Evolutionary Algorithms for the Self-Organized Evolution of Networks [*]

Katharina A. Lehmann
lehmannk@informatik.uni-tuebingen.de

Michael Kaufmann
mk@informatik.uni-tuebingen.de

Wilhelm-Schickard-Institute, University Tübingen
Sand 14
72076 Tübingen, Germany

ABSTRACT

While the evolution of biological networks can be modeled sensefully as a series of mutation and selection, evolution of other networks such as the social network in a city or the network of streets in a country is not determined by selection since there is no alternative network with which these singular networks have to compete. Nonetheless, these singular networks do evolve due to dynamic changes of vertices and edges. In this article we present a formal, analyzable framework for the evolution of singular networks. We show that the careful design of adaptation rules can lead to the emergence of network topologies with satisfying performance in polynomial time while other adaptation rules yield exponential runtime. We further show by example how the framework could be applied to some ad-hoc communication scenarios.

Categories and Subject Descriptors

F2.2 [**Analysis of Algorithms and Problem Complexity**]: Nonnumerical Algorithms and Problems—*Computations on discrete structures*; G2.2 [**Discrete Mathematics**]: Graph Theory—*Network Problems*; I6.5 [**Simulation and Modeling**]: Model Development

General Terms

Algorithms

Keywords

Evolutionary Algorithm, Self-Organization, Evolution of Networks

[*]Supported by DFG grant Ka812/11-1

1. INTRODUCTION

Theoretical work on the evolution of organisms gave rise to the new field of evolutionary algorithms, developed independently by many scientists with different perspectives [1, 4, 10]. The theory of evolution does not only give an explanation to the evolution of organisms as a whole but also of the evolution of subparts of organisms on a smaller level: Within the organisms also different types of networks have evolved, like protein-protein interaction networks [12], or metabolic networks [5]. Their evolution is conducted by mutations of the DNA and subsequent selection of the organisms with respect to their fitness. The evolution of these networks is well modeled by the general framework of evolutionary algorithms as described in [4, 9]. As such, the evolution of these networks is distributed over many organisms and no network as an individual survives forever. Instead, only the information for the generation of a successful network is passed on from generation to generation. Since networks are finite and can be displayed as a set of vertices and an adjacency matrix where entry $a[i][j]$ states that vertices i and j are connected by an edge, the evolution of a network towards an optimal topology can be viewed as a combinatorial optimization process.

The networks we are interested in this article are different from those evolved in biological organisms [7]: They are singular, i.e., at every timestep there is only one network, and their global fitness is not evaluated by some entity from the outside but by the vertices within the network. These can then change parts of the network as a result of the evaluation. Thus, the network's topology evolves in a decentralized and self-organized fashion but the network itself cannot be replaced by another network as a whole. As examples for the evolution of this kind of singular and self-organized networks we want to discuss shortly the evolution of the network of streets in a country and evolution of the global social network in which all of us participate: Singular networks are built between rational entities and these entities are able to evaluate the network's current performance with respect to their own position in the network. In the case of social networks a person might evaluate how well her position is in the network by how much information she gets from her social environment, while the network of streets in a country might be evaluated from each of the connected cities with respect to the average time it takes to get to any other city. The consequence of such an evaluation process is that vertices unsatisfied with the network's current topology will try to adapt the network to improve their situation, using some adaptation rule. In our examples this might cause a person to make new friends, or a city to build a new freeway. Since most edges in real-world networks come with a cost, the number of changes in each time step will not be too big and every vertex may only remove or build edges it participates in. Another important point to note is that in most cases it will not be totally clear for the unsatisfied vertex which edges should be changed to improve the performance. This can be easily seen in the case of social networks where a person is only able to survey a small part of the whole network, implying that the change of edges is partially a random process. In summary, singular networks evolve in a series of evaluation processes from single vertices within the network, making small changes of their local edge sets in a random process defined by some adapation rule.

Despite this randomness and the decentralized evolution conducted by single vertices within the network, many properties of evolving singular networks seem to be quite stable, e.g., the so-called small-world effect in social networks [8, 13]. These observations made on real-world networks gave the motivation to formulate an abstract framework for the self-organized evolution of singular networks. The framework provides a way to describe this kind of self-organized evolution and to analyze different adaptation rules with respect to the expected runtime until a network with a satisfying topology has emerged. This kind of analysis has also been conducted for other evolutionary algorithms and helps to decide when these kind of algorithms can be efficiently applied [9, 14, 11].

On the other hand the framework provides a way to implement well chosen instances of the framework on small communication devices to evolve ad-hoc communication networks with satisfying topologies in a self-organized way.

In Sec. 2 we present the formal framework for the evolution of singular and self-organized networks. To show both, the analyzability and applicability of the model, we will then introduce two different instances of the framework and analyze the behaviour of the evolving network under these instances in Sec. 3, followed by an instance with locally restricted evaluation functions that may be interesting for the design of self-organized communication networks in Sec. 4. We conclude the article with a discussion in Sec. 5.

2. THE MODEL

We give an evolutionary algorithm that describes the evolution of networks with the following properties:

1. There is only one, singular network at any time step.
2. The vertices evaluate the network's fitness relative to their own position in the network.
3. No vertex knows the adjacency matrix of the whole network.
4. The network evolves so slowly that at any time step only one vertex is active.
5. All vertices decide independently which edges to build and which to remove.
6. Each vertex behaves selfishly, i.e., it will built a new edge to improve its own situation, not to improve the situation of others. This implies that a vertex will only remove edges it participates in and can only insert edges it participates in.

