Solving Inverse Problems on Evolving Graphs

Andrew Fraser
University of Utah School of Computing, United States

Abstract

I investigate the inverse problem posed by [1]. Given a number of states of the graph G, I wish to create a model based upon birth and death rates on which to predict future states of G. This model uses a range-dependent graph, where each vertex is assigned an integer that determines its distance from other vertices. This problem is quadratic in nature and has already been solved using specific eigenanalysis in [1]. However, this solution takes time on the order of $O(n^3)$. In my investigation, I use techniques that we learned in CS 6958 to solve the inverse problem with greater speed and precision.

2012 ACM Subject Classification TBD

Keywords and phrases Inverse Problems, Evolving Graphs

Digital Object Identifier 10.4230/LIPIcs...

1 Introduction

Graphs of vertices and edges are a mathematical abstraction that has been used commonly to refer to people or objects and their connections. Most recently, graphs have expanded to depicting structures that we see in technology. Especially worth noting lately are graphs being used to depict social media networks, the internet, and other complex social structures. In particular, it is unique that these new graphs can evolve over time. Due to their ability to hang in data over the internet, these graphs can evolve to many different states. We can model these changes through an evolving graph, which is a collection of states of the graph at different points in time.

This new type of graph leads to the question: can we predict what the graph will look like in the future? This can be used to help predict and recommend choices of friends on social networks or predict connections and popular sites that may arise on the internet. One method of predicting the behaviour of evolving graphs has been proposed by [1], where they propose a model for evolving graphs that can be solved as an inverse problem. Using the techniques I've learned in CS 6958, I explore solving this inverse problem. In particular, I have the following goals:

- To solve the inverse problem faster using methods from class
- To solve the inverse problem more accurately
- To test these solutions on additional real data (which is made possible by speeding up these solutions)

As the current solution takes $O(n^3)$, solving the inverse problem on large datasets is not plausible. By speeding up the solution, I hope to apply this inverse problems to datasets that are larger than the 100 vertex graphs tested in [1].

2 Preliminaries

I begin by providing definitions needed to understand the inverse problem presented by Grindrod et al. in [1]. In particular, they wish to fit a series of states of the graph G by fitting them to a model. One model for this is a model of birth rates and death rates. For each pair of vertices $u, v \in G$, I want to assign the following:


Leibniz International Proceedings in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

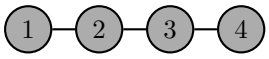

Figure 1 A path showcasing an assignment of ranges on the vertices. Vertex 1 and vertex 4 have a range of $4 - 1 = 3$, whereas vertex 1 and vertex 2 have a range of $2 - 1 = 1$.

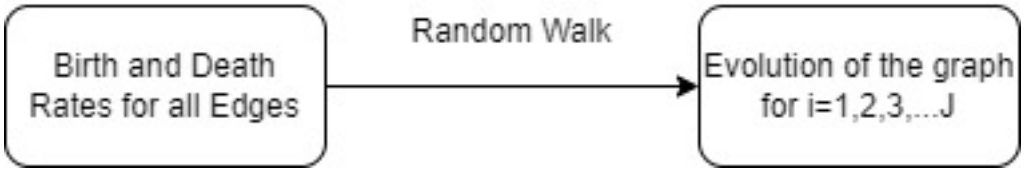

Figure 2 This figure defines the inverse problem I am aiming to solve.

- Birth Rate $\alpha(u, v)$: The probability of the edge $u - v$ appearing if it does not exist in the current state.
- Death Rate $\omega(u, v)$: The probability of the edge $u - v$ disappearing if it exists in the current state.

To find a reasonable assignment for these birth and death rates, I use the idea of range-dependent graphs.

▶ **Definition 1.** *Range-dependent graphs are an assignment of the integers $q = \{1, 2, \ldots, n\}$ on n vertices such that, for each vertex, the range between two vertices is defined as $|q(u) - q(v)|$.*

I can use this definition of range as a way to determine the probability of two vertices having an edge. The smaller the range between two vertices, the more likely that they should have an edge.

This brings us to the inverse problem I wish to solve. Given J states of the graph G, we wish to learn a range assignment q on the vertices such that the error is minimized. Refer to Figure 2 for a depiction of the inverse problem.

Formally, the problem is defined as follows:

Evolving Graph Model

Input:	J states of the graph G and a matrix R containing the frequencies of each edge in G
Problem:	What mapping on the integers q minimizes $\sum_{v_1,v_2 \in G} R[v_1][v_2](q(v_1) - q(v_2))^2$?

