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 [1] to gene interactions in C. elegans [5], 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 such a description; interacting bodies (e.g. a city, person, or protein) form nodes in the network, the connections between them (e.g. via highways, Facebook friendships, or biological suppresion) create edges. Thus, one may usefully apply the same abstraction to study such disparate topics as the spread of opinions in a society and chemical reaction networks.

However, too often this merely leads to a recasting of the original problem. While this, in itself, can be useful, it fails to exploit the true power in such a formulation. Several obstacles prevent this full realization, a selection of which will be the focus of this proposal. Broadly speaking, the generality of the construct is in some sense its undoing: while numerous problems are addressable, 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 similarities; however, this should not prevent investigators in these fields 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 were drawn due to improper modeling of the network architecture itself [**?**, **?**, **?**] [10]. As the intimate link between a network's underlying structure and its resulting dynamics becomes increasingly evident [1] [2], 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 a single network variable (e.g. the average number of connections a node has), leaving others unspecified. Therefore, work remains in creating more general algorithms capable of generating networks with a wide range of structures.

Even if one succeeds in accurately modeling a network's structure, the resulting flood of data can be overwhelming and unwieldily to analyze. To address this problem, a class of techniques have been formulated to extract various properties specifically from network systems. These range from **find two actual network analysis techniques** community detection algorithms that suggest groupings of similar nodes, to pattern recognition methods that search for specific motifs in network systems [11] [8] [9] [**?**]. While these have been effective tools in understanding network structure, their applicability is confined to analyzing single networks. An avenue that has received relatively little attention, and one especially relevant to the study of dynamic systems, is formulating techniques that operate across networks. Extracting information from a set of networks presents special challenges, but as the number of temporally-resolvable systems increases, so, too, does the importance of such an ability. Traditional data-mining algorithms such as principal component analysis (PCA), diffusion maps (DMAPS) and nonlinear embedding, can be considered a generalization of the network-analysis tools described above, reducing massive datasets to their significant low-dimensional representations. While more broadly applicable, they currently require that input data be in vector form. Extending these methods to operate on sets of *graphs* instead of vectors has the potential to automate the discovery of important parameters governing the evolution of dynamic network systems, revealing their coarse descriptions. This would be a tremendous step forward in network analytics.

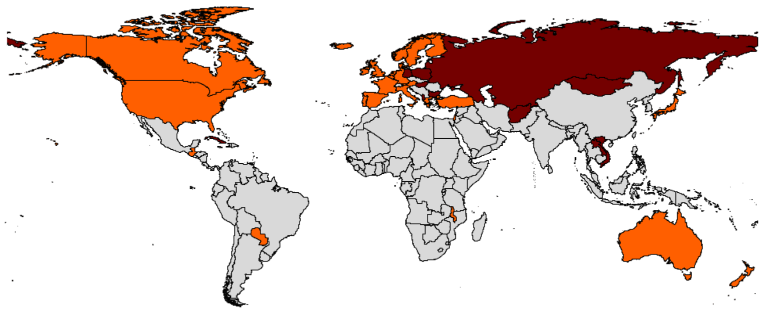


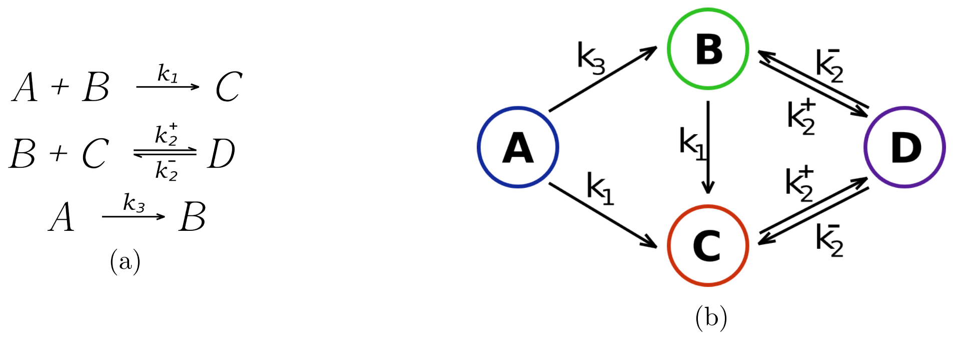
Figure 1: Grouping of countries based on United Nations voting records using the modularity method [**?**].