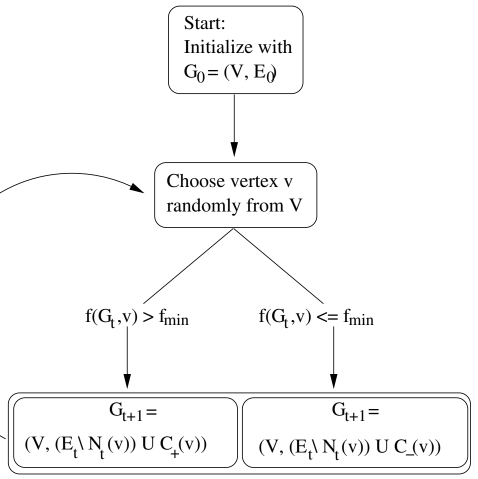


Figure 1: The framework for the evolution of S^3 networks. The model begins with a graph $G_0 = (V, E_0)$. In each time step t one of the vertices evaluates the fitness of G_t with respect to itself by calculating the value of $f(G_t, v)$. If this is higher than a given minimal value f_{min}, the vertex will change its current neighborhood $N_t(v)$ to a neighborhood given by the changing rule $C_+(v)$. If the evaluation yields a value smaller or equal than the minimal value f_{min} the vertex changes its current neighborhood according to $C_-(v)$.

The evolution of networks with these properties will be called evolution of **singular**, **selfish** and **self-organized** **(S^3)** networks. In the following paragraph we will formalize the evolution of these networks.

Let $G_t = (V, E_t)$ denote the graph at time step t where V is a set of vertices and E_t is a subset of $V \times V$. All edges $e = (u, v)$ will be regarded as undirected, i.e., $(u, v) \in E_t$ iff $(v, u) \in E_t$. $\mathcal{G}$ denotes the set of all possible graphs on the set of vertices V. In this first introduction of the model, V is static. The discussion in Sec. 5 will deal with possible extensions for dynamic vertex sets. The fitness of any graph $G \in \mathcal{G}$ is given by $f : \mathcal{G} \times V \to \mathbb{R}$, i.e., the performance of any graph is evaluated with respect to one vertex of the graph. This leaves room for designing a 'selfish' or 'egoistic' evaluation function. As we will discuss later, the formulation allows that the value of $f(G, v)$ is the same for all vertices but as we understand our model it is important to use functions that in principal give different values for different vertices. Further, a desired, minimal performance is defined for every vertex v by $f_{min} : V \to \mathbb{R}$. This is the general case. In the following cases we will use a constant value f_{min} for all vertices.

Let $N_t(v)$ denote the set of edges adjacent to v in time step t: $N_t(v) = \{(u, v)|(u, v) \in E_t\}$. $\mathcal{E}(v)$ denotes the set of all possible edge sets $E(v)$ on $v \times \{V \backslash v\}$: $\mathcal{E}(v) = \{E(v)|E(v) \subseteq v \times V \backslash v\}$. Further, we define two functions that change the set of edges adjacent to v: $C_+ : V \to \mathcal{E}(v)$ and $C_- : V \to \mathcal{E}(v)$. These are also called the *changing rules* of the algorithm. Note, that every vertex can only change its own vertex set, i.e., it can only build and remove edges attached to itself. This rule - together with the evaluation of the network relative to itself - also reflects the selfishness of vertices.

The evolution of a network can now be modeled by the following steps (s. Fig. 1):

1. Initialize G_0 with (V, E_0).

2. In every time step t choose one vertex v randomly. If $f(G_t, v) > f_{min}(v)$ then the vertex will built a new set of adjacent edges $N_{t+1}(v)$ as the result of $C_+(v)$ otherwise the new set of adjacent edges is the result of $C_-(v)$. The new graph G_{t+1} is thus built as the old edge set without the old neighborhood of v, combined with the new neighborhood, i.e.,

$$G_{t+1} = (V, (E_t \backslash N_t(v)) \cup N_{t+1}(v)) \quad (1)$$

This model describes an iterative process of evaluating and manipulating the network's structure in a decentralized way and thus we consider it to be a kind of evolutionary algorithm [7]. Note that the function $f(G_t, v)$ can be seen as an objective function that is personalized to v and that it can be combined with any rules $C_+(v)$ and $C_-(v)$. In Sec. 3 we will give two instances of this evolutionary algorithm for a function $f(G_t, v)$ that is related to the diameter of a network and where the rules are designed to minimize the diamter. This is only to give a demonstration how the framework can be analyzed for some given personalized objective function $f(G_t, v)$ and rules $C_+(v)$, $C_-(v)$. However, as mentioned above, the framework itself works with any combination of objective function $f(G_t, v)$ and any set of $C_+(v)$ and $C_-(v)$.

In the following section we will show that the exact design of these rules can be important: Analyzing two sets of rules for minimizing the diameter of a tree, we can show that the expected runtime, i.e. the number of timesteps, until a favoured network structure arises, can be either exponential or polynomial with only minor changes in the rules.

3. TWO EXAMPLES

The following definitions are needed:

3.1 Definitions

A *path* $P_t(u, v)$ from vertex u to v is a set of vertices $\{u = v_1, v_2, \ldots, v_k = v\}$ with $(v_i, v_{i+1}) \in E_t$. The *path length* is defined as the number of edges on the path. The *distance* $d(u, v)$ between two vertices u and v is defined as the minimal path length of any path between vertex u and v. $d_t(u, v)$ denotes the distance of two vertices u, v in graph G_t. The *degree* $deg_t(v)$ of vertex v in time step t is given as the cardinality of $N_t(v)$: $deg_t(v) = |N_t(v)|$. The *eccentricity* $ecc(G, v)$ of a vertex v in graph G is defined as the maximal distance of v to any other vertex y:

$$ecc(G, v) = \max_{y \in V} d(v, y) \quad (2)$$

The *diameter* $D(G)$ of a graph G is defined as the maximal eccentricity of any vertex v in the graph:

$$D(G) = \max_{v \in V} ecc(v) \quad (3)$$

Seen from the perspective of the vertex, the eccentricity represents something like a 'personalized diameter' of a graph. The *closeness centrality* $close(G, v)$ of a vertex v in graph G is defined as the sum over all distances from v to any other vertex y:

$$close(G, v) = \sum_{y \in V} d(v, y) \quad (4)$$

The *Wiener index* $W(G)$ of a graph G is defined as the sum over the distances between all pairs of vertices u, v. This equals the sum over all closeness centrality values:

$$W(G) = \sum_{v \in V} close(v) \quad (5)$$

A graph G is called a *connected tree* if there is exactly one path between any pair of vertices of u, v in the graph. Equipped with these definitions we can now introduce two instances of the evolutionary algorithm for S^3 networks and analyze their expected run-time behaviour.

3.2 Analysis of expected runtime

The goal of the following two algorithms is to provide the vertices in a network with adaptation rules that enable them to reduce the diameter of the network to a desired value. Since every vertex has only a limited view on the network it cannot know which edge will be the best to build. Both algorithms choose one vertex randomly in every time step. If its current eccentricity is greater than the desired diameter it follows that the current network has a diameter greater than the desired diameter. The vertex then tries to insert a new edge to improve its own situation.

It is clear that different adaption rules might solve this problem, even a totally random insertion and removal of edges. Here, we will show that the design of efficient adaption rules is essential for the efficiency of the algorithm in constructing a network topology with the desired property. For this, we analyze the *expected runtime* of two algorithms given below, following the general idea of similar analyzes in [9, 14, 11]. The *expected runtime* denotes in this case the expected number of time steps t from G_0 to a graph G_t such that for all time steps $t' > t$ $G_{t'} = G_t$, i.e. the evolutionary process has come to a steady state.

We will now discuss the algorithms in more detail.

3.2.1 Algorithm 1

Let G_0 be a connected tree on a set of vertices V. Fitness of the tree is evaluated by the eccentricity of the chosen vertex:

$$f(G, v) = ecc(v) \quad (6)$$

f_{min} can be set to any value between $n - 1$, the maximal possible eccentricity, and 2, the minimal possible diameter for any tree with more than two vertices. In every time step t, choose one vertex v from V at random and calculate $f(G_t, v)$. Adaptation rule $C_+(v)$ is defined as follows (s. Fig. 2):

1. Choose one of the non-leaf vertices z in distance 2 to v at random. Let w be the vertex that is connected to both, v and z.

2. Generate a new graph $G_t^*(v, z)$ by replacing edge (v, w) by edge (v, z).

3. If $ecc(G_t^*(v, z), v) \leq ecc(G_t, v)$ then set $G_{t+1} = G_t^*(v, z)$, else $G_{t+1} = G_t$.

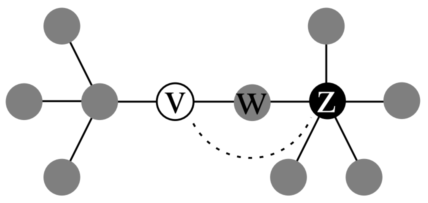


Figure 2: In algorithm 1, one vertex v is chosen at random from V in every time step. If its eccentricity is greater than f_{min} it will try to connect to a non-leaf vertex in distance 2 (black vertex). Let z be the second neighbor chosen and w be the vertex connecting both. Then edge (v, w) will be replaced by edge (v, z) if the eccentricity of v does not increase due to this process.

Note that we ignore second neighbors z with degree 1 (a leave), since they will always increase the eccentricity of v if the edge to z would be built.

This rule will maintain the graph's connectedness and thus, the graph will be a tree at any time. Furthermore, v will only remove edges that it participates in and can only initiate the building of edges it participates in. Thus, C_+ manipulates the direct neighborhood of v. There will be no change of the graph if $f(G_t, v) \leq f_{min}$, i.e., $C_-(v) = N_t(v)$.