By solving this problem, I can use the range mapping q to define birth and death rates that make reasonable predictions for future evolutions of G. As q is a non-differentiable function as a mapping, a solution here is not well defined. The problem can be relaxed by considering q as a vector. This creates the quadratic inverse problem of $q^T \Delta_R q$. Grindrod et al. [1] solve this problem via complex eigenanalysis. The rest of this report concerns my results in applying methods from the course on this inverse problem.

3 Initial Results: Gradient Descent

In trying to tackle this inverse problem as a quadratic, I've considered that I need to approach the problem from a different angle. Many of the methods we've discussed in class won't

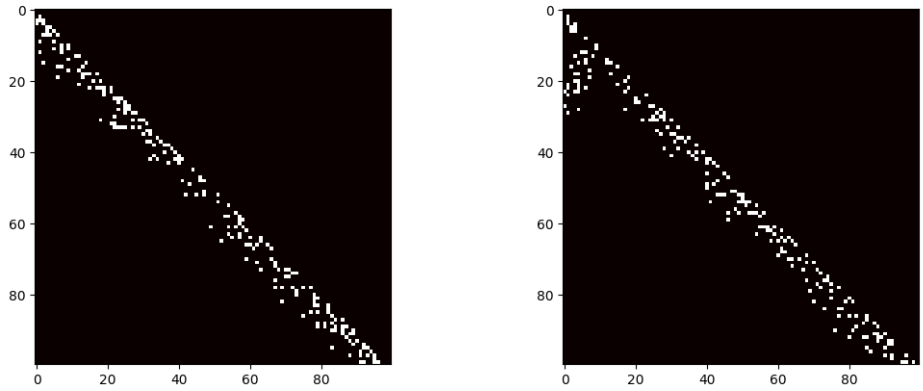

Figure 3 Left: State 200 of the original graph. This range-dependent mapping has error = 1.2M. Right: State 200 of the graph with a range mapping learned by gradient descent. Note that the mapping is mostly accurate, but gets caught in a local minimum where the vertices in the top left have been mapped inoptimally. This range-dependent mapping has error = 1.6M

necessarily apply directly to the problem. Particularly, three different methods seem to work best in combatting quadratic inverse problems:

- Linearize the problem
- Use optimization techniques
- Use specialized algorithms

The methods used in [1] seem to fall under a specialized algorithm, as they perform case analysis on the eigenanalysis of δ_R. My immediate reaction was to attack the problem as an optimization problem, as I have experience in this both in this course and in solving other problems.

I began by implementing a gradient descent algorithm to solve the problem. To test this solution, I directly created 200 states of a graph using a random range-dependency q. For two vertices with range k, they have a birth rate $\alpha(k) = 0.1(0.98^{k^2})$ and death rate $\omega(k) = 0.2$. Then, I ran my algorithm on these states to learn a mapping q', then ran another walk q' to generate new states of the graph. Assuming that the birth and death rates of the graph are similar, these simulations should result in very similar graphs after 200 iterations. See Figure 3 for the results of this experiment. Overall, this is a very solid fit. You can see that my algorithm ends up with nearly the same graph.

4 Runtime Issues and New Methods

My initial goal was to run my algorithm on real graph datasets. However, real world datasets of evolving graphs tend to have at least 100,000 vertices. With my gradient descent algorithm taking $O(n^2 * k)$, where there are k iterations, I was worried about runtime being an issue.

Following this notion, I decided to perform a test of runtime in Figure 4. It's clear that this runtime is not likely to be reasonable on anything larger than 1000 vertices. Therefore, I decided to try other methods to solving the problem. Continuing in the optimization front, I tried using various methods from scipy's optimization library. However, this did lead to some issues caused by the ill-posed nature of the problem:

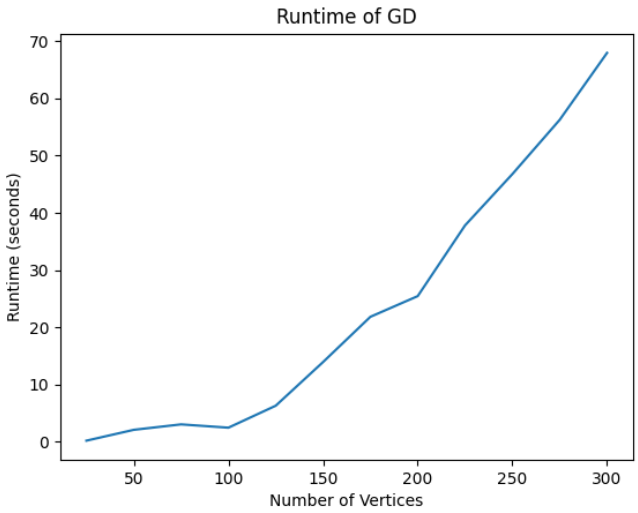

Figure 4 Runtime analysis of Gradient Descent on different sizes of graphs.

