The Influence of Collaboration Networks on Programming Language Acquisition

by

Sanjay Guruprasad

Bachelor of Technology (Engineering Physics), Indian Institute of Technology Madras (2012)

Submitted to the Department of Media Arts and Sciences
in partial fulfillment of the requirements for the degree of

Master of Science
at the
MASSACHUSETTS INSTITUTE OF TECHNOLOGY
June 2018

Signature redacted

Signature of Author ....

Program in Media Arts and Sciences
May 4, 2018

Signature redacted

Certified by ....

César Hidalgo
Associate Professor, Program in Media Arts and Sciences
Thesis Supervisor

Signature redacted

Accepted by ....

Tod Machover
Academic Head, Program in Media Arts and Sciences

The Influence of Collaboration Networks on Programming Language Acquisition

by

Sanjay Guruprasad

Submitted to the Department of Media Arts and Sciences
on May 4th 2018, in partial fulfillment of the
requirements for the degree of

Master of Science

Abstract

Many behaviors spread through social contact. However, different behaviors seem to require different degrees of social reinforcement to spread within a network. Some behaviors spread via simple contagion, where a single contact with an "activated node" is sufficient for transmission, while others require complex contagion, with reinforcement from multiple nodes to adopt the behavior. But why do some behaviors require more social reinforcement to spread than others? Here we hypothesize that learning more difficult behaviors requires more social reinforcement. We test this hypothesis by analyzing the programming language adoption of hundreds of thousands of programmers on the social coding platform Github. We show that adopting more difficult programming languages requires more reinforcement from the collaboration network. This research sheds light on the role of collaboration networks in programming language acquisition.

Thesis Supervisor: César Hidalgo

Title: Associate Professor, Program in Media Arts and Sciences

The Influence of Collaboration Networks on Programming Language Acquisition

by

Sanjay Guruprasad

The following people served as readers for this thesis:

Thesis Reader .... Signature redacted

Mitchel Resnick

LEGO Papert Professor of Learning Research

MIT Media Lab

Thesis Reader .... Signature redacted ....

Pierre-Alexandre Balland

Assistant Professor of Economic Geography

Utrecht University

Contents

2 Introduction

The spread of behaviors through networks is an important and widely studied phenomenon. The study of contagion in networks began with epidemic models for infectious diseases. As network science began to blossom as a field, the study of contagion grew to a diverse array of behaviors. Many ubiquitous phenomena are now known to spread through networks in specific ways — infectious diseases [1], memes [2], smoking habits [3], obesity [4], job information [5], innovations [6] and even the evolution of economic activities [7].

Studies of diffusion dynamics have explored the role of network structure and influential nodes [7][8][9][10] in these contagion processes. Initial research extended traditional epidemic models to understand the spread of phenomena like rumors [11], and more recently, computer viruses [8]. When these phenomena were studied further, it became apparent that two distinct types of contagion exist. Diseases [1] and information about jobs [4] spread via simple contagion, where transmission occurs through contact with a single individual. In a complex contagion, concepts require social reinforcement from multiple nodes to spread [12]. Social behaviors often spread as a complex contagion [13].

When Centola proposed the idea of complex contagion, he conjectured that the adoption of expensive, controversial or risky behaviors would be more likely to require social reinforcement from multiple actors [12]. Aral and Nicolaides decided to investigate if exercise habits spread as a complex contagion. They studied a social network of one million runners, and showed that exercise habits spread as a simple contagion [14]. This was surprising as they considered exercising to be a relatively expensive task when compared to other studied examples like the sharing of internet memes [2] or the adoption of a social media app [26].

Christakis and Fowler studied the influence of social connections on smoking, and concluded that smoking cessation was strongly infectious in people with strong ties (spouses, siblings and coworkers in small firms) [3]. On the other hand, in networks at the scale of nations and regions, Hidalgo et al. find that the knowledge of making new products spreads via complex contagion [15][6]. The same is found to be true in the spread of research areas [16] as well.

This leads us to a key question. Why do some behaviors require more social reinforcement to spread than others?

Intuitively, Centola's conjecture makes sense. Difficult or controversial behaviors are much more likely to require complex contagions. For a country to develop a new industry, a variety of underlying capabilities need to be ready. The presence of workers with the required knowledge, the availability of the required infrastructure and technology and the presence of institutions and laws that enable such an industry are all important factors [6]. Thus, the presence of related industries plays an outsized role in determining the probability that the new industry will evolve. And the presence of multiple related industries substantially increases the probability of success. In contrast, a single recommendation can lead to someone being hired for a job. With extensive standardization for many job roles, hiring reduces to the spread of information and trust. Thus, job information spreads through a network as a simple contagion [4].

The examples highlighted above are vastly different, making it hard to make the claim that the difference in difficulty between the tasks is the only reason one spreads as a complex contagion, while the other as a simple contagion. Since we would like to study how patterns of contagion change with difficulty, we choose to study a phenomenon where we can find a single concept with multiple levels of difficulty — programming languages.

Different programming languages have different levels of difficulty, and programmers frequently learn and forget languages [17]. Some programming languages are designed for a very specific task and can be learnt in a matter of hours. Other languages require one to invest in multiple new tools and frameworks, and take weeks to set up and use successfully.

We hypothesize that the more difficult a programming language, the more social reinforcement a programmer needs to learn it.