In the following we want to show that the above given evolutionary algorithm will eventually build a tree with a diameter smaller or equal to f_{min}, independent of the initial tree. It is easy to see that a graph G_t with $D(G_t) \leq f_{min}$ will not change any more, and thus the runtime of algorithm 1 is defined as the number of time steps until such a tree is built. First, we will show that the process can be described as a Markov chain. We define the set S of states $\{s_1, s_2, \ldots, s_k\}$ as the set of connected trees on vertex set V. Since any subset of graphs on a given set of vertices is finite, S is also finite. Let every state s_i represent one connected tree $s_i = (V, E_i)$. State s_i is connected to s_j if their edge sets E_i and E_j obey the following relation:

$$\exists\, v, w, z \in V \mid E_j = (E_i \cup (v, z)) \backslash (v, w), \tag{7}$$

$$deg(z) > 1 \text{ and } ecc(s_j, v) \leq ecc(s_i, v) \tag{8}$$

The weight on this edge equals the probability that a tree s_i in time step t will be changed into tree s_j. This probability is equal to $1/n_2(v)$, where $n_2(v)$ denotes the number of non-leaf vertices w that are in distance 2 to v, also called the *non-leaf second neighbors of v.* Since this number is only depending on the structure of E_i, the transition probability between two states is independent from how the network has evolved and is thus constant in time. Note, that in the Markov chain the edges are directed, i.e., there is a designated source and target vertex. We will denote directed edges as $[s_i, s_j]$. Both edges, $[s_i, s_j]$ and $[s_j, s_i]$, exist iff E_i and E_j obey Equ. 7 and $ecc((V, E_i), v) = ecc((V, E_j), v)$. All states s_i with $D(s_i) \leq f_{min}$ will be denoted as *final states.*

We will first state that the network will eventually evolve into a network with a diameter smaller than f_{min}.

Lemma 1. *For $t \to \infty$, $D(G_t) \leq f_{min}$.*

Proof. If $D(G_0) \leq f_{min}$, then $\forall t : D(G_t) = D(G_0)$. Let now $D(G_0)$ be greater than f_{min}. Then, there is at least one vertex v with $ecc(G_o, v) > f_{min}$. We will now show that there is always a path from state s_0, representing G_0, to a tree $G_{t'}$ at time t' with $D(G'_t) \leq f_{min}$. Let z, z' be two vertices with maximal distance in G_t. Let $P(z, z') = \{z, z_1, z_2, \ldots, z'\}$ be the path between z and z'. If now z is being chosen it has an eccentricity greater than f_{min}, and if itself chooses its second neighbor z_2 then the eccentricity of z will not increase. Thus, edge (z, z_2) will replace edge (z, z_1). If in each time step only vertices with maximal eccentricity are chosen, this process will eventually lead to a tree with a diameter decreased by one and eventually to a tree with the desired diameter. The probability for such a way through the states is small but non-zero.

Since every tree having a diameter of at most f_{min} (final state) will not be changed any more and there is always a path with non-zero probability from every state s_i to some final state, the system will eventually reach one of these states and stay there. $\square$

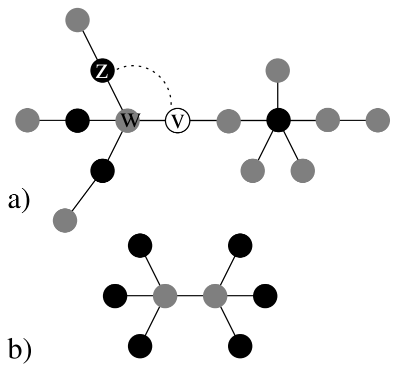


Figure 3: a) The graph has a diameter of 7, the chosen vertex v has an eccentricity of 4. If v now chooses second neighbor z and replaces edge (v, w) by (v, z), its eccentricity will not change but the diameter of the graph has actually increased to 8. b) All vertices in black have an eccentricity of 3 but even if f_{min} was to be 2, none can change the edge set if strict improvement of eccentricity was required for an edge replacement.

Fig. 3 a) shows that the rule given above holds some problems: Due to the change of perspective on the evaluation of the network in every step, it can easily happen that a step in the 'right' direction, i.e., to a tree with lower diameter, is reverted in the following step by another vertex that cannot 'see' the improvement of the last step. This is the one problem. Another problem arises if many vertices have to move together in the same direction before the diameter of the whole tree decreases. An example for such a situation is given in Fig. 3 b): If $f_{min} = 2$ is required, this can only be fulfilled if $n-1$ vertices are connected to one, central vertex, i.e. a star evolves. In the example given in Fig. 3 this implies that either of the three vertices of one side have to flip to the other side. The inner vertices will never change their edge set, and any outer vertex chosen will instantly flip sides because its only non-leaf second neighbor is the opposite inner vertex. After the first timestep there will be four vertices on the one side and two on the other. The only way to proceed to the star is that after this step one of the remaining two vertices changes side, and after that the remaining last vertex flips sides. Let x be the number of vertices on one

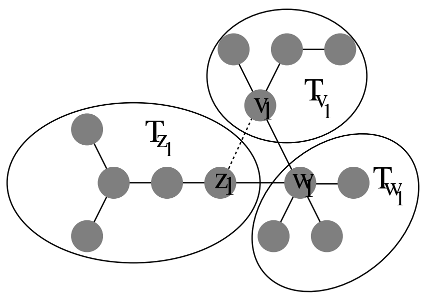


Figure 4: In the network depicted above, vertex v_1 has been chosen at random, and afterwards it chooses one of its second neighbors at random, in this case z_1. w_1 is the neighbor of both. v_1 will replace edge (v_1, w_1) by (v_1, z_1) if $|T_{z_1}| > |T_{w_1}|$.

side and $n-2-x$ the number of vertices on the other side. W.l.o.g. let $x \leq n-2-x$. The probability that any vertex of the minority flips sides is thus $x/(n-2)$ and the probability that one of the majority flips sides is $1-(x/(n-2))$. This situation can be described as the famous 'urn of Ehrenfest' model and we can use Ehrenfest's analysis on the expected number of steps until all vertices of one side have flipped which results in $\frac{1}{n-2}2^{n-2}(1+O(n-2))$ [2].