Developing methods to create specific networks on demand or to determine important system characteristics is, itself, a valuable aim; however, by combining advances in both abilities, we hope to enable a variety of coarse system analysis through the *equation free* (EF) *framework*. EF modeling depends on a key component called the coarse time-stepper, detailed below. In brief, the coarse time-stepper requires a method to a.) discover appropriate macroscopic variables describing system evolution and b.) translate these coarse values into to a fine scale, full simulation. With the data-mining and graph generation techniques to be discussed, we hope to have methods accomplishing both. After implementing the coarse time-stepper, an array of EF techniques can be realized, from coarse projective integration to optimization and design. In a field where networks may be composed of billions of nodes, the ability to examine behavior through a low-dimensional system is crucial in reducing simulation times and allows a fuller analysis of dynamics such as fixed point bifurcations and stability.

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-world networks, attain useful information from such models, and simulate full-scale systems in reasonable computational times. 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 is 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 that begins and ends at the same vertex, e_{ii}, is called na 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. A reaction network in which edges represent differential equations governing interactions between molecules would be a weighted, directed graph (the interactions between different particles could be different, and particle A's influence on B does not imply a reciprocal influence by B on A). No strict definition of a network exists. Some use the term when referring to weighted graphs; in the following, "graph" and "network" are used interchangeably.



$A + B \xrightarrow{k_1} C$
$B + C \underset{k_2^-}{\overset{k_2^+}{\rightleftharpoons}} D$
$A \xrightarrow{k_3} B$
(a)
A
B
C
D
k_3
k_1
k_1
k_2^-
k_2^+
k_2^+
k_2^-
(b)

Figure 2: Traditional chemical reaction equation (2a) and its equivalent reaction network (2b).

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, $n \times n$, matrix in which $A_{ij} = e_{ij}$. This form is especially helpful 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 $d_i = \sum_{i=0}^{n} e_{ij}$; 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 continue with the original material.

2 Network generation

Significant effort has been spent researching algorithms that create networks with different properties. By far the most popular method creates what is called an Erdős-Rényi random graph. Proposed in 1959 [6], it consists of the following simple procedure: given a set of vertices V and 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. This straightforward method facilitates theoretical calculations [?], but, as one might guess, few real-world systems follow such a simplistic scheme.

Many different methods of graph generation have been proposed to more accurately capture network structure. 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 [1]. 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 $P(e_{v_{new}v_{old}} = 1) = \frac{d_{old}}{\sum\limits_i d_i}$. Note that no graph properties have been specified *a priori*, except, trivially, the size n. In contrast, consider the Erdős-Rényi model just described. Here, the average degree, $\bar{d}$, is specified beforehand, and a graph is constructed that adheres to this stipulation (note that the average degree and the probability of attachment, p, are dependent through $p = \frac{n\bar{d}}{2\binom{n}{2}}$). While the method of evolving a graph to its final form is useful when the underlying growth mechanism is understood, these models are specialized to the system under investigation. When information on the evolutionary process 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 procedures that generate Erdős-Rényi random graphs, to those that enable the simultaneous assignment of degree distribution and clustering coefficients [3] [4], a wide range of variables are tunable in the current collection of methods. The shortcoming of nearly all of the approaches in this toolset is that each addresses only a small number of specific properties. If the network under consideration 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 given 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.

An innovative approach to this problem, the product of collaboration between the Floudas and Kevrekidis groups, aims to alleviate this hindrance. Their technique, detailed in [7], is to formulate the problem as a task for mixed-integer linear optimization. Using this optimization framework, they are 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 formulation, 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 strengths and weaknesses. While some can rapidly construct very large graphs ($n > 100,000$), these have been created to address only a

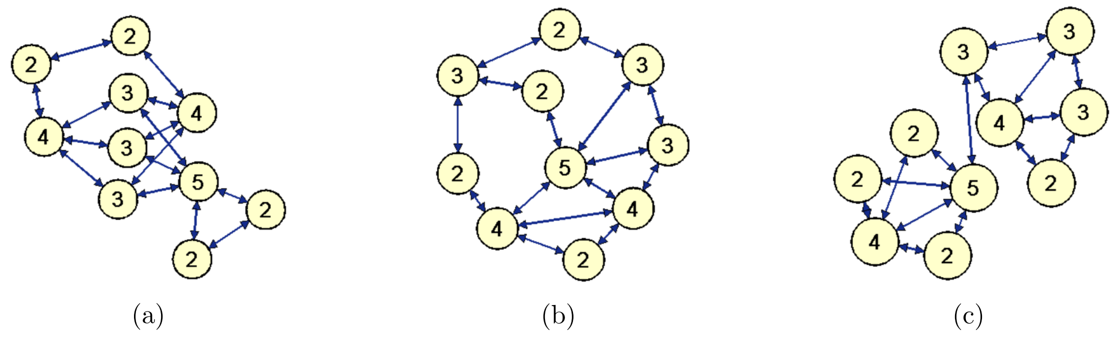


Figure 3: Networks generated by the optimization framework, taken from [**?**]. The distribution of degrees is identical in both, but the clustering of the nodes was increased from 3a to 3c.

