

---

**Package**

# **org.nongnu.multigraph**

## org.nongnu.multigraph Class AdjacencyMatrix

```
java.lang.Object
+-org.nongnu.multigraph.AdjacencyMatrix
```

---

**public class AdjacencyMatrix**  
extends java.lang.Object

Output an adjacency matrix of a graph, in a form suitable for MatLab/Octave.

### Constructor Summary

public	<a href="#">AdjacencyMatrix()</a>
--------	-----------------------------------

### Method Summary

static void	<a href="#">full</a> (java.io.PrintStream out, <a href="#">Graph</a> graph)
-------------	---

static void	<a href="#">parse</a> (java.io.InputStream in, <a href="#">Graph</a> graph, <a href="#">NodeLabeler</a> nl, <a href="#">EdgeLabeler</a> el)
-------------	---

static void	<a href="#">sparse</a> (java.io.PrintStream out, <a href="#">Graph</a> graph)
-------------	---

### Methods inherited from class java.lang.Object

clone, equals, finalize, getClass, hashCode, notify, notifyAll, toString, wait, wait
--

### Constructors

#### AdjacencyMatrix

**public AdjacencyMatrix()**

### Methods

#### full

**public static void full**(java.io.PrintStream out,  
[Graph](#) graph)

(continued on next page)

(continued from last page)

## sparse

```
public static void sparse(java.io.PrintStream out,  
                         Graph graph)
```

---

## parse

```
public static void parse(java.io.InputStream in,  
                         Graph graph,  
                         NodeLabeler nl,  
                         EdgeLabeler el)
```

# org.nongnu.multigraph

## Class debug

```
java.lang.Object
+-org.nongnu.multigraph.debug
```

```
public class debug
extends java.lang.Object
```

Custom debug class with support for various print statements, different debug levels and a ring-buffer to store debug message history that is only printed out if a message of at least a certain level (e.g. an error) is logged. This is intended as a convenience wrapper around java.util.logging.

### Nested Class Summary

class	<a href="#">debug.levels</a>
	debug.levels

### Field Summary

public static	<a href="#">logger</a>
public static	<a href="#">out</a>

### Constructor Summary

public	<a href="#">debug()</a>
--------	-------------------------

### Method Summary

static boolean	<a href="#">applies()</a>
static boolean	<a href="#">applies(debug.levels d)</a>
static void	<a href="#">buffersize(int s)</a> Change the size of the ring-buffer used to buffer log records when the push-level is higher than the debug level.
static java.lang.String	<a href="#">classfilter()</a>
static void	<a href="#">classfilter(java.lang.String s)</a> Set a regular expression to filter log records with.
static boolean	<a href="#">invert()</a>
static void	<a href="#">invert(boolean v)</a>
static debug.levels	<a href="#">level()</a> The current debug level.

static void	<a href="#"><u>level(debug.levels l)</u></a> Set the current debug level.
static void	<a href="#"><u>level(java.lang.String l)</u></a> Set the current debug level.
static java.lang.String	<a href="#"><u>methodfilter()</u></a>
static void	<a href="#"><u>methodfilter(java.lang.String s)</u></a> Set a regular expression to filter log records with.
static java.lang.String	<a href="#"><u>msgfilter()</u></a>
static void	<a href="#"><u>msgfilter(java.lang.String s)</u></a> Set a regular expression to filter log records with.
static void	<a href="#"><u>printf(debug.levels d, java.lang.String s)</u></a>
static void	<a href="#"><u>printf(debug.levels d, java.lang.String format, java.lang.Object[] args)</u></a>
static void	<a href="#"><u>printf(java.lang.String s)</u></a>
static void	<a href="#"><u>printf(java.lang.String format, java.lang.Object[] args)</u></a>
static void	<a href="#"><u>println(debug.levels d, java.lang.String s)</u></a>
static void	<a href="#"><u>println(java.lang.String s)</u></a>
static <a href="#"><u>debug.levels</u></a>	<a href="#"><u>pushlevel()</u></a> The current push-level.
static void	<a href="#"><u>pushlevel(debug.levels l)</u></a> Set the current push-level.
static void	<a href="#"><u>pushlevel(java.lang.String l)</u></a> Set the current push-level.

**Methods inherited from class** java.lang.Object

clone, equals, finalize, getClass, hashCode, notify, notifyAll, toString, wait, wait

## Fields

### out

public static java.io.PrintStream **out**

(continued from last page)

## logger

```
public static java.util.logging.Logger logger
```

## Constructors

### debug

```
public debug()
```

## Methods

### classfilter

```
public static java.lang.String classfilter()
```

### classfilter

```
public static void classfilter(java.lang.String s)
```

Set a regular expression to filter log records with. Only messages emitted from classes with names matching the regex will be logged.

**Parameters:**

s - Regular expression acceptable to Pattern, or the empty string "" to unset.

### msgfilter

```
public static void msgfilter(java.lang.String s)
```

Set a regular expression to filter log records with. Only messages whose content matches the regex will be logged.

**Parameters:**

s - Regular expression acceptable to Pattern, or the empty string "" to unset.

### msgfilter

```
public static java.lang.String msgfilter()
```

### methodfilter

```
public static java.lang.String methodfilter()
```

### methodfilter

```
public static void methodfilter(java.lang.String s)
```

(continued from last page)

Set a regular expression to filter log records with. Only messages emitted from methods with names matching the regex will be logged.

**Parameters:**

s - Regular expression acceptable to Pattern, or the empty string "" to unset.

---

**invert**

```
public static boolean invert()
```

---

**invert**

```
public static void invert(boolean v)
```

---

**level**

```
public static debug.levels level()
```

The current debug level. Messages below this level will not be captured.

**Returns:**

Current debug.level.

**See Also:**

level.

---

**level**

```
public static void level(debug.levels l)
```

Set the current debug level. Messages below this level will not be captured.

**Parameters:**

l - The debug.level to set.

**See Also:**

level.

---

**level**

```
public static void level(java.lang.String l)
```

Set the current debug level. Messages below this level will not be captured.

**Parameters:**

l - String representation of debug.level to set.

**See Also:**

level.

---

**pushlevel**

```
public static debug.levels pushlevel()
```

(continued from last page)

The current push-level. Debug messages that have been captured are buffered in a ring-buffer in memory and not pushed out until a message of at least this level is seen. This defaults to the current debug level, and is reset accordingly whenever the debug level is reset - the push-level must be set after the debug level if a different level is desired. @see #level(levels).

E.g. Setting the debug-level to "debug", but the push-level to "error" would print out the last X lines of "debug" output, but only if a "error" level message was received.

**Returns:**

---

**pushlevel**

```
public static void pushlevel(debug.levels l)
```

Set the current push-level. @see #pushlevel and @see #level.

**Parameters:**

l - the [level](#) to set the push-level to.

---

**pushlevel**

```
public static void pushlevel(java.lang.String l)
```

Set the current push-level. @see #pushlevel and @see #level.

**Parameters:**

l - String form of the [level](#) to set the push-level to.

---

**buffersize**

```
public static void buffersize(int s)
```

Change the size of the ring-buffer used to buffer log records when the push-level is higher than the debug level. This defaults to 4096000 (i.e. quite a lot). Setting this parameter destroys any existing log messages.

**Parameters:**

s

---

**applies**

```
public static boolean applies(debug.levels d)
```

---

**applies**

```
public static boolean applies()
```

---

**printf**

```
public static void printf(debug.levels d,  
                      java.lang.String s)
```

---

(continued from last page)

**printf**

```
public static void printf(java.lang.String s)
```

---

**printf**

```
public static void printf(debug.levels d,  
                      java.lang.String format,  
                      java.lang.Object[] args)
```

---

**printf**

```
public static void printf(java.lang.String format,  
                      java.lang.Object[] args)
```

---

**println**

```
public static void println(debug.levels d,  
                        java.lang.String s)
```

---

**println**

```
public static void println(java.lang.String s)
```

---

## org.nongnu.multigraph Class debug.levels

```
java.lang.Object
  +-- java.lang.Enum
    +-- org.nongnu.multigraph.debug.levels
```

### All Implemented Interfaces:

java.io.Serializable, java.lang.Comparable

public static final class **debug.levels**

extends java.lang.Enum

### Field Summary

public static final	<a href="#">DEBUG</a>
public static final	<a href="#">ERROR</a>
public static final	<a href="#">INFO</a>
public static final	<a href="#">NONE</a>
public static final	<a href="#">WARNING</a>

### Method Summary

static <a href="#">debug.levels</a>	<a href="#">valueOf(java.lang.String name)</a>
static <a href="#">debug.levels[]</a>	<a href="#">values()</a>

### Methods inherited from class java.lang.Enum

clone, compareTo, equals, finalize, getDeclaringClass, hashCode, name, ordinal, toString, valueOf

### Methods inherited from class java.lang.Object

clone, equals, finalize, getClass, hashCode, notify, notifyAll, toString, wait, wait

### Methods inherited from interface java.lang.Comparable

compareTo

### Fields

(continued from last page)

## NONE

```
public static final org.nongnu.multigraph.debug.levels NONE
```

---

## ERROR

```
public static final org.nongnu.multigraph.debug.levels ERROR
```

---

## WARNING

```
public static final org.nongnu.multigraph.debug.levels WARNING
```

---

## INFO

```
public static final org.nongnu.multigraph.debug.levels INFO
```

---

## DEBUG

```
public static final org.nongnu.multigraph.debug.levels DEBUG
```

---

## Methods

### values

```
public static debug.levels[] values()
```

---

### valueOf

```
public static debug.levels valueOf(java.lang.String name)
```

## org.nongnu.multigraph Class Edge

```
java.lang.Object
+-org.nongnu.multigraph.Edge
```

---

```
public class Edge
extends java.lang.Object
```

An edge, identified immutably by , of some mutable weight.

**Parameters:**

N - The type of the Node's in the graph., E - The type of the Edges in the graph

### Method Summary

java.lang.Object	<a href="#"><u>from()</u></a>
java.lang.Object	<a href="#"><u>label()</u></a>
java.lang.Object	<a href="#"><u>to()</u></a>
java.lang.String	<a href="#"><u>toString()</u></a>
int	<a href="#"><u>weight()</u></a>

### Methods inherited from class java.lang.Object

clone, equals, finalize, getClass, hashCode, notify, notifyAll, toString, wait, wait
--

### Methods

#### label

```
public java.lang.Object label()
```

---

#### from

```
public java.lang.Object from()
```

---

#### to

```
public java.lang.Object to()
```

---

## **weight**

```
public int weight()
```

---

## **toString**

```
public java.lang.String toString()
```

## org.nongnu.multigraph Interface EdgeLabeler

---

public interface **EdgeLabeler**  
extends

Callback interface to forward the labelling of each new edge back to the user.

**Parameters:**

N - The type of the Nodes in the graph, E - The type of the Edges in the graph

---

### Method Summary

abstract java.lang.Object	<a href="#">getLabel</a> (java.lang.Object from, java.lang.Object to)
------------------------------	---

### Methods

#### getLabel

```
public abstract java.lang.Object getLabel(java.lang.Object from,  
                                     java.lang.Object to)
```

# org.nongnu.multigraph Interface Graph

## All Known Implementing Classes:

[MultiDiGraph](#), [SyncGraph](#), [PartitionGraph](#)

public interface **Graph**

extends java.util.Set

General Graph interface, for graphs of N-typed nodes, each having 0 or more edges to other nodes, labeled L.

Edges may optionally be weighted, otherwise they are assigned a default weight of 1. Weights of edges may be revised.

This interface does not specify how many edges are allowed between nodes and does specify whether a node may have an edge to itself, or not. A graph which explicitly allows both these cases is called a "multi-graph", a graph which does not allow either condition is called a "simple-graph".

This interface does not restrict whether edges are directed or not. Where edges are directed, then the presence of an edge out from one node to another does not imply that the other node has a corresponding edge back. Where edges are undirected, then the presence of an edge at one node must imply the presence of a similar edge at the other node.

This interface does not allow for "hyper-graphs", where 1 edge may connect to multiple nodes, however hyper-graphs can be supported by using a node to act as hyper-edge.

### Parameters:

N - The type of the Nodes in the graph, E - The type of the Edges in the graph

## Method Summary

abstract boolean	<a href="#"><code>add</code>(java.lang.Object node)</a> Add the given node to the graph, sans edges
abstract void	<a href="#"><code>addObserver</code>(java.util.Observer o)</a>
abstract float	<a href="#"><code>avg_nodal_degree()</code></a>
abstract void	<a href="#"><code>clear_all_edges()</code></a> Clear all edges from the graph.
abstract int	<a href="#"><code>countObservers()</code></a>
abstract void	<a href="#"><code>deleteObserver</code>(java.util.Observer o)</a>
abstract void	<a href="#"><code>deleteObservers()</code></a>
abstract int	<a href="#"><code>edge_outdegree</code>(java.lang.Object node)</a> The out-degree for a give node.
abstract <a href="#">Edge</a>	<a href="#"><code>edge</code>(java.lang.Object from, java.lang.Object to)</a> Find the first edge from a node to another node.
abstract <a href="#">Edge</a>	<a href="#"><code>edge</code>(java.lang.Object from, java.lang.Object to, java.lang.Object label)</a> Find the edge with the given label, from the one node to another node, if it exists.

abstract java.util.Set	<a href="#"><u>edges</u></a> (java.lang.Object from)
abstract java.util.Collection	<a href="#"><u>edges</u></a> (java.lang.Object from, java.lang.Object to) Find the edges going <i>from</i> one node <i>to</i> another node.
abstract boolean	<a href="#"><u>hasChanged</u></a> ()
abstract boolean	<a href="#"><u>is_directed</u></a> ()
abstract boolean	<a href="#"><u>is_linked</u></a> (java.lang.Object from, java.lang.Object to) Determine if there is a link from one node to another.
abstract boolean	<a href="#"><u>is_simple</u></a> ()
abstract long	<a href="#"><u>link_count</u></a> ()
abstract int	<a href="#"><u>max_nodal_degree</u></a> ()
abstract int	<a href="#"><u>nodal_outdegree</u></a> (java.lang.Object node)
abstract void	<a href="#"><u>notifyObservers</u></a> ()
abstract void	<a href="#"><u>notifyObservers</u></a> (java.lang.Object arg)
abstract void	<a href="#"><u>plugObservable</u></a> () "Plug" delivery of Observable events to observers, so that any such events are instead queued up internally, and potentially coalesced, rather than delivered to Observers.
abstract java.lang.Iterable	<a href="#"><u>random_edge_iterable</u></a> (java.lang.Object n) Provide a random-access Iterable over the <Edge<N,E>>-edges from the given node.
abstract java.lang.Iterable	<a href="#"><u>random_node_iterable</u></a> () Provide a random-access Iterable over the <N>-nodes in the graph.
abstract boolean	<a href="#"><u>remove</u></a> (java.lang.Object from, java.lang.Object to) Remove all edges that go from one node to another
abstract boolean	<a href="#"><u>remove</u></a> (java.lang.Object from, java.lang.Object to, java.lang.Object label) Remove 1 specific edge, given by the label, from between two nodes.
abstract void	<a href="#"><u>set</u></a> (java.lang.Object from, java.lang.Object to, java.lang.Object label) Set an edge from one node to another with the given label.
abstract void	<a href="#"><u>set</u></a> (java.lang.Object from, java.lang.Object to, java.lang.Object label, int weight) Set an edge from one node to another with the given label and weight.
abstract java.util.Set	<a href="#"><u>successors</u></a> (java.lang.Object from)
abstract void	<a href="#"><u>unplugObservable</u></a> () Unplug the delivery of Observable events.

**Methods inherited from interface** java.util.Set

---

```
add, addAll, clear, contains, containsAll, equals, hashCode, isEmpty, iterator,
remove, removeAll, retainAll, size, spliterator, toArray, toArray
```

#### Methods inherited from interface java.util.Collection

```
add, addAll, clear, contains, containsAll, equals, hashCode, isEmpty, iterator,
parallelStream, remove, removeAll, removeIf, retainAll, size, spliterator, stream,
toArray, toArray
```

#### Methods inherited from interface java.lang.Iterable

```
forEach, iterator, spliterator
```

## Methods

### **is\_directed**

```
public abstract boolean is_directed()
```

#### Returns:

Whether the graph is directed or not, i.e. whether edges have a direction.

### **is\_simple**

```
public abstract boolean is_simple()
```

#### Returns:

Whether the graph is simple or not. A simple graph allows only single edges between nodes.

### **set**

```
public abstract void set(java.lang.Object from,
                        java.lang.Object to,
                        java.lang.Object label)
```

Set an edge from one node to another with the given label. Weight defaults to 1. Nodes are automatically added to graph, as needs be. If a given edge for }already exists, then the weight is updated.

#### Parameters:

- from - Node from which the edge originates.
- to - Node to which the edges goes.
- label - The user's label for this edge.

### **set**

```
public abstract void set(java.lang.Object from,
                        java.lang.Object to,
                        java.lang.Object label,
                        int weight)
```

Set an edge from one node to another with the given label and weight.

(continued from last page)

**Parameters:**

weight - a user-specified metric or weight for the edge

**See Also:**[set\(N, N, E\)](#)

---

**add**

```
public abstract boolean add(java.lang.Object node)
```

Add the given node to the graph, sans edges

---

**remove**

```
public abstract boolean remove(java.lang.Object from,
                               java.lang.Object to,
                               java.lang.Object label)
```

Remove 1 specific edge, given by the label, from between two nodes.

**Returns:**

Whether an edge was removed

---

**remove**

```
public abstract boolean remove(java.lang.Object from,
                               java.lang.Object to)
```

Remove all edges that go from one node to another

**Returns:**

Whether an edge was removed

**See Also:**[set\(N, N, E\)](#)[add\(N\)](#)

---

**clear\_all\_edges**

```
public abstract void clear_all_edges()
```

Clear all edges from the graph. This can NOT be done by iterating over the sets of edges from each node, as that will raise a concurrent modification exception.

---

**edge\_outdegree**

```
public abstract int edge_outdegree(java.lang.Object node)
```

The out-degree for a give node. I.e. the number of edges leaving the node.

---

**nodal\_outdegree**

```
public abstract int nodal_outdegree(java.lang.Object node)
```

**Parameters:**

node

(continued from last page)

**Returns:**

The number of distinct nodes to which this node has edges.

---

**avg\_nodal\_degree**

```
public abstract float avg_nodal_degree()
```

**Returns:**

The average nodal out-degree.

---

**link\_count**

```
public abstract long link_count()
```

**Returns:**

The number of edges in the graph

---

**max\_nodal\_degree**

```
public abstract int max_nodal_degree()
```

**Returns:**

The maximum degree of any node in the graph

---

**successors**

```
public abstract java.util.Set successors(java.lang.Object from)
```

**Parameters:**

node - The given node, which is to be queried.

**Returns:**

The set of nodes that succeed the given node. I.e. those nodes to which the given node has an edge. The returned set is read-only and may not be modified. The returned set will be null if the node does not exist, and will be the empty set if the node exists but has no successors.

---

**edges**

```
public abstract java.util.Set edges(java.lang.Object from)
```

**Parameters:**

node - The given node, which is to be queried.

**Returns:**

The set of edges that go out from this node. The returned set is read-only and may not be modified. The set will be null if the node does not exist, and an empty set if the node exists but has no edges.

(continued from last page)

## edges

```
public abstract java.util.Collection edges(java.lang.Object from,  
                                         java.lang.Object to)
```

Find the edges going *from* one node *to* another node.

**Parameters:**

*from* - Which node we want to query edges from.  
*to* - The node to which we're looking for an edge.

**Returns:**

An immutable Collection of edges going from

**See Also:**

[edges \(N\)](#)

---

## edge

```
public abstract Edge edge(java.lang.Object from,  
                         java.lang.Object to)
```

Find the first edge from a node to another node.

**Parameters:**

*from* - Which node we want to query edges from.  
*to* - The node to which we're looking for an edge.

**Returns:**

An edge going from *from->to* if any exist, or null otherwise.

---

## is\_linked

```
public abstract boolean is_linked(java.lang.Object from,  
                                 java.lang.Object to)
```

Determine if there is a link from one node to another.

**Returns:**

True if an edge exists from -> to

---

## edge

```
public abstract Edge edge(java.lang.Object from,  
                         java.lang.Object to,  
                         java.lang.Object label)
```

Find the edge with the given label, from the one node to another node, if it exists.

**Parameters:**

*from* - Which node we want to query edges from.  
*to* - The node to which we're looking for an edge.  
*label* - The label of the edge we're interested in.

**Returns:**

The edge going from *from->to* specified by the given label, if it exists, or null otherwise.

(continued from last page)

## **random\_node\_iterable**

```
public abstract java.lang.Iterable random_node_iterable()
```

Provide a random-access Iterable over the <N>-nodes in the graph.

---

## **random\_edge\_iterable**

```
public abstract java.lang.Iterable random_edge_iterable(java.lang.Object n)
```

Provide a random-access Iterable over the <Edge<N,E>>-edges from the given node.

---

## **addObserver**

```
public abstract void addObserver(java.util.Observer o)
```

---

### **See Also:**

`java.util.Observable`

---

## **countObservers**

```
public abstract int countObservers()
```

---

### **See Also:**

`java.util.Observable`

---

## **deleteObserver**

```
public abstract void deleteObserver(java.util.Observer o)
```

---

### **See Also:**

`java.util.Observable`

---

## **deleteObservers**

```
public abstract void deleteObservers()
```

---

### **See Also:**

`java.util.Observable`

---

## **hasChanged**

```
public abstract boolean hasChanged()
```

---

### **See Also:**

`java.util.Observable`

(continued from last page)

## notifyObservers

```
public abstract void notifyObservers()
```

### See Also:

`java.util.Observable`

---

## notifyObservers

```
public abstract void notifyObservers(java.lang.Object arg)
```

### See Also:

`java.util.Observable`

---

## plugObservable

```
public abstract void plugObservable()
```

"Plug" delivery of Observable events to observers, so that any such events are instead queued up internally, and potentially coalesced, rather than delivered to Observers. This potentially allows bulk updates to be made to the graph more efficiently.

---

## unplugObservable

```
public abstract void unplugObservable()
```

Unplug the delivery of Observable events. Stored events will be delivered, though potentially coalesced, so that multiple events for the same object are collapsed into one. Further, a general "notifyObservers" event will coalesce with and subsume \*all\* events for specific objects.

# org.nongnu.multigraph Class MultiDiGraph

```
java.lang.Object
  +-java.util.Observable
    +-org.nongnu.multigraph.MultiDiGraph
```

## All Implemented Interfaces:

[Graph](#)

## Direct Known Subclasses:

[MultiGraph](#), [SimpleDiGraph](#)

```
public class MultiDiGraph
extends java.util.Observable
implements Graph
```

A multi-edge, directed graph implementation of the [Graph](#) interface.

This implementation allows for multiple, directed edges between any nodes, including between the same node.

XXX: As SimpleDiGraph inherits from this, this class probably should not be public. We don't really want different restrictions of the Graph to be type-compatible.

### Parameters:

N - The type of the Nodes in the graph, E - The type of the Edges in the graph

## Constructor Summary

public	<a href="#">MultiDiGraph()</a>
--------	--------------------------------

## Method Summary

boolean	<a href="#">remove(java.lang.Object from, java.lang.Object to, java.lang.Object label)</a> The core, central edge removal method.
void	<a href="#">set(java.lang.Object from, java.lang.Object to, int weight, java.lang.Object label)</a> The core, central set method.
boolean	<a href="#">add(java.lang.Object node)</a>
boolean	<a href="#">addAll(java.util.Collection c)</a>
float	<a href="#">avg_nodal_degree()</a>
void	<a href="#">clear_all_edges()</a>
void	<a href="#">clear()</a>
boolean	<a href="#">contains(java.lang.Object o)</a>

boolean	<a href="#"><u>containsAll</u></a> (java.util.Collection c)
int	<a href="#"><u>edge_outdegree</u></a> (java.lang.Object node)
<a href="#"><u>Edge</u></a>	<a href="#"><u>edge</u></a> (java.lang.Object from, java.lang.Object to)
<a href="#"><u>Edge</u></a>	<a href="#"><u>edge</u></a> (java.lang.Object from, java.lang.Object to, java.lang.Object label)
java.util.Set	<a href="#"><u>edges</u></a> (java.lang.Object from)
java.util.Collection	<a href="#"><u>edges</u></a> (java.lang.Object from, java.lang.Object to)
boolean	<a href="#"><u>equals</u></a> (java.lang.Object o)
int	<a href="#"><u>hashCode</u></a> ()
boolean	<a href="#"><u>is_directed</u></a> ()
boolean	<a href="#"><u>is_linked</u></a> (java.lang.Object from, java.lang.Object to)
boolean	<a href="#"><u>is_simple</u></a> ()
boolean	<a href="#"><u>isEmpty</u></a> ()
java.util.Iterator	<a href="#"><u>iterator</u></a> ()
long	<a href="#"><u>link_count</u></a> ()
int	<a href="#"><u>max_nodal_degree</u></a> ()
int	<a href="#"><u>nodal_outdegree</u></a> (java.lang.Object node)
void	<a href="#"><u>notifyObservers</u></a> ()
void	<a href="#"><u>notifyObservers</u></a> (java.lang.Object arg)
void	<a href="#"><u>plugObservable</u></a> ()
java.lang.Iterable	<a href="#"><u>random_edge_iterable</u></a> (java.lang.Object n)
java.lang.Iterable	<a href="#"><u>random_node_iterable</u></a> ()
boolean	<a href="#"><u>remove</u></a> (java.lang.Object from, java.lang.Object to)
boolean	<a href="#"><u>remove</u></a> (java.lang.Object from, java.lang.Object to, java.lang.Object label)

boolean	<a href="#">remove</a> (java.lang.Object o)
boolean	<a href="#">removeAll</a> (java.util.Collection c)
boolean	<a href="#">retainAll</a> (java.util.Collection c)
void	<a href="#">set</a> (java.lang.Object from, java.lang.Object to, java.lang.Object label)
void	<a href="#">set</a> (java.lang.Object from, java.lang.Object to, java.lang.Object label, int weight)
int	<a href="#">size()</a>
java.util.Set	<a href="#">successors</a> (java.lang.Object node)
java.lang.Object[]	<a href="#">toArray()</a>
java.lang.Object[]	<a href="#">toArray</a> (java.lang.Object[] a)
java.lang.String	<a href="#">toString()</a>
void	<a href="#">unplugObservable()</a>

**Methods inherited from class java.util.Observable**

addObserver, clearChanged, countObservers, deleteObserver, deleteObservers, hasChanged, notifyObservers, notifyObservers, setChanged

**Methods inherited from class java.lang.Object**

clone, equals, finalize, getClass, hashCode, notify, notifyAll, toString, wait, wait

**Methods inherited from interface org.nongnu.multigraph.Graph**

[add](#), [addObserver](#), [avg\\_nodal\\_degree](#), [clear\\_all\\_edges](#), [countObservers](#), [deleteObserver](#), [deleteObservers](#), [edge\\_outdegree](#), [edge](#), [edges](#), [edges](#), [hasChanged](#), [is\\_directed](#), [is\\_linked](#), [is\\_simple](#), [link\\_count](#), [max\\_nodal\\_degree](#), [nodal\\_outdegree](#), [notifyObservers](#), [notifyObservers](#), [plugObservable](#), [random\\_edge\\_iterable](#), [random\\_node\\_iterable](#), [remove](#), [remove](#), [set](#), [successors](#), [unplugObservable](#)

**Methods inherited from interface java.util.Set**

add, addAll, clear, contains, containsAll, equals, hashCode, isEmpty, iterator, remove, removeAll, retainAll, size, spliterator, toArray, toArray

**Methods inherited from interface java.util.Collection**

add, addAll, clear, contains, containsAll, equals, hashCode, isEmpty, iterator, parallelStream, remove, removeAll, removeIf, retainAll, size, spliterator, stream, toArray

**Methods inherited from interface java.lang.Iterable**

```
forEach, iterator, spliterator
```

## Constructors

### MultiDiGraph

```
public MultiDiGraph()
```

## Methods

### \_set

```
protected void _set(java.lang.Object from,
                   java.lang.Object to,
                   int weight,
                   java.lang.Object label)
```

The core, central set method. All other set methods are filters for this method, meant to check invariants, apply various requirements. etc. Exported to package so that other types of graph may be subclassed from this

#### Parameters:

- from - The N-typed node from which to set the new edge
- to - The N-typed node to which the edge should be set
- weight - The weight of the node, unweighted edges should just be set to 1.
- label - The E-typed edge label object for the graph edge.

---

### set

```
public void set(java.lang.Object from,
                java.lang.Object to,
                java.lang.Object label)
```

---

### set

```
public void set(java.lang.Object from,
                java.lang.Object to,
                java.lang.Object label,
                int weight)
```

---

### \_remove

```
protected boolean _remove(java.lang.Object from,
                        java.lang.Object to,
                        java.lang.Object label)
```

The core, central edge removal method. All other remove methods are filters for this method, meant to check invariants, apply various requirements. etc. Exported to package so that other types of graph may be subclassed from this

#### Parameters:

- from - The N-typed node from which the edge is to be removed
- to - The N-typed node to which the edge is to be removed

(continued from last page)

**label** - The E-typed edge label object for the edge which is to be removed. If specified, only edges matching the label will be removed. If not specified, all edges will be removed.

---

## remove

```
public boolean remove(java.lang.Object from,  
                     java.lang.Object to,  
                     java.lang.Object label)
```

---

## remove

```
public boolean remove(java.lang.Object from,  
                     java.lang.Object to)
```

---

## edges

```
public java.util.Set edges(java.lang.Object from)
```

---

## edges

```
public java.util.Collection edges(java.lang.Object from,  
                                  java.lang.Object to)
```

---

## edge

```
public Edge edge(java.lang.Object from,  
                  java.lang.Object to)
```

---

## edge

```
public Edge edge(java.lang.Object from,  
                  java.lang.Object to,  
                  java.lang.Object label)
```

---

## successors

```
public java.util.Set successors(java.lang.Object node)
```

---

## edge\_outdegree

```
public int edge_outdegree(java.lang.Object node)
```

(continued from last page)

**nodal\_outdegree**

```
public int nodal_outdegree(java.lang.Object node)
```

---

**avg\_nodal\_degree**

```
public float avg_nodal_degree()
```

---

**link\_count**

```
public long link_count()
```

---

**max\_nodal\_degree**

```
public int max_nodal_degree()
```

---

**toString**

```
public java.lang.String toString()
```

---

**random\_node\_iterable**

```
public java.lang.Iterable random_node_iterable()
```

---

**random\_edge\_iterable**

```
public java.lang.Iterable random_edge_iterable(java.lang.Object n)
```

---

**add**

```
public boolean add(java.lang.Object node)
```

---

**addAll**

```
public boolean addAll(java.util.Collection c)
```

---

**clear**

```
public void clear()
```

(continued from last page)

---

**contains**

```
public boolean contains(java.lang.Object o)
```

---

**containsAll**

```
public boolean containsAll(java.util.Collection c)
```

---

**equals**

```
public boolean equals(java.lang.Object o)
```

---

**hashCode**

```
public int hashCode()
```

---

**isEmpty**

```
public boolean isEmpty()
```

---

**size**

```
public int size()
```

---

**toArray**

```
public java.lang.Object[] toArray()
```

---

**toArray**

```
public java.lang.Object[] toArray(java.lang.Object[] a)
```

---

**iterator**

```
public java.util.Iterator iterator()
```

---

(continued from last page)

**clear\_all\_edges**

```
public void clear_all_edges()
```

**remove**

```
public boolean remove(java.lang.Object o)
```

**removeAll**

```
public boolean removeAll(java.util.Collection c)
```

**retainAll**

```
public boolean retainAll(java.util.Collection c)
```

**plugObservable**

```
public void plugObservable()
```

**unplugObservable**

```
public void unplugObservable()
```

**notifyObservers**

```
public void notifyObservers()
```

**notifyObservers**

```
public void notifyObservers(java.lang.Object arg)
```

**is\_directed**

```
public boolean is_directed()
```

**is\_simple**

```
public boolean is_simple()
```

(continued from last page)

---

## **is\_linked**

```
public boolean is_linked(java.lang.Object from,  
                      java.lang.Object to)
```

# org.nongnu.multigraph

## Class MultiGraph

```
java.lang.Object
  +-java.util.Observable
    +-org.nongnu.multigraph.MultiDiGraph
      +-org.nongnu.multigraph.MultiGraph
```

All Implemented Interfaces:  
[Graph](#)

**public class MultiGraph**  
**extends MultiDiGraph**

A multi-edge, undirected edge graph implementation.

**Parameters:**

N - The type of the Nodes in the graph, E - The type of the Edges in the graph

**See Also:**

[MultiDiGraph](#)

### Constructor Summary

public	<a href="#">MultiGraph()</a>
--------	------------------------------

### Method Summary

boolean	<a href="#">_remove(java.lang.Object from, java.lang.Object to, java.lang.Object label)</a>
void	<a href="#">_set(java.lang.Object from, java.lang.Object to, int weight, java.lang.Object label)</a>
boolean	<a href="#">is_directed()</a>
boolean	<a href="#">remove(java.lang.Object o)</a>

### Methods inherited from class [org.nongnu.multigraph.MultiDiGraph](#)

[\\_remove](#), [\\_set](#), [add](#), [addAll](#), [avg\\_nodal\\_degree](#), [clear\\_all\\_edges](#), [clear](#), [contains](#), [containsAll](#), [edge\\_outdegree](#), [edge](#), [edge](#), [edges](#), [edges](#), [equals](#), [hashCode](#), [is\\_directed](#), [is\\_linked](#), [is\\_simple](#), [isEmpty](#), [iterator](#), [link\\_count](#), [max\\_nodal\\_degree](#), [nodal\\_outdegree](#), [notifyObservers](#), [notifyObservers](#), [plugObservable](#), [random\\_edge\\_iterable](#), [random\\_node\\_iterable](#), [remove](#), [remove](#), [remove](#), [removeAll](#), [retainAll](#), [set](#), [set](#), [size](#), [successors](#), [toArray](#), [toString](#), [unplugObservable](#)

### Methods inherited from class [java.util.Observable](#)

[addObserver](#), [clearChanged](#), [countObservers](#), [deleteObserver](#), [deleteObservers](#), [hasChanged](#), [notifyObservers](#), [notifyObservers](#), [setChanged](#)

### Methods inherited from class [java.lang.Object](#)

```
clone, equals, finalize, getClass, hashCode, notify, notifyAll, toString, wait, wait,
wait
```

#### