# Synthetic Graph Generator

***Release 0.1***

**Nicolas Lopez-Rozo*, Jorge Finke, Camilo Rocha**

**Jun 13, 2023**

# CONTENTS:

This module contains the code for generating synthetic graphs using the duplication-divergence model. This process resembles the evolution of interactions between proteins in an organism. This evolution is thought to occur in three steps:

1. **Duplication**: A protein duplicates itself. This is done by duplicating a vertex in the graph and copying all its edges.

2. **Divergence**: The two proteins diverge from each other. This is done by randomly removing edges from the duplicated vertex with a probability delta.

3. **Evolution**: The duplicated protein develops new interactions. This is done by adding new edges to the duplicated vertex with a probability alpha.

- Graph Generation

Two methods are provided for generating synthetic graphs in such a manner:

1. **duplication_divergence_graph**: This method generates a single graph using the duplication-divergence model (steps 1 and 2)

2. **extended_duplication_divergence_graph**: This method generates a single graph using the duplication-divergence model (steps 1, 2, and 3)

- Calibration methods

Besides, this module implements methods for calibrating the alpha and delta parameters given a target number of edges. This method uses a modified binary search algorithm to find the correct value for the parameter. The algorithm works as follows:

1. Given the expected number of edges, the method creates partitions of the search space [0, 1] for a parameter (alpha or delta).

2. For each partition, the method generates several graphs using the duplication-divergence model with the given parameters.

3. The method computes the average number of edges for each partition.

4. If the behavior of the average number of edges is monotonic, the method narrows the search space to the partition including the target number of edges and repeats the procedure. Otherwise, the method stops and returns the average of the limits of the partition with the expected number of edges.

Keep in mind that the calibration method finds an approximate value for the parameter, not the exact value. This is because the number of edges varies from one graph to another, even if the parameters are the same. For the extended duplication-divergence model, one of the parameters should be fixed, either alpha or delta.

- Using an existing network

Support has been added for generating a graph using the duplication-divergence model from an existing graph or the intersection of two graphs. Those methods are:

1. **synth_graph_from_network**: This method generates a graph using the duplication-divergence model from an existing graph.

2. **synth_graph_from_intersection**: This method generates a graph using the duplication-divergence model from the intersection of two graphs.

—

CHAPTER

ONE

# SYNTHETIC GRAPH GENERATION

This module contains the implementation of the duplication-divergence model, both considering only duplication and divergence, and also considering the possibility of connecting the duplicated node to other nodes not connected to it.

It is based on the following paper:

Pastor-Satorras R, Smith E, Solé RV. *Evolving protein interaction networks through gene duplication*. J Theor Biol. 2003 May 21;222(2):199-210. doi: 10.1016/s0022-5193(03)00028-6.

## 1.1 duplicationdivergence module

This module contains the implementation of the duplication-divergence model, both considering only duplication and divergence, and also considering the possibility of connecting the duplicated node to other nodes not connected to it.

### 1.1.1 Algorithmic description

It is based on the following paper:

Pastor-Satorras R, Smith E, Solé RV. *Evolving protein interaction networks through gene duplication*. J Theor Biol. 2003 May 21;222(2):199-210. doi: 10.1016/s0022-5193(03)00028-6.

The model starts from a complete graph with two nodes and then repeats this process:

1. A node is chosen at random and duplicated, creating a new node.
2. With probability delta, each of the edges of the duplicated node may be removed.
3. With probability alpha, the duplicated node may become connected to any of the nodes not connected to it before the duplication.

### 1.1.2 API Reference

duplicationdivergence.**characterize_complement**(*n*, *delta*, *alpha=0.0*)

Characterize the complement of a duplication-divergence graph by computing the number of edges, the average square clustering coefficient, the transitivity and the degree distribution. For the latter, the degree distribution is returned as a list of integers where the i-th element is the number of nodes with degree i. If alpha is different from 0 or not given, the extended duplication-divergence graph is used instead.

The complement of a graph is the graph with the same nodes and all the edges that are not in the original graph. This excludes self-loops and multiple edges.

**Parameters**

- **n** (*int*) – The number of nodes in the graph
- **delta** (*float*) – The probability of removing an edge after the duplication process
- **alpha** (*float*) – The probability of adding an edge between the duplicated node and a random node not connected to the original node

**Return (edg, squ, tri, deg)**
A tuple containing the number of edges, the average square clustering coefficient, the transitivity and the degree distribution of the graph.

duplicationdivergence.**characterize_graph**(*g*)

Characterize a graph by computing the number of edges, the average square clustering coefficient, the transitivity and the degree distribution. For the latter, the degree distribution is returned as a list of integers where the i-th element is the number of nodes with degree i.