- Using automatic differentation would lead to an optimal value of 0 for every spot in q. This is caused by relaxing q to be a vector instead of a mapping. To avoid this, a gradient must be directly provided (leading me to focus on gradient methods like CG and Newton-CG)
- To get a result that is very accurate, a gradient algorithm must walk very far down the gradient. In particular, my gradient descent walks down the gradient until the magnitude of the gradient is smaller than $10^(- 160)$. This leads to many iteration steps (even over 10,000), but this seems necessary to get an extremely accurate model.

After testing various optimization methods in the scipy library, I found the Newton Conjugate Gradient method to be the best-performing of them all. The results of Newton-CG are not nearly as good as my gradient descent results, but I think this is because the problem requires walking down such a small gradient for good results. See Figure 5 for the accuracy of Newton-CG. The program does converge after less than 100 iterations, and therefore runs far faster than my gradient descent. So this seems to be a good way to approximate this problem in general for larger datasets.

However, while trying to run larger trials, I actually found that the runtime of generating new states of the graph and reading in the states of the input took longer than the actual Newton CG method. This has lead me to the following observations:

- A model that is $O(n^2 * s)$, where s is the number of states, is far too slow for any social network or internet data. This model of birth and death rates per edge cannot hold for such graph sizes.
- Most models and algorithms that I know for dealing with normal large graphs deal directly with the sparsity of the graph and abuse common structures of the graph (for example, cliques or triangles)

Following these observations, I could not find a dataset that seemed to be a solid fit for this. The results in [1] apply their method to electroencephalography and they do mention other very scientific applications, such as protein-protein networks. However, I am interested

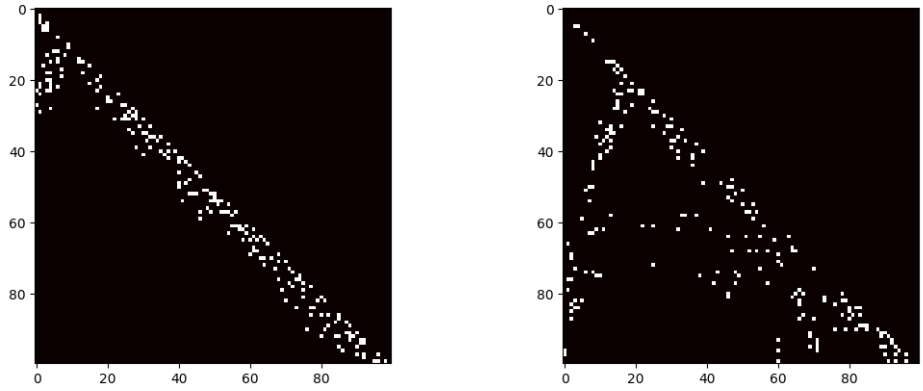

Figure 5 Left: State 200 of the graph with a range mapping learned by gradient descent. Right: State 200 of the graph with a range mapping learned by Newton Conjugate Gradient (Newton-CG).

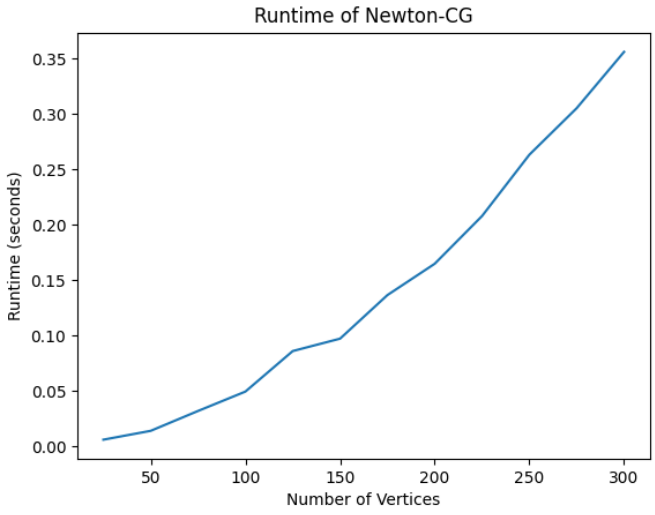

Figure 6 A runtime analysis of the Newton Conjugate Gradient method.