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.

3 Data Mining across Networks

In the past decade, several factors have combined to fuel a wave of algorithms designed to analyze 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 access to a far greater variety and quantity of information. Crucially, too, saptiotemporal resolution of research experiments has become increasingly commonplace. In these scenarios, data-mining techniques are invaluable in extracting useful information from such massive volumes of data. From the original workhorse of PCA, useful in finding data embedded in linear subspaces, to new, nonlinear manifold learning techniques such as DMAPS, today's techniques exhibit a wide range of approaches to dimensionality reduction [**?**]. However, as mentioned, these methods are not currently well-suited for data mining *across* networks.

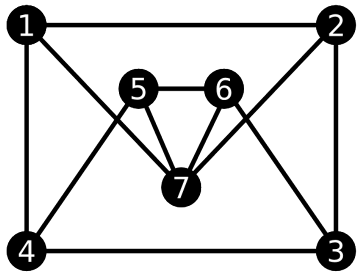
Generally speaking, the goal of a data mining algorithm is to take a collection of points in $\mathbb{R}^n$, $n \gg 1$ and embed it in $\mathbb{R}^m$, $m \ll n$, where the m new dimensions capture the important details of the data. In many cases, data is easily amenable to such a description, e.g. a vector $\{$*age, longitude, latitude*$\}$ for Amazon users. Unfortunately, when each data point is, itself, a network, there is no clear way to initially "pre-embed" each point into $\mathbb{R}^n$, 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 network data, or to devise a scheme that adapts established techniques to operate on graphs, be it through some pre-embedding or otherwise). We focus on the latter approach.

Many existing methods employ some measure of distance between points in their formulation. For example, a popular implementation of diffusion maps requires a weight matrix

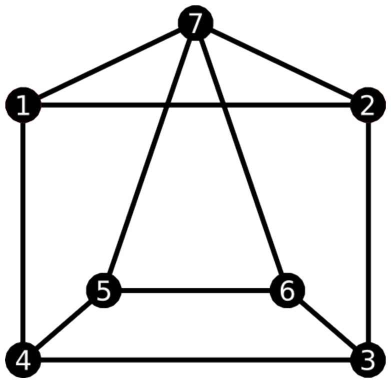
$W_{ij} = e^{-\frac{\|x_i - x_j\|^2}{\epsilon}}$, in which x_i and x_j are members of the dataset. Attempting to generalize vector or matrix norms might seem a natural place to start in defining distances between two graphs. As a norm, we would expect $\|x - y\| = 0 \Leftrightarrow x = y$. Unfortunately, even determining whether two graphs are equivalent is a complete research puzzle unto itself, and is termed "the graph isomorphism problem".

3.1 Graph Isomorphism

To understand the issue at hand, we must first discuss the distinction between labelled and unlabelled graphs. 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, say, tasked with determining whether U.S. roadways changed between 1980 and 1981, simply checking whether each city's connections remained the same would suffice. The vertices of an unlabelled graph have no such intrinsic identity. Consider the popular epidemiological SIR model, in which vertices represent people and edges interactions 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 from the first graph, v_{old}, with an equivalent node, v_{new}, 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** problem of determining graph isomorphism (whether two graphs can be labeled in such a way as to make their connections equivalent). Formally, two graphs G and H are isomorphic if there exists a function $f : V(G) \to V(H)$ such that $v_i \in V(G)$ and $v_j \in V(G)$ are adjacent if and only if $f(v_i) \in H(G)$ and $f(v_j) \in H(G)$ are also adjacent.



(a)



(b)

Figure 4: Different drawings of identical (isomorphic) graphs.

While a general polynomial-time solver is considered impossible by most as it would prove $\mathbf{P} = \mathbf{NP}$, approximation algorithms with guaranteed error do exist. In fact, polynomial-time solutions are also available, but only for restricted classes of graphs.