**Parameters**
**g** (*networkx.Graph*) – The graph to characterize

**Return (edg, squ, tri, deg)**
A tuple containing the number of edges, the average square clustering coefficient, the transitivity and the degree distribution of the graph.

**Return type**
tuple

duplicationdivergence.**duplication_divergence_graph**(*n*, *delta*, *seed=None*, *track_evolution=False*, *verbose=False*)

Duplication-Divergence model where there is a possibility to connect a duplicated node to other edges not connected to it

**Parameters**

- **n** (*int*) – Number of nodes of the final network
- **delta** (*float*) – Probability of removing an edge from the duplicated node
- **seed** (*float*) – (optional) Seed for the random number generator. Default: None
- **track_evolution** (*bool*) – (Optional) Whether topological parameters are stored and return at the end of the simulation. The returned metrics are: current appended node, number of edges, average square clustering, transitivity (triangle count) and degree histogram. BEWARE: This highly increases the computational complexity of the model. Use at your own discretion.
- **verbose** (*bool*) – (optional) Whether the progress should be displayed on the screen using tqdm. Default value: False

**Return g**
The resulting graph, 0-indexed. If track_evolution is set to True, then the return value is a tuple (graph, list(metrics))

**Rtype g**
networkx.Graph

duplicationdivergence.**extended_duplication_divergence_graph**(*n*, *delta*, *alpha*, *seed=None*, *track_evolution=False*, *verbose=False*)

Duplication-Divergence model where there is a possibility to connect a duplicated node to other edges not connected to it

**Parameters**

- **n** (*int*) – Number of nodes of the final network
- **delta** (*float*) – Probability of removing an edge from the duplicated node
- **alpha** (*float*) – Probability of connecting the duplicated node with any of the other nodes not connected to it, when the duplication occurred
- **seed** (*float*) – (optional) Seed for the random number generator. Default: None
- **track_evolution** (*bool*) – (Optional) Whether topological parameters are stored and return at the end of the simulation. The returned metrics are: current appended node, number of edges, average square clustering, transitivity (triangle count) and degree histogram. BEWARE: This highly increases the computational complexity of the model. Use at your own discretion.
- **verbose** (*bool*) – (optional) Whether the progress should be displayed on the screen using tqdm. Default value: False

**Return g**
The resulting graph, 0-indexed. If track_evolution is set to True, then the return value is a tuple (graph, list(metrics))

**Rtype g**
networkx.Graph

duplicationdivergence.**timestamp**(*msg=''*, *prefix=''*)

This function prints a timestamp with a message and a prefix (both optional)

**Parameters**

- **msg** (*str*) – Message to be shown after timestamp
- **prefix** (*str*) – Message to be shown before timestamp

## 1.2 synthgraph module

This module contains the code for generating synthetic graphs using the duplication-divergence model. This process resembles the evolution of interactions between proteins in an organism. This evolution is thought to occur in three steps:

1. Duplication: A protein duplicates itself. This is done by duplicating a vertex in the graph and copying all its edges.

2. Divergence: The two proteins diverge from each other. This is done by randomly removing edges from the duplicated vertex with a probability delta.

3. Evolution: The duplicated protein develops new interactions. This is done by adding new edges to the duplicated vertex with a probability alpha

### 1.2.1 Graph generation

Two methods are provided for generating synthetic graphs in such a manner:

1. duplication_divergence_graph: This method generates a single graph using the duplication-divergence model (steps 1 and 2)

2. extended_duplication_divergence_graph: This method generates a single graph using the duplication-divergence model (steps 1, 2, and 3)

The duplication-divergence model is parameterized by the following parameters:

- n: The number of vertices in the graph
- delta: The probability of an edge being removed during divergence
- alpha: The probability of an edge being added during evolution
- seed: The seed for the random number generator

Additional optional parameters are provided for controlling verbosity:

- track_evolution: If set to True, a list of topological parameters are stored and return at the end of the simulation. The returned metrics are: current appended node, number of edges, average square clustering, transitivity (triangle count) and degree histogram.
- verbose: If set to True, the method will display the progress of the node aggregation process using the tqdm library.

### 1.2.2 Calibration methods

Besides, this module implements methods for calibrating the alpha and delta parameters given a target number of edges. This method uses a modified binary search algorithm to find the correct value for the parameter. The algorithm works as follows:

1. Given the expected number of edges, the method creates partitions of the search space [0, 1] for a parameter (alpha or delta).

2. For each partition, the method generates several graphs using the duplication-divergence model with the given parameters.

3. The method computes the average number of edges for each partition.

4. If the behaviour of the average number of edges is monotonic, the method narrows the search space to the partition including the target number of edges and repeats the procedure. Otherwise, the method stops and returns the average of the limits of the partition with the expected number of edges.