Millions of programmers use the social coding platform Github to create and collaborate on software projects [18][19]. Thus, studying programming language adoption on Github gives us a method to quantify the relationship between the difficulty of languages and the need for social reinforcement.

The topic of technology diffusion has been studied for close to fifty years, with a variety of theoretical models that have evolved over time to incorporate different scales and phenomena. In his famous book, The Diffusion of Innovations, Everett Rogers defines diffusion as the "process by which innovation is communicated through certain channels over time among the members of a social system". In this definition, he emphasizes what he believes to be the five most important factors influencing technology diffusion [5]. All five factors are at play in the adoption of programming languages on Github.

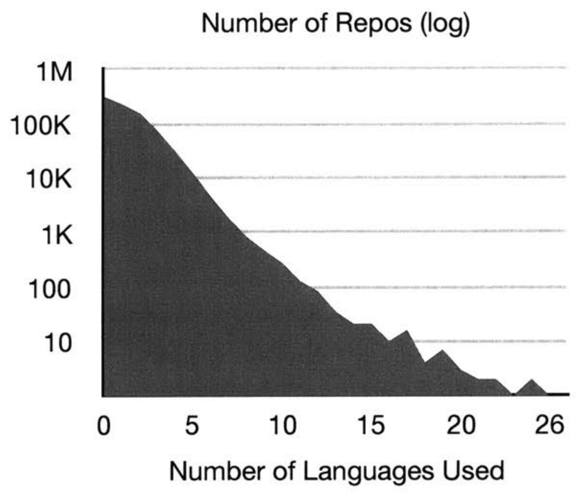
Understanding network effects on various aspects of the development lifecycle has been a topic of interest. Casalnuovo et al have shown that 90% of the "prolific" programmers on Github preferentially join projects with past social connections. They also show that programmers prefer to join projects in programming languages they have used before [20]. Meyerovich and Rabkin analyze programming languages through a combination of data from the platform SourceForge and surveys of programmers. They conclude that programming language adoption follows a power law, but the market supports many languages with niche user bases [17], hinting at network effects in programming language communities. However, little research has been done in understanding the spread of programming languages through collaboration networks. In this work we show that adopting more difficult programming languages requires more reinforcement from one's collaboration network.

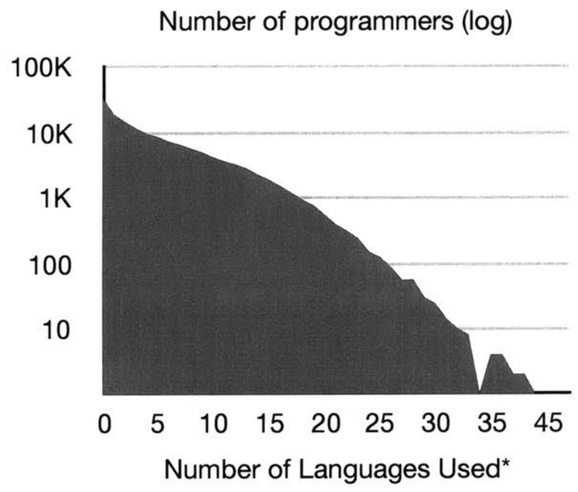
We study this hypothesis by analyzing the programming language adoption of hundreds of thousands of programmers on the social coding platform Github. We show that adopting more difficult programming languages requires more reinforcement from the collaboration network. Our research sheds light on the role of collaboration networks in programming language acquisition.

3 Data

The social coding platform Github has risen to prominence as the most popular platform for collaborative coding among developers. As of 2017, the platform has over 25 million active public repositories (projects) and continues to grow. Github is built around the version control software git. Git is the dominant choice for version control for programmers, with almost 90% of programmers using git according to the StackOverflow 2018 survey of 74,298 developers [21]. Of these developers, 60% check-in code everyday. Programmers collaborate on repositories by pushing "commits" (units of code containing changes to multiple files) to the repository. Each commit is authored by a single programmer, but can contain files written in multiple different programming languages.

The dataset was constructed by snowball sampling from a list of 256 highly active github users through the Github APIs to get data on over 600,000 developers. For the rest of this study, we use a subset of this dataset, constructed from a random sample of 1,000 programmers and all their collaborators (105,000+ programmers). Two programmers are considered collaborators if they have both pushed code to the same repository. In our dataset, the average developer has committed code in at least 5 different languages. The average repository has commits in at least 2 languages.





*a language is counted if the author/repo has at least 10 commits in the language

Github only assigns programming languages at a repository level. However, to correctly detect that a programmer has adopted a language, we require programming languages at the individual "commit" level. This is especially important since many projects have a division of labor, for example, in a web development project, some contributors write documentation in Markdown, some write front-end code code in JavaScript, and others write back-end code in Go.

To ensure we have accurate information on each programmer's contributions, we retrieved the list of files changed for every commit by all 105,000+ programmers in our dataset. This amounted to fetching the list of files changed for about 46.5 million commits from Github. The filenames and file extensions were used to determine the programming languages present in a commit (details of dataset construction in appendix).

3.2 Limitations

The dataset is restricted to open source personal projects on Github. This is a limitation as it excludes two other types of projects — personal projects in private repositories, as well as projects in organizational repositories. These are limitations on the completeness of the dataset. However, the 1000 randomly sampled programmers meet a minimum activity criterion of 500 open source commits, which means they all have significant open source output.

