**NetAttack: Co-Evolution of Network and Attacker**

**Nomenclature**

...index for generations (: Number of generations)

...index for rounds (: Number of rounds)

...index for attackers ( nodes per network)

...index for network( networks per generation)

...index for node( nodes per network)

...index for network indicator for attacker ( indicators)

...weight for choosing node to be attacked within strategy

...weight for choosing node to be attacked

...probability of attacking node

**Overall algorithm**

1. We start with a very **first** **generation** of attackers and networks. Their genes are initialized randomly and an attacker is randomly assigned to a network.
   1. All networks apply their rules for attaching new nodes to the network until the network is fully connected (or after some amount of rounds?)
   2. **Round 1** starts: the attackers calculate indicators on all nodes of their assigned network. Based on these rules, all attackers choose a node to be attacked according to their genome rules (see below). The node is removed from the network and the fitness of the attacker is calculated after the attack. The fitness is somehow aggregated over all rounds (e.g. by summing up).
   3. The network calculates network or nodal metrics (or metrics on the edges).
   4. Let’s assume the attacker has, by removing one node, also removed edges. The network may rewire edges based on their internal rules (to be defined). The removed node is added again to the network, but attached (randomly?) to another node in the network with one edge.
   5. The fitness of the network is calculated and somehow aggregated over all rounds (e.g. by summing up).
   6. **Round 1** ends.
   7. Go to **b** until maximum number of rounds is reached.
2. From the attackers and networks, we choose a certain percentage of the fittest ones. We recombine genes of the networks (cross-over) and perform random mutations.
3. The **second generation** starts their game (i.e. **1.a-1.g**). **Open issue:** how do we assign attackers to networks?
4. Go to 2. until **maximum number of generations** is reached.

**Fitness Rules**

The fitness rules still have to be designed. We aim at some network wide metric such as effective diameter . The network and the attacker have opposite fitness rules. I.e. the attacker’s fitness may be a high (i.e. maximization of ) while the network’s fitness may be a low one (i.e. minimization of ).

**Attacker strategies and genome**

The attacker selects his strategies based on a set of indicators which are calculated for each node. These indicators include e.g. centrality, degree, betweenness, closeness, clustering coefficient, eigenvector centrality, sum of communicability, and random choice and are normalized to the range [0,1]. **Open issue:** We still have to discuss in detail which measures we are going to use as computational complexity should not be underestimated.

The genes of an attacker basically consist of parameter of a weighting function for the different indicators[[1]](#footnote-0). The weight of a node with respect to indicator is determined by . The total weight of a node is then simply determined by summing over all weights for all metrics, i.e. , which is .The attacker chooses the node to be removed with probability

**Example:** Assume, a node has a degree of 3 and betweenness centrality of 2. The degree indicator is while the betweenness centrality is . The genome of the network looks like this:

|  |  |
| --- | --- |
|  |  |
| 0.5 | 2.5 |

The following values can be calculated from the given information:

(degree)

(betweenness centrality)

Let’s further assume, there are three other nodes in the network and their respective weight is 3, 1, and 8. The probability of chosen node for attack is therefore:

With a probability of 35%, node is therefore chosen to be attacked in this round.

**Comment on intercepts**: Using an intercept in the calculation of the weights does not make a lot of sense, as in the end the constant for all nodes, being a constant. It won’t have an influence on the relative weighting of the nodes. That’s a consequence of using linear relationships. If we would use non-linear transformation, using several parameters may be of use - however, we suggest to postpone this for later analysis.

**Comment on calculation of weights**: we can use efficient matrix operations to calculate the weights, if we introduce as matrix which contains the metrics associated with each node (i.e. rows=nodes,columns=metrics). We then introduce the vector which essentially represents our genome, i.e. our . Matrix multiplication , being a vector of the sum of all weights for each node.

**Network strategies**

- Networks with removed edges link up to new nodes.

- Metrics on the nodes:

Three strands of ideas:

* define metrics on edges: may be computationally very expensive
* define metrics on nodes (such as probability to be chosen in preferential attachment) and then choose two nodes randomly to connect. This may allow for a permanent loss of edges (if link between connected nodes is chosen and if this is not prevented).
* Bruno came up with a different idea: those nodes that lost their links are going to connect to new nodes (i.e. we do not have to determine 2 nodes that are connected but only one). This may be resemble the real situation where a member of an organization is much more likely to form a new link if it recently has lost a link than members which were not affected by a recent removal.

**Evolution**

The string of genes consists of and and are initially sampled from a fixed range [-a,a].

**Selection**:

**Recombination**: we recombine randomly, breaking the string of genes at a random position.

**Mutation**: we mutate randomly a random number of genes by adding to the current value of the gene a random number from the interval [-b,b]. The gene’s value is limited at a maximum and minimum.

**Side project: Roth-Erev reinforcement learning**

A side-project could be to use Roth-Erev reinforcementlearning (or any other learning algorithms)instead of genetic algorithms to “evolve” the behavior of the agents. Roth-Erev adaptive learning could be applied after each round (there are no generations) and a dynamic behavior of network and attacker can be expected. Once we have the setup for the genetic algorithms, implementing Roth-Erev learning is easy as it can rely on the fitness and strategies defined for the genetic algorithms. The need for computational resources may be lower (however, exploration of the solution space obviously depends on the amount of resources assigned).

1. We assume here, for reasons of simplicity, a linear transformation of the weights. However, any other function with any number of parameters may be used. The size of the genome basically is determined by multiplying the number of indicators with the number of parameters. [↑](#footnote-ref-0)