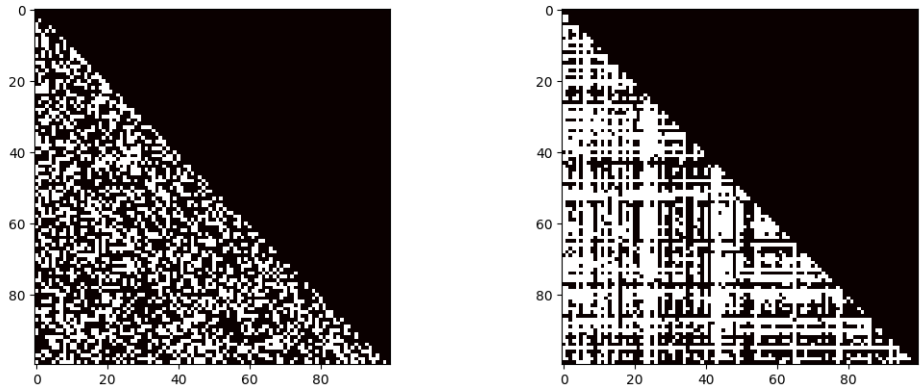

Figure 7 Left: State 100 of the original social graph. Right: State 100 of the graph predicted using gradient descent.

in how range-dependent graphs apply to social networks or website data in particular, as they don't seem like extremely natural applications. I am curious to see if these techniques have any chance of applying to such scenarios, even notwithstanding the issue of problem size.

5 Testing on Synthetic Social Networks

In this section, I test my gradient descent and Newton-CG solutions on synthetic graphs that are not produced via a range-dependent system. In particular, I generate these graphs using a trend commonly noticed in social media networks. If two people in a network are friends, then their friends are also likely to become friends. This idea is known to create 3-cliques or triangles throughout the graph. More generally, this tends to create even larger cliques throughout the graph, as mostly-adjacent 3-cliques will combine to create larger cliques over time.

Applying this idea, I synthetically created an initial graph with 5% of all edges present. Then, I apply the following for 100 evolutions:

- $\alpha(u, v) = 0.02 + k * 0.06$. k is the number of common neighbors between u and v
- $\omega(u, v) = 0.03$

Using these rates, random friendships are more likely to disappear than reappear. However, friendships that are created via a common friend are far more likely than anything. This leads to clique-esque structures in the graph.

Following this generation of evolutions, I run the first 50 evolutions through my gradient descent and Newton-CG algorithms. Then, I produce a prediction of the next 50 evolutions using the models created by the algorithms. Figure 7 and Figure 8 show the results for these experiments using a birth function of $0.2 * 0.8^k$, where k is the range between two vertices. These results generated a graph which was far too connected, which I was quite surprised about.

Figure 9 and Figure 10 shows the results for these experiments using a birth function of $0.2 * 0.97^{(k^2)}$. These results seem to match far better to the original. It makes sense that using a sense of range to indicate likelier friendships works in this scenario to an extent, but

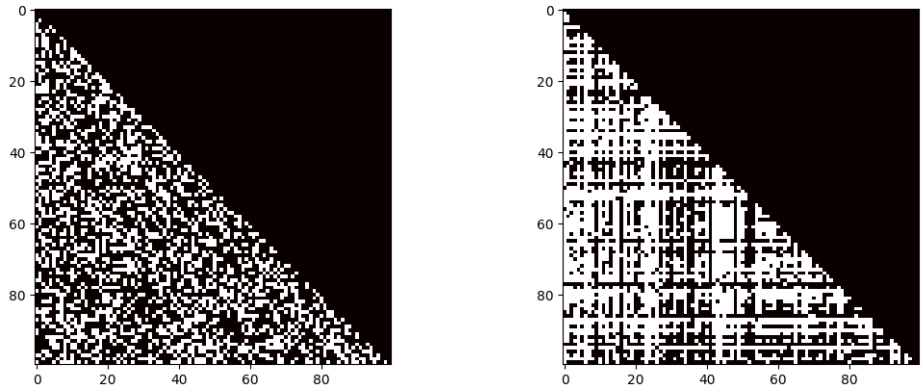

Figure 8 Left: State 100 of the original social graph. Right: State 100 of the graph predicted using the Newton Conjugate Gradient algorithm.