4 Results

Programmers become collaborators by contributing code commits to the same repository. Since the average repository uses 1-2 languages, collaborations normally begin in a specific language. However, the average programmer has made commits in at least six different languages and there are over 343 actively developed programming languages on Github. Thus, programmers often collaborate with someone who knows a language that they have never used. Given that a programmer has multiple collaborators, how does the collaboration network influence their adoption of programming languages? Figure 1, highlights the spread of the programming language "Go" through a collaboration network. The network consists of programmers who first collaborated in a language other than Go. We see that Go proliferates through this network over a six year period. However, if we look at the dataset as a whole, less than 13% of the 105,000+ programmers use Go.

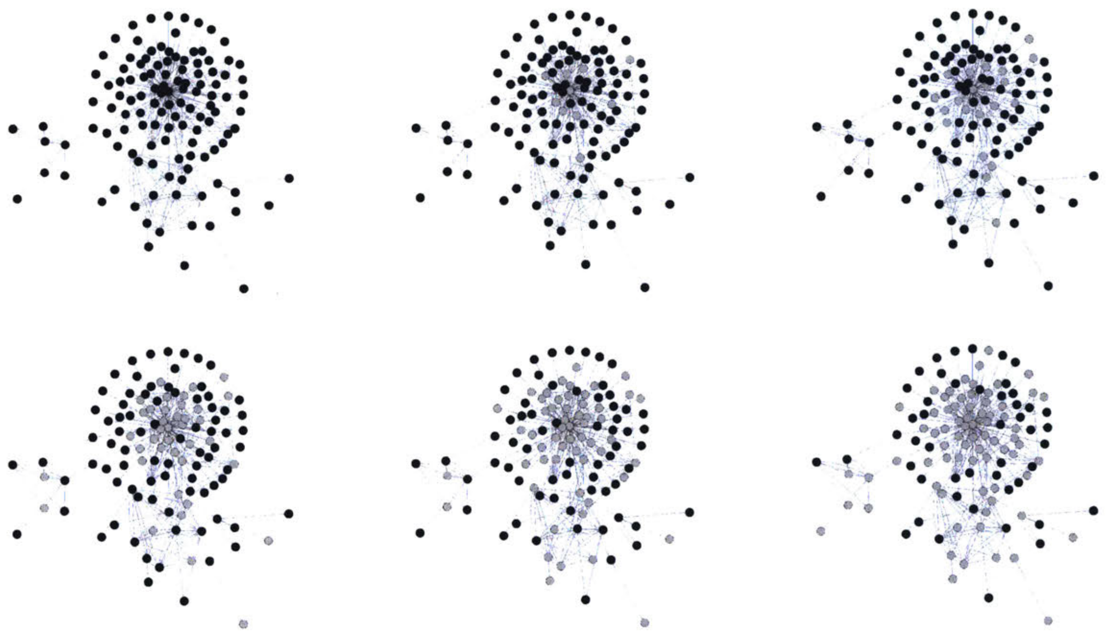


Figure 1. Diffusion of the programming language "Go" through a collaboration network on Github from 2009 (top left) to 2014 (bottom right). Nodes represent programmers. Nodes turn yellow after the first commit in Go.

Is the spread of Go within this collaboration network just an exception, or is this the norm?

To understand the contagion of languages through collaboration networks, we study the adoption patterns of ten popular programming languages (see Table 1). We also include Plain Text and Markdown for comparison as these formats can serve as a control. They are widely used for documentation and licenses on Github and require minimal effort for adoption. HTML is a "markup" language and is also quite easy to use. At a feature level, Plain Text, Markdown and HTML do not require programming concepts like "variables" and "functions". In contrast, the remaining languages are full-featured and each have their own libraries, frameworks and ecosystems. Python and JavaScript are popular languages for beginners today.

Specifically, we are interested in understanding how the probability that a programmer uses a language changes with the fraction of collaborators who use the same language. We plot these "language adoption curves" for all twelve languages in figure 2. If there is no contagion, we expect no significant correlation between the probability of use and the fraction of activated collaborators. In the case of a simple contagion, we expect a sub-linear increase in probability of use, with additional activated collaborators contributing only marginally to increasing the probability that the language is adopted. However, in the case of a complex contagion, we expect to see linear or exponential growth in the probability of adoption as the number of activated collaborators increases [24].

We model these curves using a two parameter logistic function.