Keep in mind that the calibration method finds an approximate value for the parameter, not the exact value. This is because the number of edges varies from one graph to another, even if the parameters are the same. For the extended duplication-divergence model, one of the parameters should be fixed, either alpha or delta.

The calibration methods are parameterized by the following parameters, where applicable:

- n: The number of vertices in the graph.
- target_edges: The target number of edges.

- lo_alpha: The lower bound for the alpha parameter.
- hi_alpha: The upper bound for the alpha parameter.
- lo_delta: The lower bound for the delta parameter.
- hi_delta: The upper bound for the delta parameter.
- intervals: The number of partitions for the search space.
- repetitions: The number of graphs generated for each partition.
- seed: The seed for the random number generator.
- verbose: If set to True, the method will display the progress of the node aggregation process using the tqdm library.

### 1.2.3 Using an existing network

Support has been added for generating a graph using the duplication-divergence model from an existing graph or the intersection of two graphs. Those methods are:

1. synth_graph_from_network: This method generates a graph using the duplication-divergence model from an existing graph.
2. synth_graph_from_intersection: This method generates a graph using the duplication-divergence model from the intersection of two graphs.

The methods are parameterized by the following parameters:

- g1: The graph to be used as a template.
- g2: The second graph to be used as a template (for the intersection method only).
- alpha: The probability of an edge being added during evolution.
- lo_delta: The lower bound for the delta parameter.
- hi_delta: The upper bound for the delta parameter.
- all other optional parameters like the calibration functions: target_edges, lo_alpha, hi_alpha, intervals, repetitions, seed, verbose.
- seed: The seed for the random number generator.
- track_evolution: If set to True, a list of topological parameters are stored and return at the end of the simulation. The returned metrics are: current appended node, number of edges, average square clustering, transitivity (triangle count) and degree histogram.

### 1.2.4 API Reference

synthgraph.**calibrate_alpha**(*n*, *target_edges*, *delta*, *lo_alpha=0.0*, *hi_alpha=0.99*, *intervals=10*, *repetitions=100*, *max_iterations=5*, *threads=-1*, *verbose=False*)

This method calibrates the alpha parameter for the extended duplication-divergence model given a target number of edges. The method uses a modified binary search algorithm to find the correct value for the parameter. The algorithm calibrates the value of alpha using the following steps:

1. Given the expected number of edges, the method creates partitions of the search space [lo_alpha, hi_alpha].

2. For each partition, the method generates several graphs using the extended duplication-divergence model with the given parameters.

3. The method computes the average number of edges for each partition.

4. If the behaviour of the average number of edges is monotonic, the method narrows the search space to the partition including the target number of edges and repeats the procedure. Otherwise, the method stops and returns the average of the limits of the partition with the expected number of edges.

**Parameters**

- **n** (*int*) – The number of vertices in the graph.
- **target_edges** (*int*) – The target number of edges.
- **delta** (*float*) – The delta parameter for the extended duplication-divergence model.
- **lo_alpha** (*float*) – The lower bound for the alpha parameter.
- **hi_alpha** (*float*) – The upper bound for the alpha parameter.
- **intervals** (*int*) – The number of partitions for the search space.
- **repetitions** (*int*) – The number of graphs generated for each partition.
- **max_iterations** (*int*) – The maximum number of iterations for the binary search algorithm.
- **threads** (*int*) – The number of threads to use for parallel execution. If set to -1, the method will use all available threads.
- **verbose** – If set to True, the method will display the progress of the node aggregation process using the tqdm library.

**Return alpha**
The calibrated value for the alpha parameter.

**Rtype alpha**
float

synthgraph.**calibrate_delta**(*n*, *target_edges*, *alpha=None*, *lo_delta=0.0*, *hi_delta=0.99*, *intervals=10*, *repetitions=100*, *max_iterations=5*, *threads=-1*, *verbose=False*)

This method calibrates the delta parameter for the duplication-divergence model given a target number of edges. The method uses a modified binary search algorithm to find the correct value for the parameter. The algorithm calibrates the value of delta using the following steps:

1. Given the expected number of edges, the method creates partitions of the search space [lo_delta, hi_delta].

2. For each partition, the method generates several graphs using the duplication-divergence model with the given parameters.

3. The method computes the average number of edges for each partition.

4. If the behaviour of the average number of edges is monotonic, the method narrows the search space to the partition including the target number of edges and repeats the procedure. Otherwise, the method stops and returns the average of the limits of the partition with the expected number of edges.

