-triangle correlation. One cannot simply take a method for degree distributions and combine it with a degree-triangle method to achieve the desired outcome.

A new, innovative approach to this problem, the product of collaboration between the Floudas and Kevrekidis groups, aims to alleviate this hindrance. Their approach, detailed in [?], is to formulate the problem as a task for linear optimization. Using this optimization framework, they're able to add property specifications to the algorithm as needed. Each variable that needs to be addressed can be added as a building block in the overall algorithm, tunable to the unique needs of each graph. Additionally, this method can guarantee the non-existence of certain graphs, a useful feature when detailing multiple complex properties. The downside of this wonderful generality is computational slowness. Searching the solution space of all possible graphs can become a daunting task at even modest sizes of $n = 15$ if the searched-for graph is highly detailed. Using certain pre-processing steps, the speed has been greatly increased, but work remains if the method is to be applied to larger systems of $n > 100$.
Certainly, each algorithm has its unique strength and weaknesses. While some can rapidly construct very large graphs ($n > 100,000$), these have been created to address only a few of the many potentially interesting properties of a network. The one method that allows users to add properties as desired suffers from scalability issues. The efficient creation of networks with arbitrary property specifications remains an open area of investigation.

1.3 Data Mining in Networks

In the past decade, several factors have combined to fuel a recent wave of algorithms which aim to make sense of massive data sets. The internet has played no small role, as millions of users give information to online websites wittingly (through user accounts) or not (through cookies). Additionally, the proliferation of electronic sensors in everything from cars to refridgerators [?] has given companies a far greater variety and quantity of information than before. Crucially, too, the cost to store all this has steadily decreased. Here enter data mining techqniues, used to extract useful conclusions from such massive volumes of data. From the ancienct workhorse of principle component analysis (PCA), useful in finding data embedded in linear subspaces, to new, nonlinear manifold learning techniques such as diffusion maps, the techniques are as varied as the data they operate on (if less numerous) [?]. However, these methods are not currently well-suited for data mining in networks. Generally speaking, the goal of a data mining algorithm is to take a collection of points in $\mathbb{R}^n$, $n \gg 1$ and embed it in $\mathbb{R}^m$, m, where the m new dimensions would capture the important details of the data. In many cases, data is easily amenable to such a description (e.g. a vector {*age, longitude, latitude*} for Amazon users). Unfortunately, when each data point is, itself, a network, there is no clear way to “vectorize” the data, and thus the host of techniques proven in other applications become useless. The two options are then to develop new techniques that specifically address the unique aspects of networks data, or to devise a scheme that adapts established techniques to operate on graphs (be it vectorization or otherwise). We focus on the latter approach.

Many existing methods employ some measure of distance between points in their formulation. This not always straightforward (e.g. if the data includes gender, a quantitative distance is unclear), but some *L-norm* is often suitable. One worthwhile avenue of investigation would seem to be creating a similar distance measure between graphs. This involves determining whether data representing two different graphs actually describe different underlying structures, known as the graph isomorphism problem.

1.3.1 Graph Similarity and Isomorphism

To understand the issue at hand, the distinction between labelled and unlabelled graphs must be made clear. A labelled graph is one in which the vertices have been assigned a unique identification. In the case of a transportation network of highways between cities, the city names could function as labels. Then, if tasked with determining whether U.S. roadways changed between 1980 and 1981, simply checking whether each city was connected to the same places would suffice. The vertices of an unlabelled graph have no such intrinsic identity. (Need better example!) Consider the popular epidemiological SIR model, in which vertices represent people and edges connections between them by which diseases can be transmitted. Here, each node lacks information beyond the fact that it represents a single person. Comparing two networks is no longer so easy as checking whether all connections to and from each node are the same, as there is no way to identify a node in the first graph’s equivalent in the second. To be sure two unlabelled graphs are, indeed, different, all possible pairings of nodes from graph one and two must be considered. This is the NP hard problem of determining graph isomorphism (whether two graphs can be labeled in such a way as two make their connections equivalent).

Given the inability to tell even whether two graphs are the same (in reasonable computational time), the difficulties in defining a computationally tractable distance measure can be understood. Short of attempting a Clay Millenium Prize, one possibility is to accept an imperfect method with guaranteed error bounds. While there is a wealth of research into isomorphism problem, this sort of error-bound approximation has yet to be researched. The hope lies in the possibility of using such an approximation to provide some quantitative distance between graphs, which, in turn, could be used in pre-existing data mining routines. While much work has covered mining of data in a network, there is as yet nothing regarding mining data of networks.

1.4 Accelerated Network Simulations

Real-life networks are often many-membered ($n > 10,000$) and frequently exhibit complex dynamics between nodes (e.g. a differential equation describing a flow along an edge). Modeling such systems is a challenge in itself, but even if models are available, simulating such large, sophisticated networks is computationally prohibitive. In a number of cases, the time required to reach a steady-state of the system is a function of the problem size [?] [?] [?]. Thus, both storage space and simulation length increase with n. A method of accelerating such laborious calculations would be highly valuable. A coarse projective integration scheme could enable such a speedup, but this would require a good macroscopic description of the network, along with some method of translating these macroscopic variables into realizations of a microscopic system. Advances in the previous two sections would allow for both.

Coarse projective integration (CPI) exploits the separation of timescales inherent in many systems to expedite simulations. The framework is very general, and has been successfully applied to many systems [?]. There are two prerequisites: that there exist a small number of good, coarse variables that adaquately describe the system (but for which closed equations or models are not available), and that, given certain macroscopic property specifications, a microscopic system can be created that exhibits these characteristics. Given these, a CPI method can be implemented using the following general outline: first, detailed, microscopic simulations are initialized, and the selected coarse variables are periodically calculated as the system evolves. Given this evolution of coarse variables, one then projects the system forward in the coarse, low-dimensional space to new values of coarse variables. Finally, these new variables are used to restart full microscopic simulations, at which point the process can be iterated until the desired state has been reached.

CPI has already been applied to evolving networks, but these implementations have depended on two things. First, that suitable coarse variables could be discerned from careful observation of the system, and second, that there existed an algorithm to generate graphs with such coarse variables [?] [?]. Demonstrating that either a data mining algorithm could be employed to find such coarse variables or that a generic network generator can initialize the necessary microscopic simulations would significantly generalize the CPI approach for use in a wider variety of network problems. Combining the two (network generation and pattern identification) would be the ideal culmination of this work.

Graph generation, detection of signficant features across networks, and accelerated simulation of such complex systems are each interesting avenues of research in their own right. It is hoped that the synthesis of these three elements will allow researchers to more accurately model real-life networks, attain useful information from such models, and simulate full-scale systems in reasonable computational times.

2 Current Work

Initial research has focused on accelerating simulations of dynamical network systems. While leaping to the third of three steps may seem premature, it quickly reveals areas for improvement in the process, and serves as a guide for future investigation. Two cases were studied, a voting model with possible applications in sociology, and an edge reconnecting model, which mainly serves as a toy mathematical network for which behavior can be derived theoretically. After briefly describing the dynamics of each, current progress in simulation acceleration will be discussed.

2.1 Voting Model

The k opinion voter model initally consists of an Erdős-Rényi graph of size $n(> 10,000)$, with small average degree ($\bar{d} \asymp 1$). Given some initial distribution of the k different opinions $\{p_i\}_{i=1}^{k}$, each vertex is randomly assigned an opinion such that $P(\zeta(v_i) = k) = p_k$ where $\zeta(v_i)$ denotes the opinion of v_i. With this initial state and a "probability of reattachment" α, the graph evolves as follows:

1. Choose an edge $e_{ij} \in E(G)$ uniformly at random
2. Randomly assign one of the endpoints of e_{ij} to be v_i
3. If $\zeta(v_i) = \zeta(v_j)$, repeat step one
4. With probability α, remove e_{ij} from the graph
   (a) Choose a new vertex $v_k \in V(G)$ uniformly at random
   (b) If $e_{ik} \in E(G)$, choose another vertex
   (c) Let $E_{new}(G) = E(G) \cup e_{ik}$
5. Otherwise (with probability $1 - \alpha$), set $\zeta(v_i) = \zeta(v_j)$

This process is iterated until the system reaches a consensus state in which $\zeta(v_i) = \zeta(v_j) \forall v_i, v_j \in V(G)$, i.e. every edge connects vertices of the same opinion. An edge e_{ij} is called a conflict whenever $\zeta(v_i) \neq \zeta(v_j)$. Thus the system is evolved until all conflicts are removed. While the general k opinion model has received some attention [**?**], our focus will be the simpler two opinions (0 and 1). In this case, it has been shown that, for a certain initial minority fraction $p_{minority} \leq 0.5$, a bifurcation occurs as α increases. At low values of α, fewer edges are rewired and the system approaches a static state. In this limit, the system should converge approximately once each of the m edges are selected, which occurs in $O(n\sqrt{n})$ steps (according to results from the coupon collector problem). As α increases, so does the frequency of rewiring. This slows the rate of consensus to $O(n^2)$. Interestingly, besides differences in convergence speed, the final consensus state also changes. In the low-α limit, the final minority fraction is an increasing function of α. In the slower, high-α range, the final minority fraction is unchanged from the initial value. Fig. **??** illustrates the effect of α and $p_{minority}$ on the final consensus state.

By creating phase plots of different system properties (e.g. minority fraction, number of conflicts), it was observed that the minority fraction dictates the state of the system. Fig. **??** shows how both the number of conflicts and the number of connected vertex triplets with two vertices of opinion 0, and one of 1, could be written as functions of the minority fraction. This suggested that the system minority fraction could serve as a macroscopic variable in CPI. As Fig. **??** shows, the evolution of the minority fraction is not monotonic. If data is collected haphazardly, it would be possible to project the system away from its steady state. To alleviate this issue, an ensemble of voting models is created. The average trajectory of the minority fraction can then be used safely, as the expected evolution is one of decreasing minority fraction. In fact, both the minority fraction and conflict count were used as coarse variables (why?). After projecting these variables forward in macroscopic space, the graph generation problem is confronted. Thankfully, creating a graph with a certain minority fraction and conflict number is a simple task, and a customized algorithm was developed. CPI results are shown in Fig. **??**. This CPI implementation reduced the number of microscopic steps needed to reach consensus by an average of 50% (need actual number).

2.2 Edge Reconnecting Model

The edge reconnecting model, proposed in [**?**], presents special difficulties in creating accurate low dimensional representations of the overall system. Namely, the network is allowed to have multiple edges connecting the same two vertices (such a construction is termed a "multigraph"). The initial configuration is a graph with $m \asymp n^2$ edges distributed uniformly among the n vertices. The multigraph evolves as a markov chain according to the following dynamics:

1. Choose an edge $e_{ij} \in E(G)$ uniformly at random and flip a coin to label one of the ends as v_i
2. Choose a vertex v_k using linear preferential attachment, i.e. $P(v_k = v_l) = \frac{d_l}{\sum\limits_{i=1}^{n} d_i}$
3. Replace e_{ij} with e_{ik}

This process is repeated until a frozen state is reached, at which point the degree distribution ceases to change.