$$y = \frac{1}{1 + Ae^{-\alpha x}} \quad \text{(eq. 1)}$$

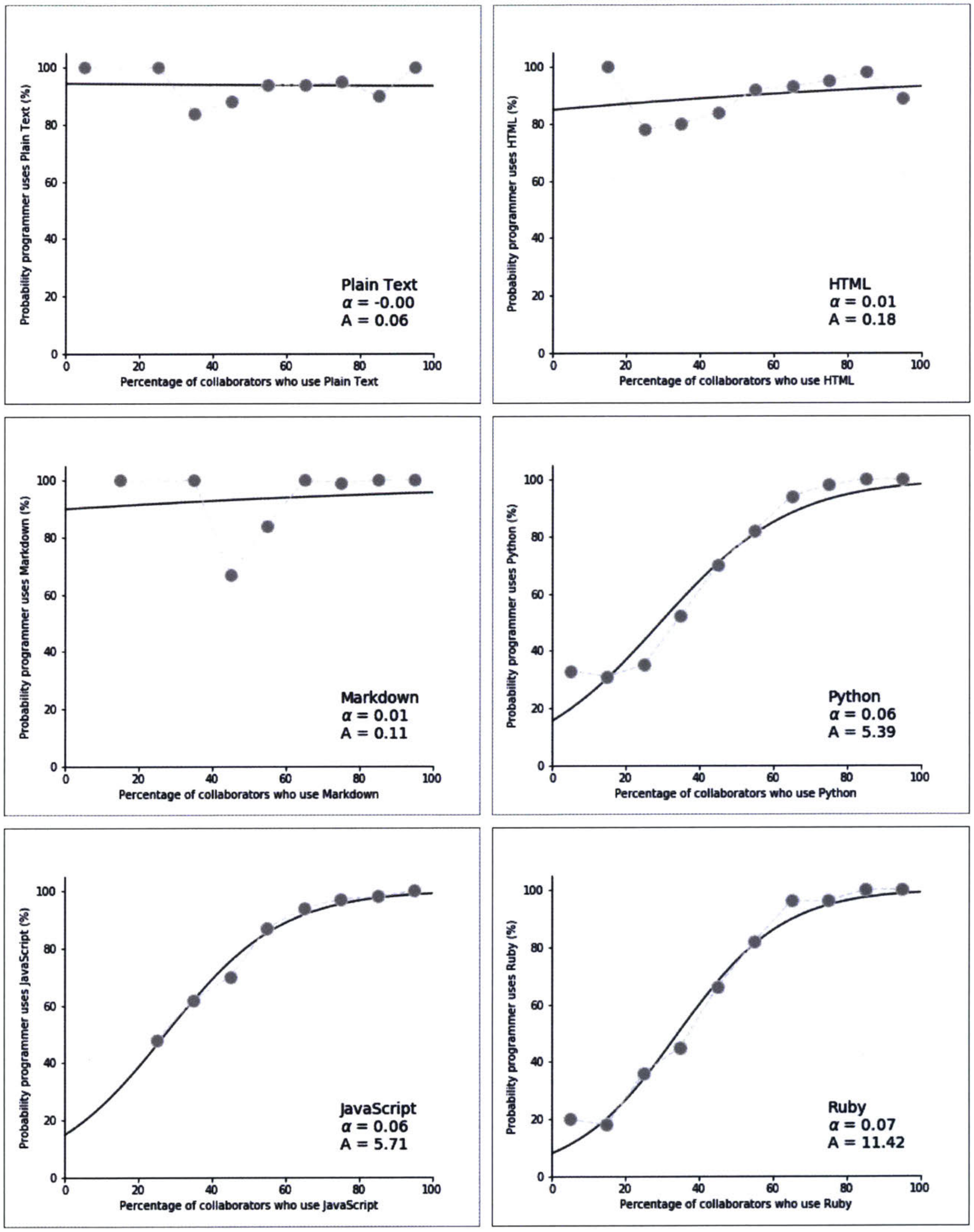


Figure 2a. Language adoption curves for languages 1-6 (easier languages). α increases as we go from Plain Text to markup languages like HTML and Markdown, and then increases further for Python / Javascript.