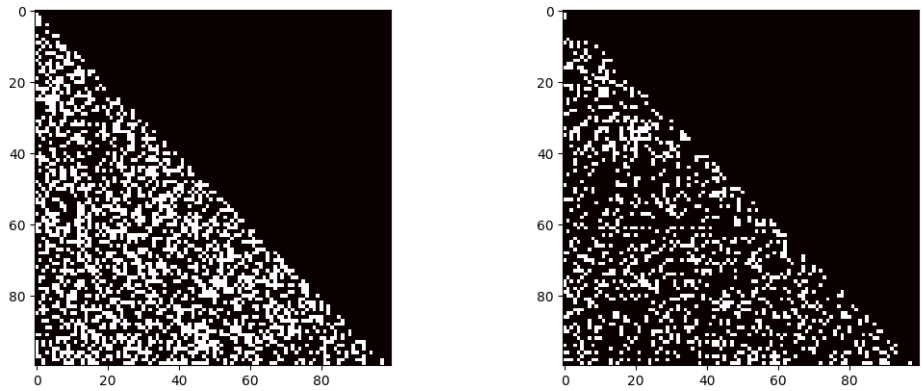

Figure 9 Left: State 100 of the original social graph. Right: State 100 of the graph predicted using gradient descent and a quadratic birth rate.

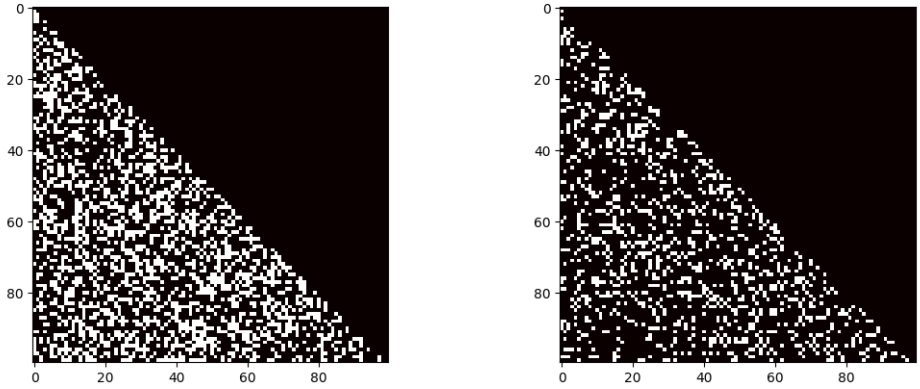

Figure 10 Left: State 100 of the original social graph. Right: State 100 of the graph predicted using the Newton Conjugate Gradient algorithm and a quadratic birth rate.

I am surprised that it worked so well. The results are still a bit sparser than the original graph, but overall these seem surprisingly accurate. I would be interested to see if this holds on larger datasets if the algorithms would work on those instances.

6 Conclusion

Overall, I am reasonably pleased with these results. I feel that the algorithms I presented to solve the inverse problem at hand are reasonable. My gradient descent algorithm is very accurate but takes a long time to run. My Newton-CG algorithm is not as accurate but runs surprisingly fast, such that even the model itself is slower to compute. Following these restrictions, I think that these approaches seem to solve the inverse problem I chose to tackle quite well.

Given more time, there are a few more points that I would like to explore:

- Applying the DBAR method to this problem. I think it's possible that this algorithm could run as fast as Newton-CG and get more accurate results, but this is untested. A similar inverse problem solved in a case study from class uses this algorithm to great effect.
- Applying my algorithms to more scientific data (for example, a protein-protein network). Finding data to do this on was difficult given the problem size restriction, but if I could find something in the area I would be very interested to test on it
- I'd like to do more research on approaches to evolving graphs. I'm unsure that these other approaches involve inverse problems, so I did not find them appropriate to try in this project, but I would find them interesting to investigate nonetheless.

References

1 P. Grindrod and D. J. Higham. Evolving graphs: dynamical models, inverse problems and propagation. *Proceedings of the Royal Society A: Mathematical, Physical and Engineering Sciences*, 466(2115):753–770, 2010.