Evolutionary Modeling of a Blog Network

Telmo Menezes
CREA & ISCPIF
CNRS
ISC - 57-59, rue Lhomond
F-75005 Paris, France
telmo@telmomenezes.com

ABSTRACT

Blog networks are cases of complex networks that emerge from the interactions of agents. A common approach to produce theory to explain the genesis and dynamics of such networks is to create multi-agent simulations that output networks with similar characteristics to the ones derived from real data. For example, a well know explanation for the power law degree distributions found in blog (and other) networks is the agent-level endogenous mechanism of preferential attachment. However, once simplifying assumptions are dropped, finding lower level behaviors that explain global network features can become difficult. One case, explored in this paper, is that of modeling a network generated from agents with heterogeneous behaviors and *a priori* diversity. We propose an approach based on an hybrid strategy, combining a generic behavioral template created by a human designer with a set of programs evolved using genetic programming. We present experimental results that illustrate how this approach can be successfully used to discover a set of non-trivial agent-level behaviors that generate a network that fits observed data. We then use the model to make successful testable predictions about the real data. We analyze the diversity of behaviors found in the evolved model by clustering the agents according to the execution paths their programs take during the simulation. We show that these clusters map to different behaviors, giving credence to the need for exogenous, in addition to the more conventional endogenous explanations, for the dynamics of blog networks.

Categories and Subject Descriptors

J.4 [**Social and Behavioral Sciences**]: Sociology; I.6.5 [**Simulation and Modeling**]: Model Development—*Modeling methodologies*; I.2.2 [**Artificial Intelligence**]: Automatic Programming—*Program synthesis*

General Terms

weblogs, social networks, complex systems, multi-agent simulations, genetic programming

1. INTRODUCTION

Blog networks are an interesting subset of Internet-based social networks. They allow for the emergence of mostly unregulated and decentralized content-generating communities, organized both around common interests and social ties. As is the case with many other classes of complex networks, non-trivial global structures emerge from the local interactions of agents. A common approach to modeling this type of complex system is to develop multi-agent simulations, trying to find low-level agent behaviors that produce global metrics that fit the ones observed in the real world. The value of these models lies in both its explanatory and predictive capabilities. On one hand, they allow us to understand which type of individual behaviors lead to the global behaviors. On the other hand, they allow us to test scenarios, for example: how will global phenomena be affected by changes of behavior at the agent level, or how will the system evolve over time.

Modeling efforts require a simplification of the observed reality. A problem with devising multi-agent simulations that explain the genesis and dynamics of a complex networks is that, as we try to consider more aspects of the observed networks, it becomes increasingly difficult for a human modeler to find low-level behaviors that lead to a simulation that fits that observed reality. For example, current simulation of web page networks make the implicit simplifying assumption that all agents have the same fundamental behaviors, and that asymmetries only emerge from phenomena like preferential attachment [3]. While it is clear that preferential attachment and similar phenomena play an important role in web and blog network formation, it is also clear that the agents in these networks are not homogeneous in their behavior. Agents differ, for example, in their posting rates and their tendency to cite other posts from other agents in the network [14]. By observing the content of the posts at the semantic level, it can also be observed that agent differ in their level of interest on different topics [5] or their tendency to generate or follow topics [15].

In this paper we propose an approach that combines human modeling with an evolutionary search for specific low-level behaviors that fit the observed data.

In the next section we discuss works related to the ideas we propose in this paper. In section 3 we present our hybrid model for agents in a blog network, combining a human designed template with a set of evolved programs. We also

present the evolutionary algorithm used to evolve the models and a description of the fitness function used to make the evolved model approximate real data. In section 4 we describe the setup and results of an experiment where we evolve a multi-agent model to explain the emergence of a real blog network. We finish in section 5 with some conclusions and final remarks. In appendix A, we provide the full set of programs for the best model found in the experiment described in section 4.

2. RELATED WORK

2.1 Modeling Blog Networks and the Web

The effort to model blog networks is preceded by the more generic attempt at modeling the web [13, 22]. Preferential attachment [3, 4] is the most well known agent-level behavior used to explain the emergence of the scale-free characteristics of many complex networks, including blog networks. In [8], a multi-agent generative model for networks of blogs, based on preferential attachment, is proposed.

In [7], generic blog networks are also modeled as multi-agent systems, combining preferential attachment with a random walk algorithm to decide when agents create new posts. This algorithm is shown to generate exponential post inter-arrival times that fit what is observed in real data [10].

Other models were developed to tackle more specific questions relating to blog networks. For example, the model described in [23] deals with blog mortality.

2.2 Heterogeneity in Blog Networks

There are several examples in the literature of works that explore the idea that blog networks are maintained by agents with heterogeneous characteristics. The differences between agents are found at several levels, including at the structural level of linking behavior and the semantic level of the posts' contents.

At the structural level, in the work described in [14], blog types are revealed by classification according to the topology of the cascades they are involved in, as well as their temporal activity. In [6], the different personality traits of bloggers are analyzed and related to what the bloggers write about and their motivations for doing so. In [1], differences between blogger behaviors are shown depending on their political affiliation. In [19], four different forms of communication between bloggers and blog commentators are identified.

2.3 Evolutionary Models

In a work with some parallels to the ideas presented in this paper, scientific laws are extracted from experimental data using genetic programming [20]. The similarity between this work and ours is that, in both cases, we are using evolutionary search and genetic programming to automate the search for theory that fits the data. In the case of [20], the generated theory takes the form of laws in the form of mathematical equations. In our case, the generated theory is a set of programs that describe the low level behavior of agents that, collectively, generate a network with similar characteristics to some observed real world network.

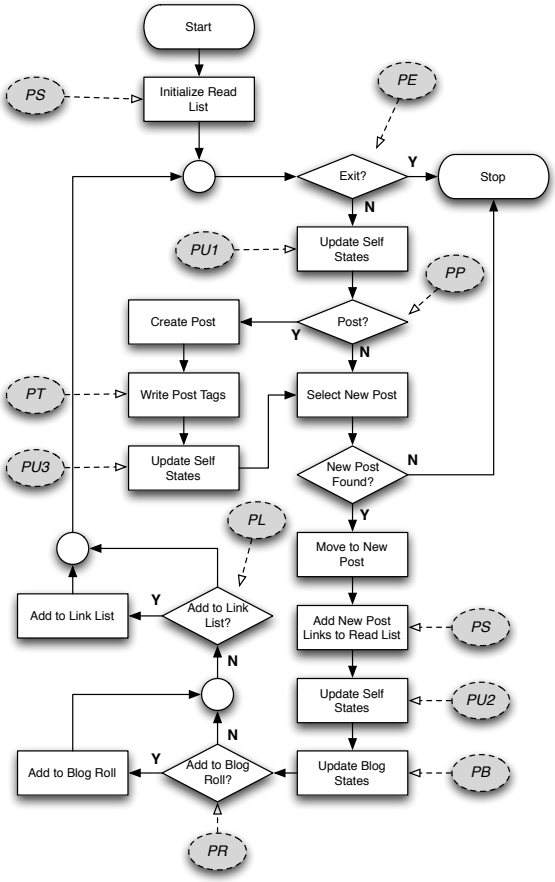

Figure 1: Flowchart describing the simulation cycle of an agent.

3. EVOLUTIONARY MODELING

3.1 Agent Model

We propose a multi-agent model where each agent simulation cycle follows a template based on common sense assumptions about the behavior of a blogger. This template is described by the flowchart shown in figure 1. Although the flowchart looks complex, the underlaying assumptions are quite simple. They can be summarized as follows: during a simulation cycle, and agent spends some times reading posts generated by other bloggers. The blogger maintains a list of blogs she likes, which serve as the starting point for navigation. We will refer to that list as the *blog roll*[1]. From there, she may proceed to posts cited by the posts she is reading or just move to another blog in the *blog roll*. During this process, she may accumulate links to posts that she intends to cite later on her own posts. At some point during the navigation of the blog network, she may decide to create a new post, containing the accumulated citations,

[1]Many blogs maintain a list of links to other blogs, not associated with a specific post, called a *blog roll*. Blog rolls may contain blogs for several reasons: topical similarity, social proximity or just because the blogger likes them. Here, we use blog roll in the more abstract sense of the list of blogs that a blogger regularly follows, by whatever process, irrespectively of that preference being made public or not.

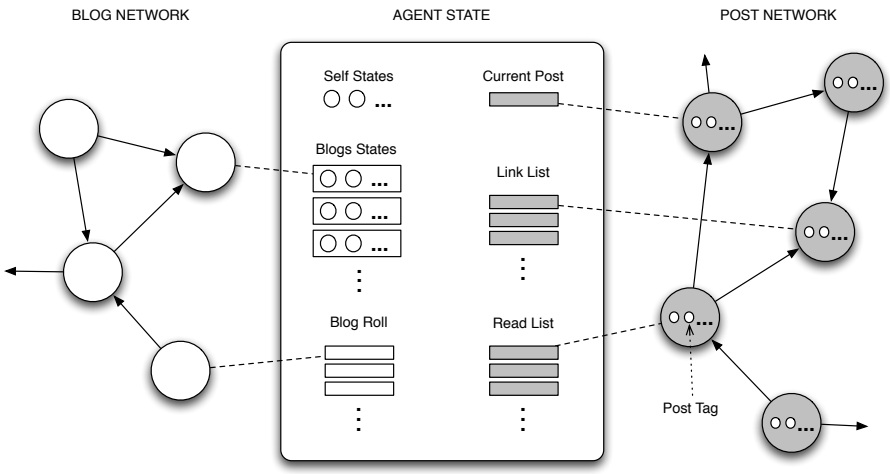

Figure 2: Detail of the agent states and its relationships with the blog and post networks.

if any. While reading a post, the blogger may decide to add the blog to which that post belongs to her *blog roll*. Eventually, the blogger will decide to terminate her activity for the cycle. These assumptions are empirically based on the observation of the activity of human bloggers. We tried to avoid introducing unnecessary complexity, but also not lose sight of the most relevant aspects of this type of activity.

This simple behavioral template can accommodate a wide range of specific behaviors, because the decision to take any of the described actions is determined stochastically, with the probability of its occurrence being computed by a program. Through evolutionary search we will look for the set of programs that generates the behaviors that best fit observed metrics of a real network. The specific technique we use to evolve the programs is tree-based *genetic programming* [18]. In figure 1, programs are represented by their code inside a gray ellipse, with an arrow pointing to the stage in the flowchart that they influence.

Agents have an internal state, which is detailed in figure 2. One of the elements of this state is the *blog roll*, already discussed. Another one if the *read list*, where the next post to read is selected from. This list is initialized with the last post from each blog in the agent's *blog roll*. When a new post is visited, all the posts linked from this post are added to the *read list*. Also, a visited post is flagged, so that it cannot be selected again as the next post to read in the current simulation cycle.

After reading a post, there are two actions that the agent may take: adding the post in its *link list*, and adding the blog that owns the post to its *blog roll*. The *link list* is the list of posts to be cited by the next post created by the agent. The decision to create a new post can be taken before selecting the next post to read. In that case, the post is created with citations to all posts currently in the *link list*, and then this list is returned to an empty state. As can be inferred, the agent can create several posts in a simulation cycle.

Another important aspect of our model is that the agent's behavior may be influenced by the current state of the agent, and in some cases by the context defined by the current state of the blog being visited and the specific characteristics of the post currently being read (which we will call *post tags*). These states and tags are represented by vectors of floating point numbers, and are provided as input variables to the programs. The states and tags are also depicted in figure 2. The agent's internal state is a vector S of size s, the state of blog i (from the perspective of the agent) is vector B_i of size b and the specific characteristics of a post j are defined by a vector T_j of tags, of size t. These states and tags have no *a priori* meaning. They are meant as a mechanism for the agent to regulate its behavior according to the context of its previous actions and the part of the network being visited. Of course, it is possible to imagine scenarios where these values could be used to represent features of the real system. For example, the internal state of the agent could store its level of tiredness. An agent could become more tired when generating a post and tiredness level could decrease with time and affect the probability of posting. Blog states could represent the current view of the agent on the quality level of a blog, according to the posts that it has previously read from that blog. Post tags could represent the type of topics being addressed by the post. Although no meaning is ever imposed, after the evolutionary process one may attempt to attach meaning to the several components of the state and tag vectors, according to the way they are used by the evolved programs.

For the states and tags mechanism to work, it is not sufficient that these values are provided as inputs to the programs that determine action probabilities. It is also necessary that there is some way for the values to be initialized and updated. In fact, there is one set of programs that is responsible for initializing agent states (PI_1 to PI_s) and three set of programs that are responsible for updating these states ($PU1_1$ to $PU1_s$, $PU2_1$ to $PU2_s$ and $PU3_1$ to $PU3_s$). These are sets of programs, because there is one program for each component of the S vector. The PI set of programs is not shown in figure 1, as it is not used during the simulation cycle. It is executed once for each agent when the simulation

is initialized. The $PU1$ set is executed after a decision to continue reading posts is taken, and takes only the current agent state as context. $PU2$ is executed after a new post is selected for reading, and also takes the tags for that post and the state of the post's owner blog as input. $PU3$ is executed after the agent creates a new post, and only uses the agent states as inputs. This diversity of updates programs allow for the agent to update is internal state in distinct ways during different stages of its simulation cycle, according to a context that is relevant for each stage. Another important aspect of the update programs is that they are able to disable themselves, according to a mechanism that we will detail later. If disabled, no update to the respecting state is performed. We found experimentally that this mechanism facilitates the finding of high quality solutions by the evolutionary process. Here, a biological analogy can be established with the process of *gene regulation*. Parts of the genetic code that determines the agent behavior are expressed conditionally, according to environmental conditions and the current state of the "organism".

In a similar vein, the PB set of programs updates the state of the blog owning the current post being read. All blog states are initialized to zero in the beginning of the simulation. The PT set of programs is used to generate post tags, after a new post is created.

The only program left to be described is PS, which is responsible by attributing a score to posts. All posts that enter the *read list* are attributed a numerical score. These scores are used to decide which post to read next. Being R the set of all posts belonging to the *read list* and that were not already visited, the probability $p(R_i)$ of post R_i being selected as the next post to read, where $S(R_j)$ is the score attributed by PS to post R_j is given by equation 1.

$$p(R_i) = \frac{S(R_i)}{\sum_{R_j \in R} S(R_j)} \quad (1)$$

Table 1 enumerates all the programs used by the model.

It is important to be aware of a number of simplifications that we still assume in this work. For example, the blog network under study is not isolated from the entire web. In reality, bloggers may link to blogs and sites outside of the network under study and, also, discover blogs or post in the network under study by way of external web pages or using a search engine. We intend to address these limitations as we continue to explore the ideas we propose here.

3.2 Program Details

Programs use an *S-Expression* representation common to tree-based genetic programming. Each program is thus defined by a tree, where each non-terminal node is a function, and each leaf is a constant or variable. A program can take multiple input variables, and is recursively evaluated so that a single output value is produced.

The basic function set used by the programs consists of simple arithmetic operations (*sum*, *subtract*, *multiply*, *divide*) and comparison functions (*greater than*, *lesser than*, *greater or equal than*, *lesser or equal than*, *is zero*). Comparison functions apply the comparison to the first two branches (or only the first in the case of *is zero*), and then evaluate to the second-to-last or last branche depending on the result. We will call this function set *vanilla*, as shown in 1.

The *random* function set adds two functions to the *vanilla* set: *rand_uniform* and *rand_normal*. The first one generates a random value in the $[0, 1]$ interval, while the second one generates a random value from a standard normal distribution ($\mu = 0, \sigma^2 = 1$).

The *ON/OFF* function set also adds two functions to the *vanilla* set: *ON* and *OFF*. These functions only take one parameter, and return its value unchanged. They have the side effect of setting the program state to *active* or *inactive*, as previously discussed. The programs that use this set are initialized to the *inactive* state.

The number of variables available to a program depend on its inputs, as detailed in table 1. Programs that take agent states as inputs have s variables available, one for each element in the agent vector. Programs that also take the current blog state and current post tags as inputs have an additional $b + t$ number of variables available.

Constants are real values in the $[0, 1]$ interval.

The PI program set is, in a way, a special case. As discussed, it is used to initialize agent states, and is not part of the simulation cycle. Furthermore, it takes no inputs, and includes function in its function set to generate random values. Since we are assuming that the agents are behaviorally heterogeneous, the PI programs play the important role of allowing for a diversification of the initial agent states. By combining the random number generators with the other functions, they can evolve to assign initial values stochastically, from complex probability distributions.

3.3 Running a Simulation

A simulation based on a model is run for a predefined number of cycles. For each simulation cycle, the set of agents is iterated through, so that each performs one cycle as described previously. In the current work, we are not accounting for agents entering or leaving the system, so the number of agents remains constant throughout the simulation. In fact, we chose a number of agents that matches the number of blogs constituting the real network that we are trying to model.

The simulation is initialized with no posts and, consequently, no links between blogs. However, given the described behavioral template, there has to be some way for the agents to initially find each other. For this purpose, we initialize each agent's *blog roll* with a predefined number of references to other randomly selected blogs in the network.

One problem we have to address is computational limitations. Given the unbounded nature of the domain of possible behaviors expressed by the evolved programs, some of these programs will lead to models where agents tend to read a very large number of posts each cycle, or create a very large number of posts. This type of models will have large compu-

Table 1: List of Programs Used to Control Agent Behaviors

Program(s)	Inputs	Output	Function Set
$PI_1..PI_s$	None	Initial blog state	Random
PE	Agent State	Probability of ending the cycle	Vanilla
PP	Agent State	Probability of posting	Vanilla
$PU1_1..PU1_s$	Agent State	New self states values	ON/OFF
$PU2_1..PU2_s$	Agent State, Current Blog State, Current Post Tags	New self states values	ON/OFF
$PU3_1..PU3_s$	Agent State	New self states values	ON/OFF
$PB_1..PB_b$	Agent State, Current Blog State, Current Post Tags	New blog states values	ON/OFF
PS	Agent State, Blog State, Post Tags	Post score	Vanilla
PR	Agent State, Current Blog State, Current Post Tags	Probability of adding current blog to roll	Vanilla
PL	Agent State, Current Blog State, Current Post Tags	Probability of adding current post to link list	Vanilla
$PT_1..PT_p$	Agent State	New post tag values	Vanilla

tation times, which will reflect on the time taken to process a generation and thus, hinder the evolutionary process. To sidestep this problem, we define global limits for the number of posts read and new posts created by all agents in the simulation. If this limit is reached, the simulation simply stops. The limits are defined to be large enough to contain the behaviors of the scale found in the real data, filtering only models that are certainly unrealistic.

3.4 The Evolutionary Algorithm

We use a conventional evolutionary algorithm. A fixed-sized population of models (represented by the set of programs) is randomly initialized. This initialization consists of generating random programs using the *ramped half-and-half* approach [11]. For each generation, the fitness of each model is evaluated by running the model for a number of simulation cycles, and then comparing the simulation output with real data. The details of this comparison will be described in the next subsection.

The individuals used to generate the next population are chosen using *tournament selection* [16]. These individuals can be recombined and mutated, according to pre-defined probabilities.

Recombination consists of randomly selecting one of the programs from the set, and then performing *subtree crossover* with the equivalent program on the other parent. All other programs are copied unchanged from the first progenitor. Mutation is also performed in one of the programs, randomly selected from the set, with the other ones remaining unchanged. We employ the *headless chicken* mutation strategy [2].

3.5 The Fitness Function

The fitness function we use is a distance between the final status of the simulated network and the status of a network extracted from real data. We classify each node in both networks according to k metrics. For example, in the experiments we will discuss later we used three metrics: in-degree, out-degree and the number or posts created by the node during the observation period. We selected these metrics for both their simplicity and their ability to expose the heterogeneity in agent behaviors.

For each metric m_j, nodes are classified with a discrete value $n_j \in \{1, ..., q\}$, where q is a predefined constant specifying the number of possible classifications.

The vector of observed values in the real data for metric m_j is sorted and divided into q-quantiles. The classification n_j of a given node (in real or simulated data) is simply the number of the q-quantile to which its m_j value belongs. We found that a low value of q helps prevent the evolutionary process to get stuck in local minima, by giving more leeway for exploration instead of rewarding excessively early overfitting to the real data. In our experiments we used $q = 3$ [2].

The metrics-based classifications are used to generate an overall node classification for a node a, according to the expression in equation 2.

$$c(a) = \sum_{i=1}^{k} q^{i-1}.(n_i(a) - 1) \quad (2)$$

For a given network N, we can now determine how many nodes correspond to a classification i, as shown in equation 3.

[2]Apart from being a low value, this was an arbitrary choice, but it fits nicely with the classifications of *low*, *medium* and *high*.

$$C_i(N) = |\{a \in N : c(a) = i\}| \quad (3)$$

Finally, we compute the distance between two networks as the number of nodes that do not have the same classification in both networks.

The fitness function, shown in equation 4, is the distance between the simulated network (N_s) and the real network (N_r).

$$f = \frac{\sum_{i=0}^{q^k-1} |C_i(N_r) - C_i(N_s)|}{2} \quad (4)$$

Since the simulated network is initialized to a state where no posts or links between blogs exist, we allow for an initialization period in the simulation. Supposing we decided to run the simulation for t_{total} cycles, we define a number of $t_{initialize}$ cycles during which we collect no metrics. Metrics used by the fitness function are thus only collected for the $]t_{initialize}, t_{total}]$ period. This is derived from the assumption that the real data we have at our disposal respects to some period of study of a mature network, not contemplating its initial stages. We are also assuming that there is not a relevant number of agents entering or leaving the system during the period of study.

We cannot thus claim that our evolved model corresponds to a realistic representation of the initialization stage of the network, but only of its more mature and stable observed state. Given data describing the state of a network since its initial stages, it could be feasible to define a fitness function that led to models that defined more realistic theories on the process occurring at the birth of a network. This is, however, not a focus of the work being presented. Here, we will only claim that we can evolve a model that has some generative process that leads to a mature state similar to the one observed in the real system.

4. RESULTS AND DISCUSSION

4.1 Experimental Setup

The previsouly described methodology was applied to evolve a model against real data extracted from a crawl of the French political blogosphere, consisting of 916 blogs[3], between the days of October 1^{st} 2009 and February 11^{th} 2010. During this period, 40,191 posts were published, containing 16,909 citation links to other blogs in the network. We assume each simulation cycle to correspond to a time period of one day. We run the simulations for periods of 300 cycles, with an initialization period of 170 cycles, and the final 130 cycles roughly matching the observation period of the real network.

We run the evolutionary algorithm for 500 generations, with each generation consisting of a population of 1000 models. Each model contains 916 agents, matching the number of blogs in the real network. We use a recombination probability of 0.8, a mutation probability of 0.6 and a tournament size of 5. This parameter set for the evolutionary algorithm was determined empirically, resorting to some initial experimentation.

[3]These blogs were selected by human experts to be a good representation of the French political blogosphere.

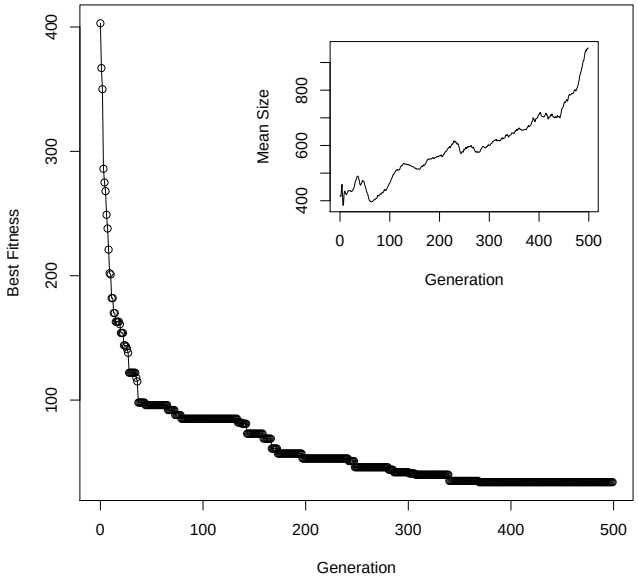

Figure 3: Fitness improvement during the evolutionary process. Inset: mean size of the programs per generation.

We chose a size of 2 for the agent states, blog states and post tag vectors. We chose this size because we wanted to see if the evolved programs would take advantage of more than one variable for each one of these states, and use them in different ways. At the same time, we did not want to specify a larger number, that could lead to harder to interpret results. The relationship between the size of the state vectors and the quality and nature of the evolved models is still an open question, that we intend to address in the future.

4.2 General Analysis of Results

In figure 3, we can observe the fitness improvement during the evolutionary process. The best fitness achieved was a distance of 34 between the simulated network and the real one. This represents a rate of correct classifications of the final state of network nodes, according to the method described in section 3.5, of approximately 96.29%.

In the inset of figure 3 we show the evolution of the mean size of the programs per generation. The increase in this value throughout the successive generations suggests an increase in complexity of the models throughout the evolutionary process. We cannot, however, be certain that this increase in complexity is useful. As, we will see in a following section, part of the complexity of the evolved programs is attributable to *bloat*. *Bloat* is a common phenomena in genetic programming, commonly described and an useless increase in the size of evolved entities [12]. We leave the

Table 2: Comparison of estimated power-law exponents: real data vs. simulation

Metric	Simulation α	Real Data α
Post Count	1.254 ± 0.014	1.234 ± 0.013
In-Degree	1.623 ± 0.022	1.537 ± 0.02
Out-Degree	1.737 ± 0.026	1.745 ± 0.026

evolution of useful complexity in this method as an open question for future study.

We wanted to test if the evolved model was able to make correct predictions about features of the real network that were not directly contemplated by the fitness function. To this end, we compared the simulated and real estimated exponents (α) of the power-law distributions of the post counts, in-degree and out-degree of blogs. This comparison is shown in table 2. The exponents were estimated by fitting the observed data to power-law distributions, using maximum likelihood, as recommended in [17]. Indeed, it can be observed that the estimated exponents of the distributions generated by the simulation are good approximations of the ones observed in real data.

4.3 Evolved Programs

Here we discuss the programs that constitute the evolved model. The complete listing of these programs can be found in appendix A.

Firstly, the post scoring program (PS) is simply a constant, meaning that all posts in the agent's *read list* have the same probability of being selected. The probability of posting is directly given by *agent state* 1. *Agent state* 1 is initialized by the simple uniform distribution in the $[0, 1]$ interval, and remains unchanged, as all the programs that could change it are always inactive. The more complex behaviors are found in the mechanisms of determining the probability of adding the current post to the *link list* and the current blog to the *blog roll*, both depending on *agent state* 0 as well as the context of the current *blog state* and current *post tags*. Furthermore, the value of *agent state* 0 can be updated during the simulation cycle in a non-trivial way.

Agent state 0 changes during the simulation, and its meaning is less obvious than that of *agent state* 1. It seems to be strongly related to the more complex mechanisms defined by the program set. The initialization program for *agent state* 0 has an approximate probability of 0.57 of generating the constant value 0.775, or extracting its value from a standard normal distribution otherwise. *Agent state* 0 can have its value updated after a post is read (by program $PU2_0$) and after a new post is created (by program $PU3_0$). After a post is read, the decision to update *agent state* 0 depends on *agent state* 1 and *post tag* 1. The new value is generated taking into account the values of *blog state* 1, *agent state* 1 and *post tag* 0. $PU3_0$ is the most complex program in the set, but depends only on *agent states* 0 and 1.

The programs PB_0 and PB_1, which update blog states, are also fairly complex. Both use *agent states* and *blog states* as inputs, and PB_0 also uses *post tags* 0 and 1 as inputs. As for the programs that generate *post tags*, it is interesting to notice that PT_0 uses only *agent state* 0 as input, and PT_1 only *agent state* 1. This means that *post tag* 1 is completely determined by the initialization value of *agent state* 1 (which never changes), while *post tag* 0 reflects a state of the agent at the moment the post was created that reflects the agent's previous activities in the simulation.

4.4 Execution Paths

One of the motivations we presented for this work was the ability to model a system made of agents with heterogeneous behaviors. It is thus interesting to analyze the evolved model from this perspective. The programs that define the model include conditional execution functions in their function set. By observing the listings in appendix A, it can be verified that these functions are extensively used by the evolved programs. We propose that it is possible to determine if two agents have qualitatively different behaviors, by determining if they consistently take different branches at some of these functions.

To perform this analysis, we recorded the choices taken by each program during the simulation at every condition function. All our conditional functions have an *if-else* structure, resulting in the program taking one of two possible branches each time they are evaluated. Observing the program evaluation for a number of cycles, we can then determine if the program always takes one of the branches, or if it can take both. These possibilities configure three possible branching behaviors at each conditional function. For each agent a, we can consider a vector $B(a)$ that contains the branching behavior value for each conditional function in the set of programs. We will call this vector the *execution path* of the agent. It is then possible to compute the *execution path distance* ($\delta_{aa'}$) between two agents a and a', as the number of positions where the vectors $B(a)$ and $B(a')$ are different. Being b the number of conditional functions in the model's program set, the execution path distance is formalized in equation 5.

$$\delta_{aa'} = |\{i \in \mathbb{N} : 1 \leq i \leq b \wedge B_i(a) \neq B_i(a')\}| \qquad (5)$$

As discussed before, not all of the complexity of the evolved programs is necessarily useful. Before computing execution paths, we applied a simplification strategy to programs based on a dynamical analysis of their behavior while being evaluated in the context of the simulation. First we run the entire simulation, recording the values that nodes evaluate to during program execution. If a node is found to evaluate to a constant value throughout the entire simulation, we replace that node, and the subtree for which the node is a root, with a constant value node. This value is the same that we found the node to evaluate to in every case. The stripped version of the programs is functionally equivalent to the previous one, in the context of the simulation. This simplification step has two advantages. The first is that it removes meaningless distinctions from the execution path analysis. The second is that it facilitates the human interpretation of the programs, by removing useless complexity. In fact, it is the simplified version of the programs that we present in appendix A.

Agents were grouped into clusters, according to their execution path distances, using the k-medoid approach of *partitioning around medoids* [21]. The optimal number of clusters (k) was estimated by *optimum average silhouette width* [9].

As can be observed in figure 4a), the number of execution path clusters was estimated to be 6, with most of the agents falling into 4 of those clusters. In the following plots of figure 4 we show the distributions of the three metrics we considered in the fitness function over each one of the clusters. These distributions indicate different combinations of behaviors in terms of number of posts and links generated by agents in each cluster. We can conclude that to the differentiation of behaviors in terms of execution paths corresponds a differentiation of behaviors as measured by the *number of posts* and *out-degree* metrics.

This diversity of behaviors appear to arise both from the initial agent state differentiation generated by the initialization programs and by self-organisation phenomena during the simulation. It was not imposed in any way by the fitness function, but evolved as a likely explanation for the observed data, giving credence to our hypothesis of a need for exogenous *a priori* agent diversity in addition to the more traditional endogenous explanations for blog network dynamics. Also, the low number of different clusters of behaviors generated suggests that the evolved model is a credible description of the observed system, as opposed to *ad-hoc* over-fitting to the real data, that could be made possible by the high degree of freedom allowed by our system.

5. CONCLUSIONS

In this paper we proposed an hybrid modeling approach to multi-agent systems, specifically applied to the problem of modeling the dynamics of a blog network generated and maintained by heterogeneous agents. A high-level, stochastic behavioral template for agents is defined by the human modeler, and then regulated at each step by evolutionarily generated programs, and their interactions with agent and blog states, as well as post tags. We described an experiment where we tested this approach by evolving a model against the data extracted from a crawl of the French political blogosphere. The outcome of this experiment was encouraging for a number of reasons.

Firstly, we found that the evolutionary algorithm is indeed capable of a progressive decrease in the distance between the real and the simulated network. Although we believe there is room for improvement, the result we obtained of a 96.29% correct node classification according to the three metrics we considered, already leads us to believe that the model we evolved in this work is a useful approximation of the network under study.

Secondly, the programs that define the evolved model display non-trivial behaviors. It seems clear that it would be hard for a human modeler to arrive at the type of mechanisms described by these programs. It is also interesting to notice that the programs do make use of the *agent state*, *blog states* and *post tags*, that the initialization programs generate a diversity of values for the *agent state* and that the elements of each state and tag vectors are used by the programs for distinct purposes. This gives credence to our

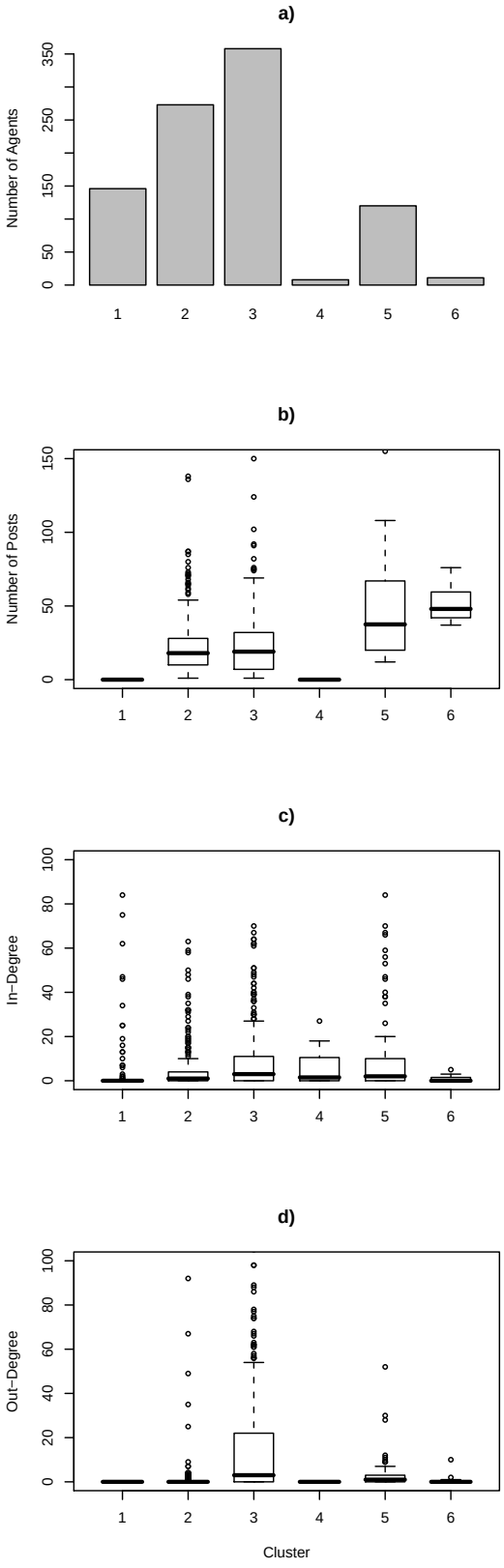

Figure 4: Several metrics according to execution path cluster: a) number of agents in cluster; b) post count distributions; c) in-degree distributions; d) out-degree distributions.

initial assumption that these features would be useful to the system.

Thirdly, we showed that the evolved model is capable of making correct predictions about the real network, as the power-law exponents of the distributions of the several metrics in the simulated network are very close to the exponents observed in the real network, although the fitness function does not directly promote this similarity.

Lastly, we were able to observe that the evolved model generates agents that can be grouped into distinct types of behaviors, both at the level of program evaluation and at the level of measurable outcomes, like the posting rate and the tendency to link to other posts in the network. This represents a success in accordance to our initial objective of evolving a multi-agent model with heterogeneous agent behaviors, and our hypothesis that *a priori* diversity adds to the explanatory power of endogenous phenomena alone.

We would like to finalize by pointing out that the methodology proposed in this work may be adapted to other types of complex networks, both inside and outside the domain of Internet social media.

6. ACKNOWLEDGMENTS

This work has been partially supported by the French National Agency of Research (ANR) through grant "Webfluence" #ANR-08-SYSC-009.

The author would like to thank Camille Roth and Jean-Philippe Cointet for their feedback and constructive criticism of this work.

7. REFERENCES

[1] L. A. Adamic and N. Glance. The political blogosphere and the 2004 u.s. election: divided they blog. In *LinkKDD '05: Proceedings of the 3rd international workshop on Link discovery*, pages 36–43, New York, NY, USA, 2005. ACM Press.

[2] P. J. Angeline. Subtree crossover: Building block engine or macromutation? In J. R. Koza, K. Deb, M. Dorigo, D. B. Fogel, M. Garzon, H. Iba, and R. L. Riolo, editors, *Genetic Programming 1997: Proceedings of the Second Annual Conference*, pages 9–17, Stanford University, CA, USA, 13-16 July 1997. Morgan Kaufmann.

[3] A. Barabasi and R. Albert. Emergence of scaling in random networks. *ArXiv Condensed Matter e-prints*, Oct. 1999.

[4] A.-L. Barabasi, R. Albert, and H. Jeong. Scale-free characteristics of random networks: The topology of the world-wide web, 2000.

[5] J.-P. Cointet and C. Roth. Socio-semantic dynamics in a blog network. In *IEEE Intl. Conf. Social Computing*, pages 114–121, 2009.

[6] A. Gill, S. Nowson, and J. Oberlander. What are they blogging about? personality, topic and motivation in blogs. 2009.

[7] M. Goetz, J. Leskovec, M. Mcglohon, and C. Faloutsos. Modeling blog dynamics. In *International Conference on Weblogs and Social Media*, May 2009.

[8] A. Karandikar, A. Java, A. Joshi, T. Finin, Y. Yesha, and Y. Yesha. Second space: a generative model for the blogosphere. In *ICWSM 2008: International AIII Conference on Weblogs and Social Media*, 2008.

[9] L. Kaufman and P. J. Rousseeuw. *Finding Groups in Data: An Introduction to Cluster Analysis (Wiley Series in Probability and Statistics)*. Wiley-Interscience, March 2005.

[10] J. Kleinberg. Bursty and hierarchical structure in streams. In *KDD '02: Proceedings of the eighth ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 91–101, New York, NY, USA, 2002. ACM.

[11] J. R. Koza. *Genetic Programming: On the Programming of Computers by Means of Natural Selection*. MIT Press, Cambridge, MA, USA, 1992.

[12] W. B. Langdon. The evolution of size in variable length representations. In *Proc. of the 1998 IEEE International Conference on Evolutionary Computation*, pages 633–638, Anchorage, Alaska, 1998. IEEE Press.

[13] M. Levene, T. Fenner, G. Loizou, and R. Wheeldon. A stochastic model for the evolution of the web. *Computer Networks*, 39:2002, 2002.

[14] M. McGlohon, J. Leskovec, C. Faloutsos, M. Hurst, and N. Glance. Finding patterns in blog shapes and blog evolution. In *ICWSM 2007: International AIII Conference on Weblogs and Social Media*, 2007.

[15] T. Menezes, C. Roth, and J.-P. Cointet. Precursors and laggards: An analysis of semantic temporal relationships on a blog network. In *SocialCom10: Proc. of the 2010 IEEE International Conference on Social Computing*, Minneapolis, USA, 2010.

[16] B. L. Miller, B. L. Miller, D. E. Goldberg, and D. E. Goldberg. Genetic algorithms, tournament selection, and the effects of noise. *Complex Systems*, 9:193–212, 1995.

[17] M. E. J. Newman. Power laws, pareto distributions and zipf's law. *Contemporary Physics*, 46(5):323–351, May 2006.

[18] R. Poli, W. B. Langdon, and N. F. McPhee. *A field guide to genetic programming*. Published via `http://lulu.com` and freely available at `http://www.gp-field-guide.org.uk`, 2008. (With contributions by J. R. Koza).

[19] C. Prieur, D. Cardon, H. Delaunay-Téterel, and C. Fluckiger. Self expression as a relational technique, four types of personal blog networks, 2006.

[20] M. Schmidt and H. Lipson. Distilling free-form natural laws from experimental data. *Science*, 324(5923):81–85, April 2009.

[21] S. Theodoridis and K. Koutroumbas. *Pattern Recognition, Third Edition*. Academic Press, Inc., Orlando, FL, USA, 2006.

[22] M. Vafopoulos, E. Amarantidis, and I. Antoniou. Modeling Web Evolution. *ArXiv e-prints*, Jan. 2010.

[23] G. Venolia. A matter of life or death: Modeling blog mortality. Tech report, Microsoft Research.

APPENDIX

A. PROGRAM LISTINGS

Here we present the programs from the evolved model in a simple Lisp-like notation. This type of notation is the most natural representation for programs based on *S-Expressions*.

$\$SELF\_0$ and $\$SELF\_1$ are the two variables corresponding to the two components of the agent state. $\$BLOG\_0$ and $\$BLOG\_1$ correspond to the two components of the current blog state and $\$POST\_0$ and $\$POST\_1$ to the two tags of the current post.

A.1 PI_0

```
(<= 0.204022
    (- 0.771396 RAND_UNIF) RAND_NORM 0.775062)
```

A.2 PI_1

```
RAND_UNIF
```

A.3 PE

```
(>= $SELF_0
  (/ $SELF_1 0.422206)
  (<= 0.436451 $SELF_1 0.589641 $SELF_0) $SELF_0)
```

A.4 PR

```
(>= $BLOG_0 $SELF_1 $POST_0 0.0125404)
```

A.5 PL

```
(>= (+ $POST_0 0.543933) $SELF_1
  (>= 0.294613 $SELF_0 $BLOG_0 $POST_1) 0.110038)
```

A.6 PP

```
$SELF_1
```

A.7 PB_0

```
(>= (>= $BLOG_1 $SELF_1 0.438753 0.0930686)
  (/ 0.0930686
    (>= (>= 0.68306 $SELF_1 0.0930686 $SELF_1)
      0.298139
      (+ $SELF_0
        (<= (ZER $POST_1 $POST_0 $POST_0)
            0.312165 0.966044 $POST_1))
      (ON 0.68306)))
  (<= (ZER $POST_1 $POST_0 $POST_0)
    0.312165 0.966044 $POST_1) 0.311976)
```

A.8 PB_1

```
(- (+ (<= $BLOG_0 0.871431 0.0940904 0.137115)
    (<= $BLOG_1 $SELF_1 $SELF_1 0.0528009))
  (* (ON $BLOG_0)
    (<= 0.744341 $SELF_0
      (- $POST_1 $SELF_1) 0.139808)))
```

A.9 $PU1_0$

Inactive

A.10 $PU1_1$

Inactive

A.11 $PU2_0$

```
(>
  (OFF $SELF_1)
  (* 7.23054 $POST_1) 1.26306
  (<= $BLOG_1
    (ON 0.573527)
    (+ $BLOG_1 $SELF_1) $POST_0))
```

A.12 $PU2_1$

Inactive

A.13 $PU3_0$

```
(>= (OFF (* 0.509936 $SELF_1))
  (<= (* (/ $SELF_1 0.996491) $SELF_0)
    (+ 0.85273
      (<= $SELF_1 0.83792
        (<= (ON 0) $SELF_0
          (- $SELF_1
            (ON 0.36302)) 0.769006) 0.104445))
    (>= 0.997519 $SELF_0
      (ON 0.471153) 1)
    (+ 0.0189248
      (OFF (* 0.509936 $SELF_1))))
  (<= (ON 0) $SELF_0
    (- $SELF_1
      (ON 0.36302)) 0.769006)
  (>= $SELF_0 0.0337225
    (<= 0.208748 $SELF_0 $SELF_0 0.65818)
    (ON 0)))
```

A.14 $PU3_1$

Inactive

A.15 PT_0

```
(-
  (<=
    (-
      (<=
        (-
          (<= $SELF_0 0.333656 0.61046 $SELF_0)
          0.333656)
        0.333656 0.61046 $SELF_0)
      0.333656) 0.333656 0.61046 $SELF_0)
  0.333656)
```

A.16 PT_1

```
(>=
  (- $SELF_1 0.167926) 0 0 $SELF_1)
```

A.17 PS

```
0.210522
```