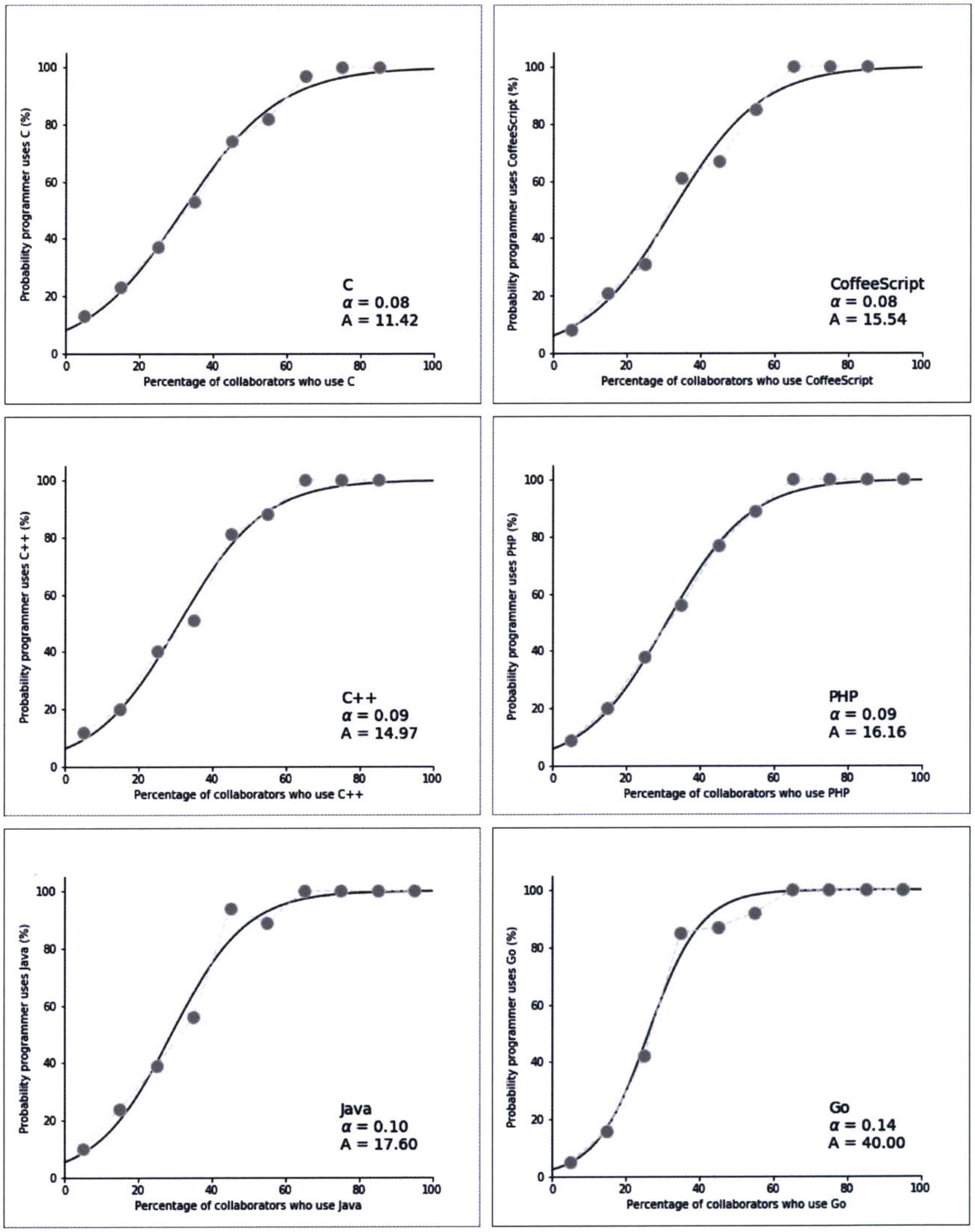


Figure 2b. Language adoption curves for languages 7-12 (more difficult languages). α continues to increase and is highest for the programming language Go.

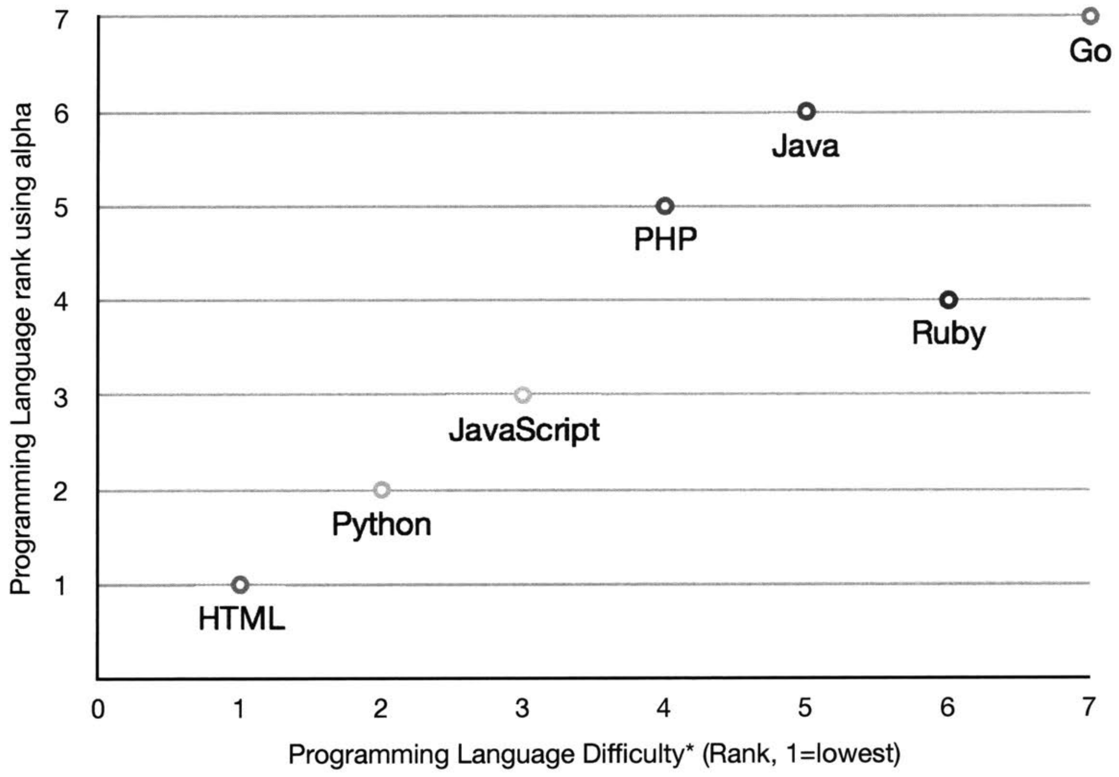
First, we observe two distinct behaviors. Plain Text, HTML and Markdown have flat adoption curves, indicating no correlation between the probability of adoption and the fraction of collaborators who use these languages. For all other languages, the curve increases exponentially before saturating. This means that the probability that an individual adopts one of these nine languages increases exponentially with the fraction of activated collaborators. We quantify this, using the parameter "α" in the language adoption curves.

Language	Users (%)	α
Plain Text	93.2	0
Markdown	99.0	0.01
HTML	92.4	0.01
Python	69.3	0.06
JavaScript	90.3	0.06
Ruby	61.4	0.07
C	47.1	0.08
CoffeeScript	36.0	0.08
C++	37.5	0.09
PHP	36.3	0.09
Java	42.4	0.10
Go	24.8	0.14

Table 1. Programming languages with number of programmers ranked by "α" values

The parameter α captures the "growth" characteristic of the logistic function we use to model the language adoption curve. Higher values of α imply steeper adoption curves. In table 1, we plot the value of alpha for each of the programming languages. The fact that Plain Text, HTML and Markdown have $\alpha = 0$ can be attributed to the fact that they are not full-featured programming languages and are therefore easy to adopt. The remaining languages have values of α ranging from 0.06 to 0.14. But what explains the difference between Python ($\alpha = 0.06$) and Go ($\alpha = 0.14$)?

We use a survey of 909 programmers [22] who were asked to rank languages on how easy they are to learn. Specifically, we look at the programming language rankings for the question "What do programmers think is the easiest language to learn?". In the survey, the languages are ranked based on the number of people who say a specific language is the easiest to learn. Seven of the ten languages in their rankings overlap with our dataset. In figure 3, we compare the rankings from the survey with the languages in table 1.



*Source: WP Engine Survey

We see that ranking programming languages using "alpha" correlates well with the difficulty rankings provided by developers in the survey. The more difficult a programming language, the steeper its adoption curve. This confirms our hypothesis — the more difficult a programming language, the more social reinforcement a programmer needs to learn it.

5 Conclusion

We study the influence of collaborators on programming language adoption and show that programming language adoption spreads via complex contagion for full-featured languages. We also show that more difficult languages possess steeper language adoption curves, indicating that adopting a difficult programming language requires more social reinforcement.

Programming languages constitute an important example of a problem of great interest — technology diffusion. In Everett Roger's seminal work on technology diffusion, he studied a variety of technologies and practices. For example, he showed the influence of an early adopter in the spread of a new weed spray among a group of 13 farmers [6]. Technology diffusion has been important at every scale, from small teams to entire countries. With the introduction of the internet and platforms like Github, networks of influence now span continents, with many people collaborating without ever meeting each other. Millions of developers are constantly learning and adopting new software technologies today. We are only beginning to understand the power and influence of these networks on the diffusion of knowledge.

Organizations that facilitate and encourage technology diffusion within their workforce will possess a strong advantage in this world of rapidly evolving technologies. Most software organizations today use collaborative platforms like Github internally, making granular technology adoption data available to all members of the organization. Understanding the mechanics of technology diffusion through networks of programmers will lead us to a paradigm where we can consciously exploit these networks to enhance technology adoption within organizations and teams.

6 Discussion and Future Work

There has been a long history of studying contagion processes in social behaviors. In this study, we have attempted to capture the relationship between the difficulty of a concept and the need for social reinforcement in its spread. In choosing programming language adoption, we study a phenomenon with varying levels of "difficulty" and have shown that the difficulty of a language is strongly correlated with the need for social reinforcement in adopting it.

In cases of contagion, it is often difficult to disentangle homophily and influence [27]. While we have demonstrated that social reinforcement plays a role in programming language adoption, it is unclear whether this is direct influence, or whether homophily is more common in communities who know difficult languages. Extending this work to measure the role of direct influence is an important next step. Since we have constructed a granular and timestamped dataset, we can attempt to disentangle homophily and contagion using matched sample estimation techniques [27]. However, these matching techniques will require us to expand our sample to a much larger pool of programmers with complete commit information for each of their collaborators to provide meaningful results.

A second interesting idea that emerges from this work, is to enable individuals and organizations to explore and understand their collaboration networks. Whether through homophily or through contagion, activated collaboration networks certainly enable the adoption of programming languages. It would be useful to use this data to enable programmers to visualize their own collaboration networks, so that they can understand their influencers and learn better from them. This idea is explored in more detail in the appendix, and provides the basis for the platform "CodeSpace".

The number of programmers on Github has now crossed 25 million, making it more populous than three-fourths of the countries in the world. Studying the mechanics of contagion on these platforms will play a key role in advancing our understanding of knowledge diffusion. Progress on characterizing these diffusion patterns will also aid in attempting to enhance the diffusion of technologies [24]. Engineering contagion is not a new idea, and is frequently used in social media advertising [23]. Employing similar strategies may help open source libraries to increase adoption and target communities to enhance contagion.

As we move toward a world where larger teams collaborate on ever advancing technologies, our ability to effectively learn new technologies will be paramount. This research contributes insights towards the larger goal of understanding knowledge diffusion in collaboration networks.

7 Appendix: CodeSpace

The CodeSpace platform (http://codespace.media.mit.edu) visualizes the learning paths of over 500,000 developers from Github. The platform consists of a PostgreSQL database, a NodeJS based API layer and a ReactJS based front-end.

7.1 Dataset construction

The dataset was constructed using Github APIs, through snowball sampling from a list of 256 high activity github users. Since the Github APIs are rate limited, the dataset was constructed over several weeks, traversing over 500,000 users and fetching all commits from their open source personal repositories.

Each time a repository's commits were retrieved, every new collaborator was added to the queue and the process was repeated for the collaborators. The API presented numerous challenges, requiring a complex set of scripts that asynchronously fetched the data while maintaining a balanced load on the Github API as a good API citizen.