3.2 Graph Similarity

Given the inability to tell even whether two graphs are the same (in reasonable numerical time), the difficulties in defining a computationally tractable distance measure are understandable. A novel approach to this issue would be to build off existing approximation algorithms. The hope would be to extend one of these methods to output some indication of the differences between graphs, and not just a boolean judgement of ismorphism. This modified method would then be used as the requisite distance measure in pre-existing data mining routines. This would be an innovate compuatation of graph similarity, and is worthy of investigation. If this fails, alternative methods have been established in the literature. These are detailed below.

3.2.1 Edit Distance

This class of methods is founded on the idea that a graph G is similar to a graph H if G requires few additions and deletions of vertices and edges to become H. Basically, each type of "edit", be it adding a vertex, deleting an edge, or switching a vertex label, is assigned a cost. Then, the total cost of obtaining H from G defines their similarity. Unfortunately determining the changes in G required to obtain H is computationally infeasible, so multiple approximations have been developed. These range from neural-network based frameworks to modeling the edit operations as random phenonema with specified probability distributions. However, a major hurdle in implementation remains the definition of the cost function, as different weighting will yield inconsistent measures of similarity.

3.2.2 Maximal Common Subgraph

Here, the distance between G and H is defined as

$$d(G,H) = 1 - \frac{|mcs(G,H)|}{max(|G|,|H|)}$$

where $mcs(G,H)$ is the maximal common subgraph between G and H. That is, the largest subgraph of G isomorphic to a subgraph of H. This straightforward formula doesn't require the subjective cost function of the edit distance, but it is ***CHECK**** **NP-Complete**. A combination of this method and edit distance has been proposed that may yield a more computationally feasible metric [**?**].

3.2.3 Graph Kernels

This class of methods has a solid theoretical basis that has been extended into a few different flavors. The requirements for a kernel $k(x,x')$ is that k is symmetric and positive semi-definite. Thus, the requirements of metrics are relaxed, allowing more freedom in a kernel's formulation. All existing methods are based on random walks over graphs, and subsequent comparison of the traversed paths. These random-walk algorithms have recently been shown to take a general form

$$k(G,H) = \sum_{k=0}^{\infty} \mu(k) q_{\times}^{T} W_{\times}^{k} p_{\times}$$

where q W and p need to be explained. Defining kernel methods outside of random-walks is a promising direction.

4 Equation Free Modeling

Dynamic network models prescribe behavior at the level of individual vertices and edges, enabling researchers to construct systems with an incredible level of detail. However, while we seek to understand the long-term macroscopic dynamics of the networks, explicit, accurate equations governing this system-level evolution are typically unavailable or poorly understood. To circumvent this need for macroscopic equations in system-level analysis, we turn to *equation free* (EF) modeling. However, there are two prerequisites to implementing the EF framework: an appropriate 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 fulfill both.

4.1 Coarse time-stepper

An integral component in other EF methods, the coarse time-stepper provides the evolution of coarse variables over a short time interval. This is accomplished by controlling the initiation of short periods of fine-scale simulations. Define the high- and low-dimensional spaces as F and C (fine and coarse). To move between these two levels of system descriptions, we define the restriction operator $\mathbf{R} : F \to C$ (e.g. an extended DMAPS method) and the lifting operator $\mathbf{L} : C \to F$ (e.g. some graph generation algorithm). Then, starting from some initial state $u(t_0) \in F$, the coarse time-stepper proceeds as follows:

1. Run the fine-scale simulation till the system evolves on the slow manifold and high-dimensional variables are coupled to the coarse descriptors. Behavior can now be described with the low-dimensional variables in C.

2. Continue fine-scale simulation, using $\mathbf{R}$ to record n collections of coarse variables at intervals of δt.

This provides a time-series of macroscopic variables $\{\mathbf{R}(u(t + k\delta t))\}_{k=0}^{n}$. With the ability to investigate low-dimensional dynamics at will, more powerful EF analysis becomes available. **add something about ensembles of systems**

4.2 Coarse projective integration

Coarse projective integration (CPI) exploits the assumed smoothness of the coarse variable's evolution to accelerate simulations. The coarse time-stepper is used to record the macroscopic variables trajectory over some time interval. These points are used to form numerical derivatives of variables' change with time, which, in turn, can be used with standard integration routines, such as Forward Euler, to project these values across C. The new set of variables, $g_{new} \in C$ is used to initialize fine-scale simulations at $u = \mathbf{L}(g_{new})$. This procedure is iterated until the desired system state is reached. By projecting forward the few variables evolving smoothly on the system's slow manifold, expensive, full simulation is significantly reduced. The result is a steeply improvemed in computation times.