**Parameters**

- **n** (*int*) – The number of vertices in the graph.
- **target_edges** (*int*) – The target number of edges.
- **alpha** (*float*) – The alpha parameter for the duplication-divergence model. If set to None, the method will use the extended duplication-divergence model. Otherwise, the method will use the duplication-divergence model.
- **lo_delta** (*float*) – The lower bound for the delta parameter.
- **hi_delta** (*float*) – The upper bound for the delta parameter.
- **intervals** (*int*) – The number of partitions for the search space.
- **repetitions** (*int*) – The number of graphs generated for each partition.
- **max_iterations** (*int*) – The maximum number of iterations for the binary search algorithm.
- **threads** (*int*) – The number of threads to use for parallel execution. If set to -1, the method will use all available threads.
- **verbose** – If set to True, the method will display the progress of the node aggregation process using the tqdm library.

**Return delta**
The calibrated value for the delta parameter.

**Rtype delta**
float

synthgraph.**synth_graph_from_intersection**(*g1*, *g2*, *alpha=None*, *lo_delta=0.0*, *hi_delta=0.99*, *intervals=10*, *repetitions=100*, *max_iterations=5*, *threads=-1*, *verbose=False*, *seed=None*, *track_evolution=False*)

This function generates a synthetic graph from the intersection of two given networks using the extended duplication-divergence model. The parameters of the model are calibrated by considering the number of vertices as n and searching for the value of delta that generates a graph with the same number of edges as the given network. The alpha parameter should be given as input to the function.

The number of nodes to use is the intersection between the two networks, g1 and g2. The number of edges to use is the maximum between the number of edges from g1 having both nodes in the node intersection and the number of edges from g2 having both nodes in the node intersection.

**Parameters**

- **g1** (*networkx.Graph*) – The first network used as base to extract information to generate the synthetic graph.
- **g2** (*networkx.Graph*) – The second network used as base to extract information to generate the synthetic graph.
- **alpha** (*float*) – The alpha parameter for the extended duplication-divergence. if set to None, the duplication-divergence model is used.
- **lo_delta** (*float*) – The lower bound for the delta parameter.

- **hi_delta** (*float*) – The upper bound for the delta parameter.
- **intervals** (*int*) – The number of partitions for the search space.
- **repetitions** (*int*) – The number of graphs generated for each partition.
- **max_iterations** (*int*) – The maximum number of iterations for the binary search algorithm.
- **threads** (*int*) – The number of threads to use for parallel execution. If set to -1, the method will use all available threads.
- **verbose** (*bool*) – If set to True, the method will display the progress of the node aggregation process using the tqdm library.
- **seed** (*int*) – The seed for the random number generator.
- **track_evolution** (*bool*) – If set to True, the method will return the evolution of the number of edges for each partition.

**Return g_synth**

The synthetic graph generated using the extended duplication-divergence model.

synthgraph.**synth_graph_from_network**(*g*, *alpha=None*, *lo_delta=0.0*, *hi_delta=0.99*, *intervals=10*, *repetitions=100*, *max_iterations=5*, *threads=-1*, *verbose=False*, *seed=None*, *track_evolution=False*)

This function generates a synthetic graph from a given network using the extended duplication-divergence model. The parameters of the model are calibrated by considering the number of vertices as n and searching for the value of delta that generates a graph with the same number of edges as the given network. The alpha parameter should be given as input to the function.

**Parameters**

- **g** (*networkx.Graph*) – The network used as base to extract information to generate the synthetic graph.
- **alpha** (*float*) – The alpha parameter for the extended duplication-divergence. if set to None, the duplication-divergence model is used.
- **lo_delta** (*float*) – The lower bound for the delta parameter.
- **hi_delta** (*float*) – The upper bound for the delta parameter.
- **intervals** (*int*) – The number of partitions for the search space.
- **repetitions** (*int*) – The number of graphs generated for each partition.
- **max_iterations** (*int*) – The maximum number of iterations for the binary search algorithm.
- **threads** (*int*) – The number of threads to use for parallel execution. If set to -1, the method will use all available threads.
- **verbose** (*bool*) – If set to True, the method will display the progress of the node aggregation process using the tqdm library.
- **seed** (*int*) – The seed for the random number generator.
- **track_evolution** (*bool*) – If set to True, the method will return the evolution of the number of edges for each partition.

**Return g_synth**

The synthetic graph generated using the extended duplication-divergence model.

**Rtype g_synth**
networkx.Graph

—

CHAPTER

TWO

# EXAMPLE

Each of the modules includes test functions if called directly using one of the following options:

```
python3 synthgraph.py
```

or:

```
python3 duplicationdivergence.py
```

For using them as part of another script, you can just load the synthgraph module, it loads all functions from the other module as well:

```
import synthgraph
```

**Note**: Include the files with *.py* extension in the folder you need to execute your code, for easy access.

# PYTHON MODULE INDEX

# INDEX