The dataset contains over 600,000 developers, close to 3 million repositories and over 108 million commits and continues to grow. It is stored in a highly optimized PostgreSQL database and contains a large set of materialized views that pre-compute various slices of the data to be rapidly served to the CodeSpace front-end.

7.2 CodeSpace Architecture

The web-based platform, "CodeSpace" is built with multiple complementary goals. Since software development is one of the most rapidly evolving innovation ecosystems, programmers need to frequently switch to newer technologies to remain competitive in the market. Thus, enabling programmers to visualize and understand their historical learning paths and key collaborators will help them understand the techniques that seemed to work for them in the past. This is one of the key goals of CodeSpace.

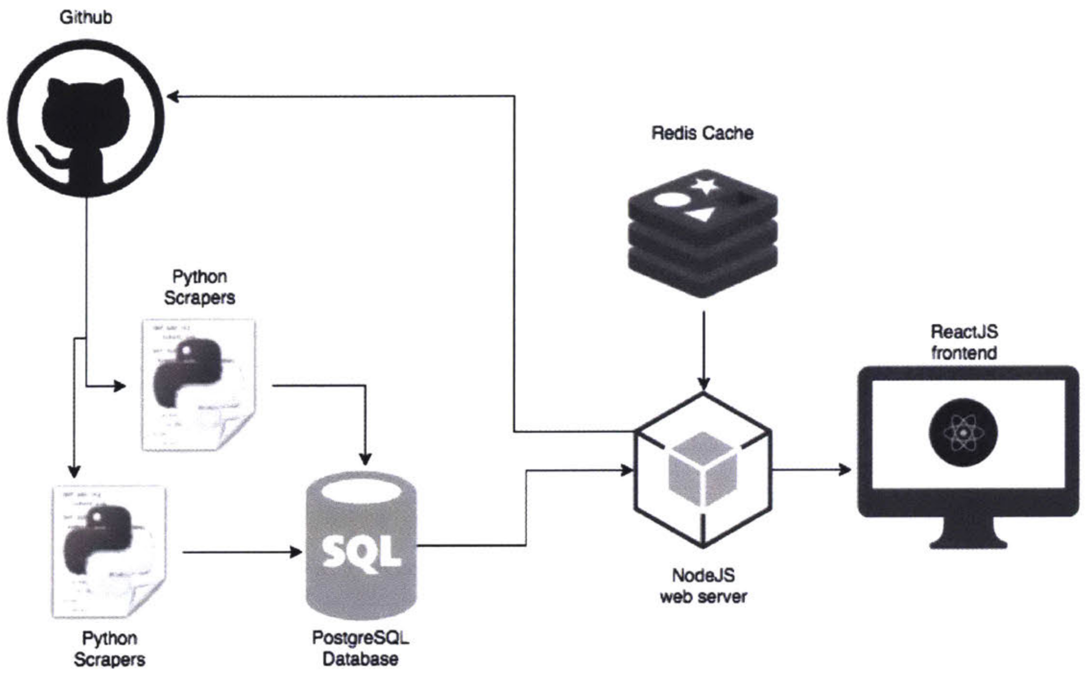
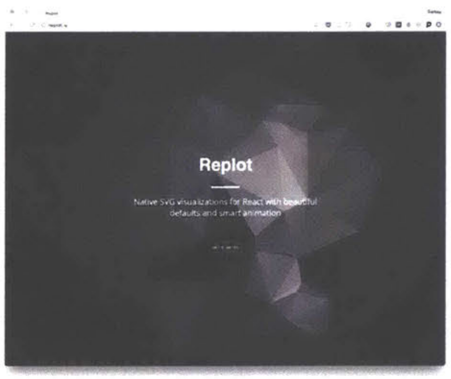


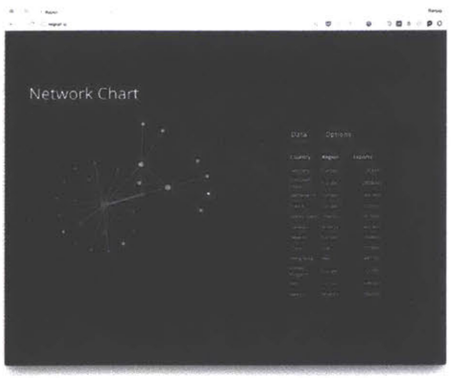
Figure 2: Codespace architecture diagram

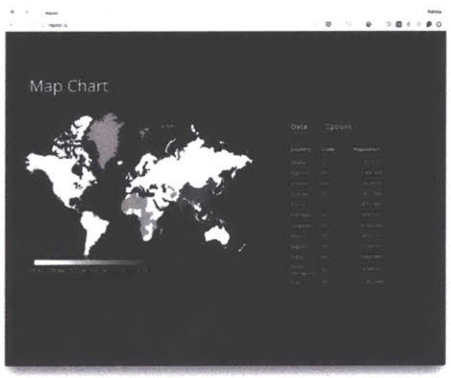
The technical architecture of the platform is fairly complex, as it involves multiple server side processes, an extremely large data layer, and a front end with complex, data heavy visualizations. The server side is built as a collection of dockerized containers, with API scrapers written in Python using asyncio and asyncpg for extremely fast and highly concurrent data collection, as well as a NodeJS web server, a large PostgreSQL database and Redis for caching.

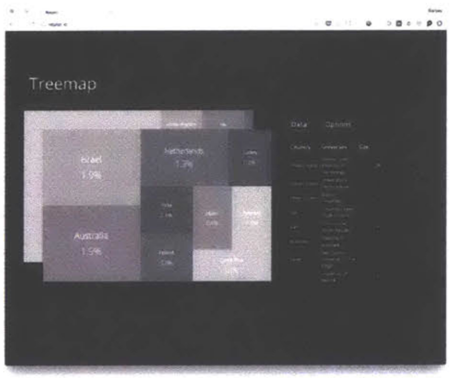
7.3 Replot - Open Source Visualization Library

The front-end of CodeSpace is powered by a visualization library called "Replot" (http://replot.io) developed by a team of collaborators I led at the Collective Learning Group. While there are a number of libraries that port popular visualization libraries like d3 to ReactJS, Replot uses a different approach, rendering SVG components directly using React. This required us to build every component from the ground up. Using this React first approach will lead to numerous benefits in the future as browsers begin to optimize for React's virtual DOM based rendering.


Replot
Native SVG visualizations for React with beautiful defaults and smart animation







I led a team of contributors in creating Replot, collaborating with Alisa Ono, Almaha Almalki, Matthew Farejowicz, Ariel Levy and Nancy Luong. The Replot framework has been deployed to npm and open-sourced as part of this thesis. It powers multiple visualizations in both the CodeSpace and Opus platforms.

Replot consists of seven types of visualizations — TreeMaps, Bar Charts, Box Plots, Line Charts, Scatter Plots, Network Charts and Maps. Each of these visualizations can be customized and extended in numerous ways using props. All components have built in animations and automatically animate changes in the data. This enables a paradigm where Replot visualizations can be used to easily build a "data timelapse" — an extremely useful feature when visualizing time series data. A common example is visualizing the evolution of a network over time.

8 References

[1] Pastor-Satorras, R. and Vespignani, A. (2001)

Epidemic spreading in scale-free networks.

[2] Weng, L., Menczer, F., & Ahn, Y.-Y. (2013)

Virality Prediction and Community Structure in Social Networks.

[3] Christakis, N. A, & Fowler, J. H. (2008)

The collective dynamics of smoking in a large social network.

[4] Christakis, N. A, & Fowler, J. H. (2008)

The Spread of Obesity in a Large Social Network over 32 Years.

[5] Granovetter, M. S. (1973)

The Strength of Weak Ties.

[6] Rogers, E. M. (1971)

Diffusion of Innovations.

[7] Hidalgo, C. A., Klinger, B., Barabási, A.-L. & Hausmann, R. (2007)

The product space conditions the development of nations.

[8] Kitsak, M., Gallos, L. K., Havlin, S., Liljeros, F., Muchnik, L., Stanley, H. E., & Makse, H. A. (2010).

Identification of influential spreaders in complex networks.

[9] Watts, D. J., Dodds, P. S. (2004)

Universal Behavior in a Generalized Model of Contagion.

[10] Watts, D. J., Dodds, P. S. (2007)

Influentials, Networks, and Public Opinion Formation.

[11] Daley, D. J., & Kendall, D. G. (1964)
Epidemics and Rumours.

[12] Centola, D. M., & Macy, M. (2007).
Complex Contagions and the Weakness of Long Ties.

[13] Centola, D. (2010)
The spread of behavior in an online social network experiment.

[14] Aral, S., & Nicolaides, C. (2017)
Exercise contagion in a global social network.

[15] Bahar, D., Hausmann, R. & Hidalgo, C. A. (2014)
Neighbors and the evolution of the comparative advantage of nations: evidence of international knowledge diffusion?

[16] Guevara, M. R., Hartmann, D., Aristarán, M., Mendoza, M. & Hidalgo, C. A. (2016)
The research space: using career paths to predict the evolution of the research output of individuals, institutions, and nations.

[17] Meyerovich, L. A., & Rabkin, A. S. (2013)
Empirical Analysis of Programming Language Adoption.

[18] Github Annual Report (2017)
The State of the Octoverse.

[19] Lima, A., Rossi, L., & Musolesi, M. (2014)
Coding Together at Scale: GitHub as a Collaborative Social Network.

[20] Casalnuovo, C., Vasilescu, B., Devanbu, P., & Filkov, V. (2015)
Developer Onboarding in GitHub: The Role of Prior Social Links and Language Experience.

[21] StackOverflow (2018)

Stack Overflow Developer Survey 2018.

[22] WP Engine Survey (2017)

How do developers feel about programming languages?

[23] Aral, S., Muchnik, L. & Sundararajan, A. (2013)

Engineering social contagions: optimal network seeding in the presence of homophily.

[24] Alshamsi, A., Pinheiro, F. L. & Hidalgo, C. A. (2018)

Optimal knowledge diffusion strategies in the networks of related products and of related research areas.

[25] Newman, M., Barabasi, A., Watts, D. J. & Boguñá, M. (2013)

The Structure and Dynamics of Networks.

[26] Ugander, J., Backstrom, L., Marlow, C., & Kleinberg, J. (2012)

Structural diversity in social contagion.

[27] Aral, S., Muchnik, L., & Sundararajan, A. (2009)

Distinguishing influence-based contagion from homophily-driven diffusion in dynamic networks.