4.3 Coarse fixed point calculations

Given a macroscopic state $g_n \in C$, let $g_{n+1} = \Phi_T(g_n)$ be the state of the system evolved for time T starting at g_n. A fixed point could be evaluated by locating $g_n$such that

$$F(g^*) = g^* - \Phi_T(g^*) = 0$$

Typically, as in Newton-Raphson iteration, the derivatives $\frac{dF}{dg}$ would be needed to update the solver. While certainly not available analytically, these derivatives can be approximated numerically. This is an effective but expensive task, as it requires the repeated use of the coarse time-stepper. Using fixed point methods like the generalized minimization of residuals (GMRES) would improve performance [**?**].

4.4 Coarse bifurcation analysis

5 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 weaknesses in the overall process of simulation, 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.

5.1 Voting Model

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

1. Choose an edge e_{ij} uniformly at random from $E(G)$
2. Randomly choose one of the endpoints of e_{ij}, call it v_i (the other endpoint will be v_j)
3. Repeat steps one and two until $\zeta(v_i) = \zeta(v_j)$, i.e. the vertices' opinions do not match
4. With probability α, remove e_{ij} from the graph
   (a) Choose a new vertex v_k uniformly at random from $V(G)$ until $e_{ik} \notin E(G)$
   (b) Add e_{ik} to $E(G)$
5. Otherwise (with probability $1 - \alpha$), set $\zeta(v_i) = \zeta(v_j)$

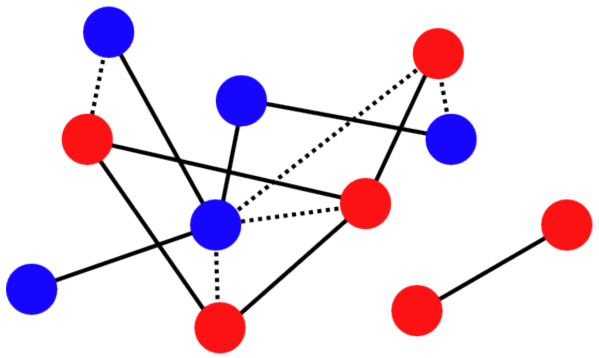


Figure 5: A possible configuration of a two-opinion model. The dotted lines represent conflicts. We may take the blue nodes to be Democrats, the red Republicans; then, the highly connected blue node may represent President Obama, the isolated red component Michele Bachmann and Sarah Palin.

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 what is known as the coupon collector's 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. The system is stochastic, so in order to obtain smooth evolutions of variables the average trajectory of an ensemble of voting models is used. In fact, both the minority fraction and conflict count were used as coarse variables (though conflict count appears slaved to minority fraction is likely unecessary). After projecting these variables forward in macroscopic space, the graph generation problem must be addressed. Thankfully, creating a network 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%.

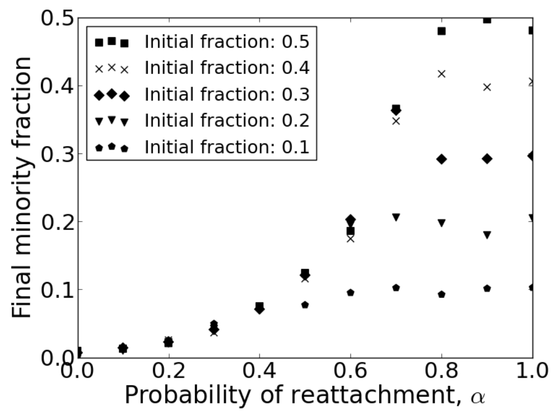
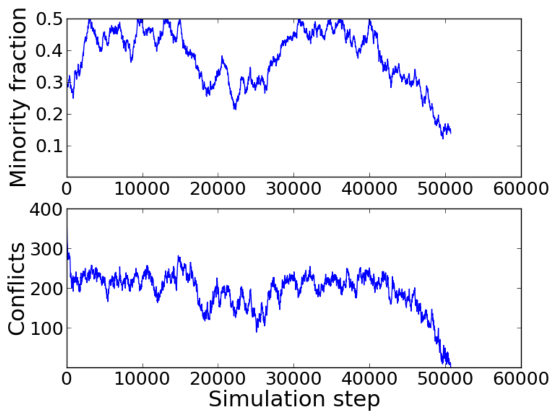
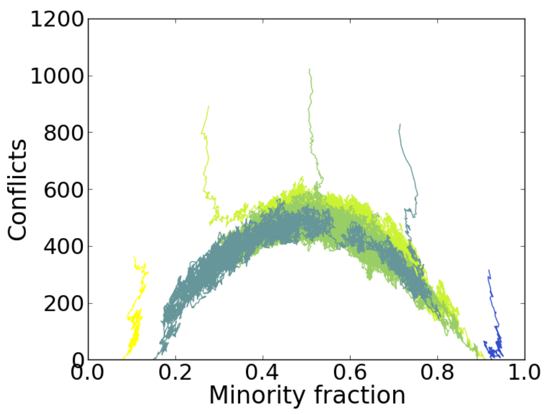


Figure 6: The effect of α on the final minority fraction. Above $\alpha \approx 0.72$, the final and initial fractions are approximately equal. Below this, a curve is mapped out. Any initial minority fraction above the curve will evolve towards its steady state value below.



(a)



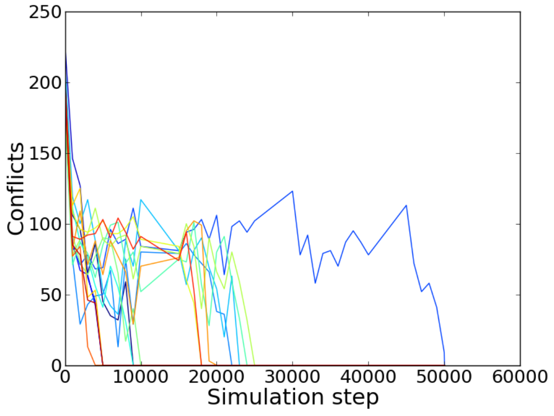
(b)

Figure 7: Evolution of minority fraction and number of conflicts, 7a, and phase plot, 7b. Note the strong correlation between the minority fraction and conflict evolution in 7a.

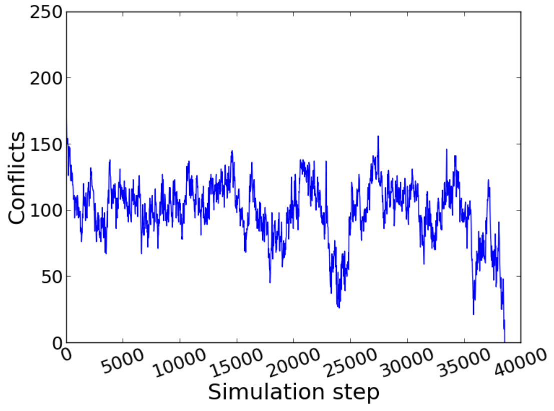
5.2 Edge Reconnecting Model

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 vertices (such a construction is termed a "multigraph"). Initially, $m \asymp n^2$ edges are distributed uniformly among the n vertices. Then 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



(a) CPI in an ensemble of 16 simulations



(b) Direct simulation

Figure 8: System evolution with and without CPI. Note that initially in 8a, with many trajectories averaged together, a large number of models quickly converge. However, the remaining runs become somewhat erratic as they lose the dampening effect that averaging with many simulations had provided. In both cases, $n = 200$.

2. Choose a vertex v_k using linear preferential attachment: $P(v_k = v_l) = \frac{d_l}{\sum\limits_{i=1}^{n} d_i}$

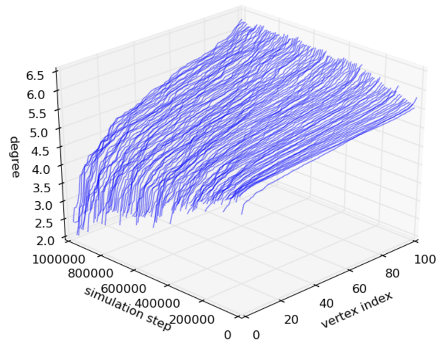
3. Replace e_{ij} with e_{ik}

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

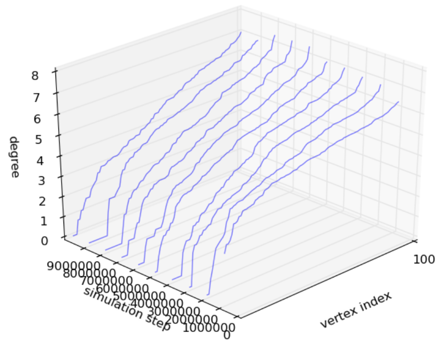
Interestingly, while the network is strictly unlabelled, the preferential attachment dynamics tend to stratify the vertices into a low degree majority and extremely high degree minority. This permits a sort of pseudo labelling of the vertices, as a vertex of high (low) degree at time t_1 is likely to retain its high (low) degree at t_2. Therefore, when sorted by degree, the evolution of the adjacency matrix appears smooth, as shown in 10b.

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. 9a and 9b illustrate attempts to visualize these separate scales of evolution. The degree distribution is plotted every n^2 steps in the 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



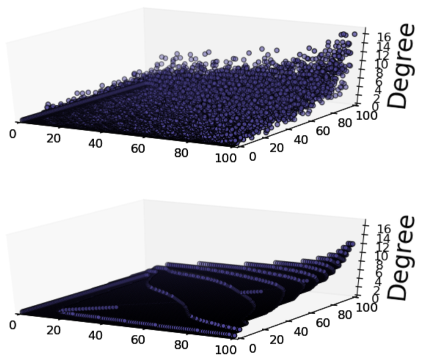
(a)



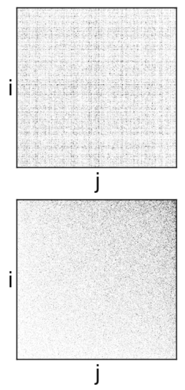
(b)

Figure 9: Degree evolution on n^2 and n^3 timescales, in 9a and 9b respectively.

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 $|\lambda_1| \gg |\lambda_i|, i = 2, 3, ..., n$.



(a) Reconstruction of original adjacency matrix (top) using first eigenvalue/eigenvector pair (bottom).



(b) The unsorted adjacency matrix appears random (top), but sorting by vertex degree reveals its structure (bottom).

Figure 10: Degree evolution on n^2 and n^3 timescales, in 9a and 9b respectively.

Fig. 10a illustrates the reconstruction of the adjacency matrix as $A_{ij} = \lambda_1 v_i^{(1)} v_j^{(1)\,\dagger}$ (after multiplication, each entry A_{ij} 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 the system. I norder to further reduce dimensionality, the eigenvector was fitted with a fifth-degree polynomial, as shown in Fig. **??** (a fifth-degree polynomial was used as additional terms didn't significantly increase fitting accuracy). 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 $v(i) = \sum_{k=0}^{k=6} i^k c_k$ (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 $A_{ij} = \lambda_1 v_i^{(1)} v_j^{(1)\,\dagger}$ and round as discussed previously
5. Repeat step one until system reaches steady state

Preliminary results of this method are shown in Fig. 11, 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 than the full simulation to converge to a fixed point.

6 Future Work

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

6.1 Graph Generation

The recent optimization based method of graph generation will certainly serve as a starting point for this avenue. The main limitation of the approach its scalability. Preprocessors have been shown to significantly reduce runtimes, but have been created to address specific properties. These operate by removing search paths unlikely to yield the specified graph before the optimization algorithm begins. This works particularly well when many distinct (non-isomorphic) graphs exist that satisfy the stipulations. In this case, the preprocessor does not need to be particularly careful in its deletions, as it is unlikely to remove all branches

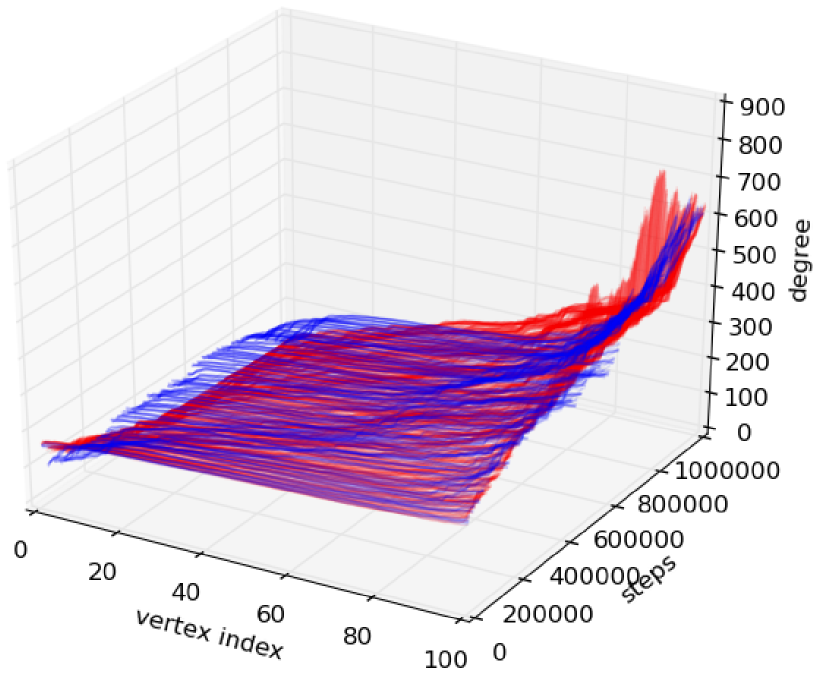


Figure 11: Overlay of direct simulation (red) and CPI (blue).

leading to feasible solutions. When more specificity is needed, it is possible to tune this step to be more cautious in its selections. Creating a generic preprocessor for arbitrary properties would retain the generality of the method while increasing speed.