Two distinct timescales arise from these dynamics, $T \asymp n^2$ and $T \asymp n^3$ where T is the number of steps. On the faster, $O(n^2)$ scale, the degrees of the vertices may be considered constant, while the number of parallel edges between vertices changes. On the slower $O(n^3)$ scale, the degree distribution evolves to a steady state value. While this separation of timescales has been proven in [**?**], identifying them through numerical simulations is complicated by a couple of factors. First, the exact timescales themselves are difficult to discern. Both scales are really $O(\rho_1 n^2)$ and $O(\rho_2 n^3)$, where the constants ρ_i are evaluated at the limit of infinite-sized graphs ($n \to \infty$). This hints at the second, larger, problem: many of the results on the existence of these timescales in the first place are only valid in this large-n limit. Simulation time then becomes problematic. Figs. **??** and **??** illustrate attempts to visualize these separate scales of evolution. The degree distribution is plotted every n^2 steps in thse figures, with a total number of n^3 steps in each. The changes appear quite gradual, and no distinct timescales are evident.

Our approach in coarse-graining system dynamics is based on the existence of a gap in the spectrum of the adjacency matrix, and the subsequent ability to approximate $A \approx \lambda_1 v^{(1)} v^{(1)\,\dagger}$ where A is the adjacency matrix of the system, λ_1 is the leading eigenvalue of A, and $v^{(1)}$ the corresponding eigenvector. Fig. **??** shows that clearly $|\lambda_1| \gg |\lambda_i|, i = 2, 3, ..., n$.

Fig. **??** illustrates the reconstruction of the adjacency matrix as $A_{i,j} = \lambda_1 v^{(1)}_{i,j} v^{(1)\,\dagger}_{i,j}$ (after multiplication, each entry $A_{i,j}$ was rounded to the nearest integer if $i \neq j$ or to the nearest even integer if $i = j$). Visually, the two correspond very well.

As $\lambda_1 v^{(1)} v^{(1)\,\dagger}$ appears a good approximation of A, it was reasoned that we could use this eigenvalue/eigenvector combination as a coarse description of A. In order to further reduce dimensionality, the eigenvector was fitted with a fifth-degree polynomial, as shown in Fig. **??**. The six coefficients of this function were then used as a smaller set of coarse variables, leading to a final seven-dimensional representation of the system (six coefficients plus an eigenvalue). The following outlines the CPI framework:

1. Simulate the full edge reconnecting model dynamics for some number of steps until the fast variables are sufficiently slaved to the slow
2. Record the adjacency matrix as the system evolves on the slow manifold (in fact, it is more efficient to immediately compute the leading eigenvector, fit it with a polynomial, and store only these coefficients, along with the leading eigenvalue, as time progresses)
3. Project forward the coarse variables (coefficients and eigenvalue)
4. Reconstruct a new adjacency matrix from the new, projected coefficients and eigenvalue:
   (a) Compute a new eigenvector as $v(i) = \sum\limits_{k=0}^{k=6} i^k c_k$ (where c_k represents the coefficients of the polynomial and $v(i)$ the i^{th} component of v) and round to the nearest integer
   (b) Compute the new adjacency matrix as $A_{i,j} = \lambda_1 v^{(1)}_{i,j} v^{(1)\,\dagger}_{i,j}$ and round as discussed previously
5. Repeat step one until system reaches steady state

Preliminary results of this method are shown in Fig. **??**, in which the evolution of the degree distribution is shown for both the full simulation and a simulation in which CPI has been employed. On average, CPI required 50% fewer steps (actual number needed) than the full simulation.

3 Recent Advances in Literature

4 Future Work

A number of areas present themselves as good research directions. They're divided by subject below.

4.1 Graph Generation

The main limitation of the optimization based algorithm is its scalability. Preprocessors have been shown to significantly reduce search times, but have been created to address specific properties. They operate by removing search paths unlikely to yield the specified graph before the optimization algorithm begins. Generalizing this preprocessor to arbitrary properties would retain the generality of the method while increasing speed (how would this

actually be done in process? need to understand the actual preprocessing process)
Work on developing graph similarity measures in simple models, such as Erdős-Rényi, where guarantees of success or failure may be easier to derive.

4.2 Data Mining in Graphs

Continue on work of Karthik's second prop.

4.3 CPI

Try to apply diffusion maps to voting model data to see if it can recover the minority fraction as a good coarse variable.