THEOREM 1. *There is a family of graphs such that with algorithm 1 and $f_{min} = 2$, the expected runtime is bounded by $\Omega(\frac{1}{n}2^n)$.*

Note that for $f_{min}(v) = 2$ any tree will evolve into this situation sooner or later.

From this, it seems obvious that the changing rule $C_+(v)$ is not constructed sensibly: Of course, backward steps could be prevented by allowing only those steps that lead to a strict improvement of the vertex' eccentricity. But, as Fig. 3 b) shows this small adjustment will eventually lead to trees in which none of the vertices can change anything despite the fact that the tree still has a diameter that is higher than desired. We will show next *how* the changing rule can be adapted such that it is possible to improve the expected runtime to $O(n^5)$.

3.2.2 Algorithm 2

Algorithm 2 does only differ in the formulation of $C_+(v)$ from Algorithm 1:

1. Choose one of the vertices z in distance 2 to v at random. Let w be the vertex that is connected to both, v and z.
2. Generate a new graph $G_t^*(v, z)$ by replacing edge (v, w) by edge (v, z).
3. If $close(G_t^*(v, z), v) < close(G_t, v)$ then set $G_{t+1} = G_t^*(v, z)$, else $G_{t+1} = G_t$.

As before, only vertices with an eccentricity higher than f_{min} will be allowed to change the current network. However, the decision, whether a new edge will actually replace one of the old edges is made by a comparison of the closeness centrality in the possible new tree with the closeness centrality in the current tree. Note, that this time the new value has to be strictly smaller than the value before. Nonetheless, the change in the formulation of $C_+(v)$ from algorithm

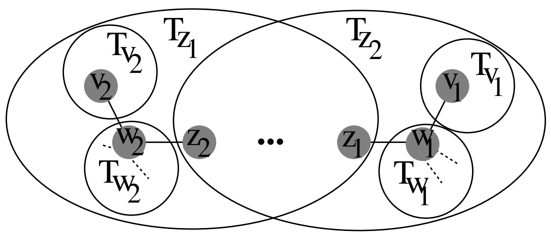


Figure 5: Vertices v_1 and v_2 are maximally distant to each other, i.e., $d_t(v_1, v_2) = D(G_t)$. As sketched in Fig. 4 the tree can be partitioned into three different subtrees for each pair of vertices v_1, z_1 and v_2, z_2, namely $T_{v_1}, T_{w_1}, T_{z_1}$ and $T_{v_2}, T_{w_2}, T_{z_2}$.

1 to 2 is actually quite small: Recall, that the eccentricity of a vertex v is the *maximal* distance of v to any other vertex w while the closeness centrality is the *sum* over all distances from v to any other vertex w. Thus, in the latter case, we use much more information about the current tree but the calculation of eccentricity and closeness centrality values in a tree takes the same time, namely $O(n)$, with a simple variant of a single-source shortest path algorithm [3]. We will now prove the following theorem:

THEOREM 2. *In Algorithm 2, the expected runtime is bounded by $O(n^5)$.*

PROOF. The proof is based on the following two lemmata.

LEMMA 2. *In every connected tree T with $D(T) > f_{min}$ there is at least one vertex v with a second neighbor z such that $close(G_t^*(v, z)) < close(G_t, v)$*

PROOF. Let G_t be a tree with $D(G_t) > f_{min}$. Let v_1 and v_2 denote two vertices with distance $d_t(v_1, v_2) = D(G_t)$. It is clear that these vertices have to be leaves, i.e., vertices with only one edge. Let w_1 and w_2 be the respective neighbor vertices of v_1 and v_2. The path $P(v_1, v_2)$ between v_1 and v_2 will contain w_1 and w_2: $P(v_1, v_2) = \{v_1, w_1, z_1, \ldots, z_2, w_2, v_2\}$. Since v_1 and v_2 are in distance $D(G_t)$ of each other, all vertices neighbored to w_1 other than z_1 have to be leaves, otherwise v_1 and v_2 could not be in maximal distance within the graph. Let now vertex v_1 be chosen by the algorithm and try to insert an edge to z_1.

As can be seen in Fig. 4, the closeness centrality of vertex v_1 in $G_t^*(v_1, z_1)$ is given by:

$$close(G_t^*(v_1, z_1), v_1) = close(G_t, v_1) + |T_{w_1}| - |T_{z_1}| \quad (9)$$

where $|T_X|$ denotes the number of vertices contained in subtree T_X, as depicted in Fig. 4. Subtree T_X denotes the subtree containing vertex X that emerges if edges (v_1, w_1) and (w_1, z_1) are removed from G_t. Equ. 9 states that the distance from v_1 to vertices from T_{w_1} is increased by one, and that the distance from v_1 to vertices from T_{z_1} is decreased by one in $G_t^*(v_1, z_1)$. If this new closeness centrality value is smaller than $close(G_t, v_1)$ then the new edge will be built and the old edge will be removed from the network and the case is proven. Let us now assume that the closeness centrality value in $G_t^*(v_1, z_1)$ is not smaller than the one in tree G_t. It follows, that

$$|T_{w_1}| \geq |T_{z_1}| \quad (10)$$

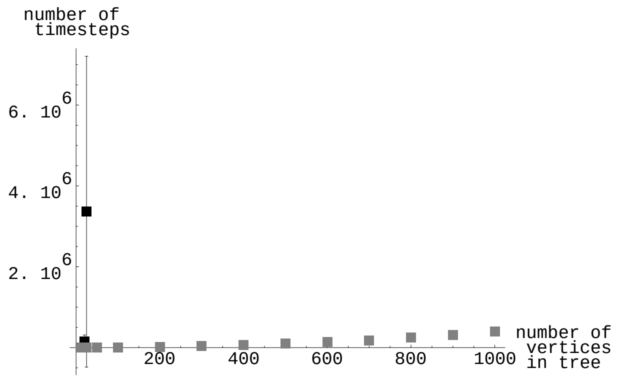


Figure 6: To test the runtime of algorithm 1 and algorithm 2 in worst-case scenarios, we began with chains of n vertices and $f_{min} = 2$, i.e., the network evolves to a star. The number of time steps needed by algorithm 1 are depicted as black boxes, algorithm 2 is represented by grey boxes. It can be clearly seen that algorithm 2 is not only much faster but also that the deviation in runtime is much less compared with algorithm 1. All points were sampled over 35 instances.

We will now show that in this case any vertex v_2 with $d(v_1, v_2) = D(G_t)$ would decrease its closeness centrality value by replacing edge (v_2, w_2) with (v_2, z_2) where z_2 is a second neighbor of v_2 on the path to v_1 (see Fig. 5).

$$close(G_t^*(v_2, z_2), v_2) = close(G_t, v_2) + |T_{w_2}| - |T_{z_2}| \quad (11)$$

As can be seen in Fig. 5, T_{z_2} can be expressed as follows:

$$|T_{z_2}| = |T_{z_1}| - (|T_{w_2}| + |T_{v_2}|) + |T_{w_1}| + |T_{v_1}| \quad (12)$$

Inserting this in Equ. 11 and regarding that $|T_{v_1}| = |T_{v_2}| = 1$, yields:

$$close(G_t^*(v_2, z_2), v_2) = close(G_t, v_2) + 2 \cdot |T_{w_2}| - (|T_{w_1}| + |T_{z_1}|) \quad (13)$$

With $|T_{w_1}| \geq |T_{z_1}| > |T_{w_2}|$ it follows that
$close(G_t^*(v_2, z_2), v_2) < close(G_t, v_2)$. □

We will now prove the following lemma regarding the non-increasing Wiener index of the evolving trees:

LEMMA 3. *Whenever G_{t+1} has emerged from G_t due to an edge replacement, then $W(G_{t+1}) < W(G_t)$.*

PROOF. Let G_{t+1} be evolved from G_t by the replacement of edge (v_1, w_1) with edge (v_1, z_1) according to algorithm 2. Let again denote $T_{v_1}, T_{w_1}, T_{z_1}$ the subtrees as depicted in Fig. 4. It is clear that the distance of vertices has only changed for pairs of vertices x, y where x is either in T_{w_1} or T_{z_1} and y is in T_{v_1}. The distance of vertices from T_{w_1} to vertices in T_{v_1} (and vice versa) has increased by one, the distance of vertices from T_{z_1} to vertices in T_{v_1} (and vice versa) has decreased by one. Thus, the Wiener index of the graph changes as follows

$$W(G_{t+1}) = 2 \cdot (|T_{w_1}| - |T_{z_1}|) \cdot |T_{v_1}| + \sum_{x \in V} close(G_t, x) \quad (14)$$

Since edge (v_1, w_1) has been replaced by (v_1, z_1) it is clear that $|T_{w_1}| < |T_{z_1}|$ and the lemma proven. □

Combining both lemmata now yields Theorem 2: The first lemma states that there is at least one pair of vertices v, z such that $G_t^*(v, z)$ has a smaller closeness centrality for v. The expected number of time steps until this pair is chosen is bounded from above by $O(n^2)$. The maximal Wiener index is bounded from above by $O(n^3)$ when the tree is arranged as a chain. Since every edge replacement implies a decrease of the Wiener index of at least 1, the expected runtime of algorithm 2 is thus bounded from above by $O(n^5)$. □

Fig. 6 shows a comparison of the runtimes of algorithm 1 and algorithm 2 for trees with different number of vertices. The two instances of evolutionary algorithms for the evolution of networks given above serve very well to show how a small change in the adaptation rule can make the difference between polynomial and exponential expected runtime. Although there might be applications for the algorithms introduced so far, they suffer from the need to flood the whole network to evaluate the eccentricity or closeness centrality of a vertex. In the next section we want to present an algorithm that uses a local evaluation function and shows very robust behaviour and interesting properties, but on the other hand is not as easy to analyze as the other algorithms.