6.2 Data Mining across Graphs

This nascent research topic hasn't received much attention yet. Certain sub-problems (graph isomorphism, vertex matching) have been thoroughly investigated, but the concept of applying dimensionality techniques when data points are, themselves, networks seems new. As mentioned, it may be possible to adapt an existing approximation algorithm that determines graph isomorphism to additionally yield information on the differences between two networks. A thorough understanding of the structure of these methods through an extensive literature review will be a necessary starting point.

The little related work that has been done involves defining kernels over graphs [**?**]. These all seem to be based on random walks over the network, and subsequent comparisons of the traversed paths. By defining a measure of similarity between two paths, the collection of walks over each graph can be compared to find a similarity of the graphs as a whole. It would be interesting to investigate the possible relationships between these methods and diffusion maps, which appear to have been formulated independently. Otherwise, the foundation of these graph kernels, a paper from 1999 [**?**], provides a general framework from which other kernels could be invented.

In general, work would focus on developing graph similarity measures in simple models, such as Erdős-Rényi, in which issues would be more readily diagnosed and guarantees of success or failure more easily proven. Additionally, simpler, heuristic measures of graph similarity have been shown to work in certain scenarios [**?**]. These could be used while more grounded methods are developed, or a combination of simple measures could be found that adequately addresses our needs.

6.3 Coarse System Dynamics

The immediate aim in this direction will be to apply diffusion maps to the voting model data in an attempt to recover the minority fraction as a good coarse variable. This is expected to work, as DMAPS has been able to discover appropriate coarse variables in other collections of graphs using heuristic similarity measures. It may be difficult to implement a coarse fixed point method on the system, as the final system state is artificial. Unlike a tradional steady state, a voting model simulation ends not when it ceases to change with time, but when all conflicts are resolved. If the system were left running, it would not necessarily remain in the same configuration. Outside of the voting and edge reconnecting models, the search will continue for systems with separations of timescales amenable to coarse analysis.

References

[1] A. Barabási and R. Albert. Emergence of scaling in random networks. *science*, 286(October):509–512, 1999.

[2] B. Barzel and A.-L. Barabási. Universality in network dynamics. *Nature Physics*, 9(9):1–9, Sept. 2013.

[3] V. Batagelj and U. Brandes. Efficient generation of large random networks. *Physical Review E*, 71(3):036113, Mar. 2005.

[4] M. Deijfen and W. Kets. Random intersection graphs with tunable degree distribution and clustering. *Probability in the Engineering and the Informational Sciences*, 2009.

[5] M. Dreze, B. Charloteaux, S. Milstein, P.-O. Vidalain, M. a. Yildirim, Q. Zhong, N. Svrzikapa, V. Romero, G. Laloux, R. Brasseur, J. Vandenhaute, M. Boxem, M. E. Cusick, D. E. Hill, and M. Vidal. 'Edgetic' perturbation of a C. elegans BCL2 ortholog. *Nature methods*, 6(11):843–9, Nov. 2009.

[6] P. Erds and A. Rényi. On random graphs I. *Publicationes Mathematicae*, 1959.

[7] C. E. Gounaris, K. Rajendran, I. G. Kevrekidis, and C. A. Floudas. Designing Networks : A Mixed-Integer Linear Optimization Approach. 2013.

[8] J. Kleinberg. Authoritative sources in a hyperlinked environment. *Journal of the ACM*, 1999.

[9] M. Newman and M. Girvan. Finding and evaluating community structure in networks. *Physical Review E*, 69(2):026113, Feb. 2004.

[10] D. J. Watts and S. H. Strogatz. Collective dynamics of 'small-world' networks. *Nature*, 393(6684):440–2, June 1998.

[11] L. a. Zager and G. C. Verghese. Graph similarity scoring and matching. *Applied Mathematics Letters*, 21(1):86–94, Jan. 2008.