4. LOCALLY RESTRICTED EVALUATION

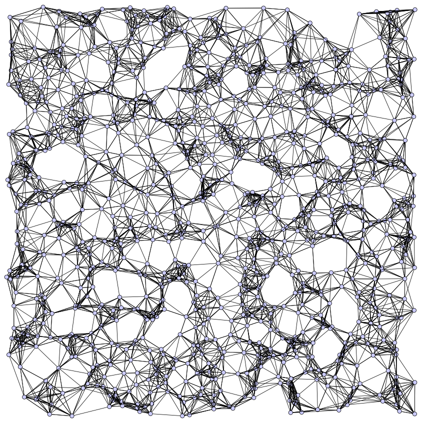
G_0 is initialized as a k-next-neighborhood graph G, where n vertices are placed in a metric space, e.g., Euclidian space, and every vertex is connected to its k geometrically next vertices.

In some scenarios, it might be required that every vertex holds as few edges as possible, e.g. to reduce the number of communications needed in a one-to-all broadcast, without increasing the diameter too much. It seems natural that triangles of the sort described above give a reasonable way to reduce the number of edges in the graph with the following algorithm.

4.1 Algorithm 3

Start with G_0, a k-next-neighborhood graph in a metric room. In every time step choose one vertex v at random. The fitness $f(G, v)$ is given by the number of triangles v participates in and f_{min} is set to 0. $C_+(v)$ chooses one of the triangles at random and removes one of the two edges connected to v.

Note that this rule will keep a connected network connected. It is especially appealing because it uses only information of the direct neighborhood of v. Furthermore, many of the operations could be conducted simultaneously without fearing any inconsistent information on the network. On the other hand, there are worst case scenarios in which the diameter will increase from $O(\sqrt{n})$ to $\Theta(n)$. Our empirical results have shown that this simple rule is nonetheless very robust and shows only slight increases of the original diameter. Furthermore, the number of time steps until nearly all



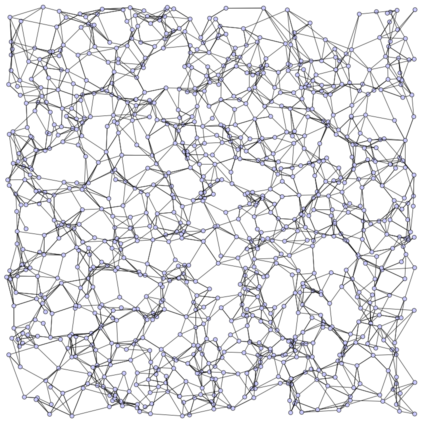
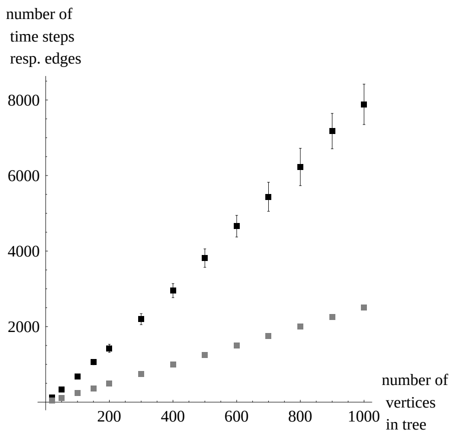


Figure 7: The left network is a k-next neighborhood graph with 1000 vertices, each connected to its ten geometrically next vertices. On the right, the same network is shown after the random removal of triangles as described in algorithm 3. The diameter of the network has increased from 24 to 30 while the average distance between vertices has only slightly increased from 10.4 to 12.5 .



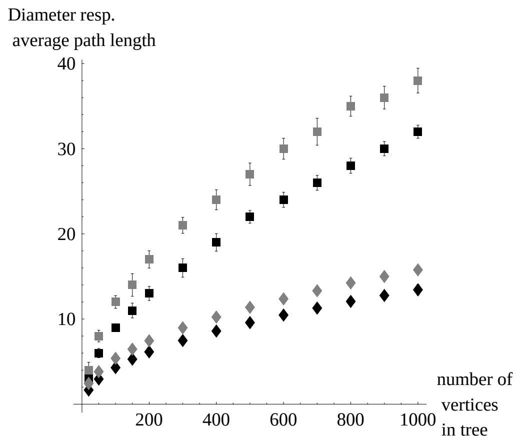


Figure 8: Empirical results for algorithm 3. All values have been obtained from 35 samples, each with n vertices, connected as a k-next-neighborhood graph with $k = 7$. Algorithm 3 was stopped after timestep t' when in n subsequent steps no change of the edge set had occured, the runtime is thus given by $t = t' - n$. The left diagram shows the runtime (black boxes) of algorithm 3. For fixed k it is linear with n and shows small deviation. Gray boxes indicate the number of edges after the process has stopped. The deviation is so small that it cannot be seen in this scaling. On average, the number of edges of the beginning graph $(n \cdot k)$ has been reduced to a third. The right diagram shows the diameter of the initial graph (black boxes) and after the process has stopped (grey boxes) and the corresponding average path lengths (black and grey diamonds). The empirical results show that the diameter and average path length is only slightly increased by the procedure despite the fact that approximately two thirds of the edges have been removed.

vertices have no more triangles they participate in is quite low (cf. Fig. 7 and Fig. 8).

Some ad-hoc communication networks like sensor networks are often modeled as unit-disk networks where every vertex is connected to all other vertices within its unit-disk [6]. If the devices are uniformly distributed this comes near to a k-next-neighborhood graph. To guarantee connectedness, k might be quite large at the beginning, but for most communication protocols it is better to reduce the number of edges in the graph subsequently. Here, the algorithm depicted above might be an interesting protocol to reduce the number of edges without increasing the diameter too much. Of course, the above given algorithm can easily be altered to guarantee a certain diameter: the chosen vertex could additionally evaluate its current eccentricity and then decide whether it will remove an edge. Similarly, vertices with an eccentricity that is too high could also begin to build new triangels. Further research will have to show which kind of rules are best applicable to which specific situation.

5. DISCUSSION

In this article we have presented a new class of evolutionary algorithms, suitable to model the evolution of those S^3 networks persisting over time where the constituent vertices change their edge set dynamically to adapt the network to a given situation. Two instances of this class were analyzed with respect to the expected runtime and it could be shown that the local adaptation rule can make the difference between an expected exponential or polynomial runtime. Additionally, we have presented a possible practical implementation of such an evolutionary algorithm for adhoc communication networks and shown how stable it is for a given k-next-neighborhood topology.

The evolutionary algorithms presented here differ in some aspects from conventional evolutionary algorithms and they have some things in common with them. From one perspective, the evolution of networks is just a change of their adjacency matrix $A(G)$ and since there is only one parent and one offspring in each generation, these evolutionary algorithms could be regarded as (1+1) evolutionary algorithms ($(1+1)EA$). But there are three main differences: The first is that every vertex will only evaluate its own situation within the graph and will only try to improve its own situation at any time step (**Egoism**).

Second, the algorithm does not aim for minimality regarding the evaluation function $f(G, v)$ but only that it is low enough, i.e., $\sum_{v \in V} f(G, v) \leq n \cdot f_{min}$ (**Satisfying topology**). Regarding this might protect a network from becoming overly adapted to a given environment.

A third aspect is that the evaluation of the network is decentralized which can reduce the number of messages to be sent over the network in many cases (**Dezentralized Evaluation**). Note, that we define a calculation to be *decentral* if no vertex needs to know the whole adjacency matrix for its evaluation but only its own neighborhood. Thus, all functions that can be evaluated with a memory space of $O(n)$ and some communication protocol are suitable. It seems necessary to define the term *decentral* so wide because even a test for connectedness in a network needs a flooding protocol in which all vertices participate. Of course, the implementation of any rule will be the more interesting in a practical sense the more 'local' it is, i.e., the less communication has to take place in order to evaluate it (s. Sec. 4).

Although the general framework given above allows to construct normal $(1+1)EA$s, it seems most fruitful to us to regard the basic properties mentioned above for modeling or designing the evolution of self-organized networks.

In the framework given above the vertex set is static for reasons of simplicity. Most real-world networks will show both, a dynamic edge set and a dynamic vertex set, and this can easily be implemented in the framework, either by *moves of nature* that change the vertex set from outside or by adaptation rules that allow the vertices to incorporate new vertices into a network or to disconnect parts of the network. Along with the analysis of other adaptation rules and the incorporation of constraints, e.g., on the number of edges a vertex can hold, this sketches the field where our future research is directed to.

6. REFERENCES

[1] G. Box. Evolutionary operation: a method for increasing industrial productivity. *Appl. Stat.*, 6(2):81–101, 1957.

[2] P. Bremaud. *Markov Chains - Gibbs Field, Monte Carlo Simulation, and Queues.* Springer Verlag, 1st edition, 1999.

[3] A. Grama, G. Karypis, V. Kumar, and A. Gupta. *An Introduction to Parallel Computing.* Addison Wesley, 2nd edition, 2003.

[4] J. Holland. *Adaptation in natural and artificial systems.* Ann Arbor: The University of Michigan Press, 1975.

[5] H. Jeong, B. Tombor, R. Albert, Z. Oltvai, and A.-L. Barabási. The large-scale organization of metabolic networks. *Nature*, 407:651–654, 2000.

[6] F. Kuhn, T. Moscibroda, and R. Wattenhofer. Unit disk graph approximation. In *Proc. DIALM-POMC 2004*, 2004.

[7] K. Lehmann. Why simulating evolutionary processes is just as interesting as applying them. In *Proc. GECCO 2005*, 2005.

[8] S. Milgram. The small world problem. *Psychology Today*, 1:61–67, 1967.

[9] M. Mitchell. *An introduction to genetic algorithms.* The MIT Press, 1996.

[10] I. Rechenberg. *Evolutionsstrategie: Optimierung technischer Systeme nach Prinzipien der biologischen Evolution.* Frommann-Holzoog, 1973.

[11] J. Scharnow, K. Tinnefeld, and I. Wegener. The analysis of evolutionary algorithms on sorting and shortest paths problems. *Journal of Mathematical Modeling and Algorithms*, 4(3):349–366, 2004.

[12] A. Vázquez, A. Flammini, A. Maritan, and A. Vespignani. Modeling of protein interaction networks. *Complexus*, 1(1):38–44, 2003.

[13] D. Watts and S. Strogatz. Collective dynamics of 'small-world' networks. *Nature*, 393:440–442, June 1998.

[14] C. Witt. An analysis of the (μ+1) EA on simple pseudo-boolean functions. In *Genetic and Evolutionary Computation - GECCO 2004*, number 3103 in LNCS, pages 761–773. Springer, 2004.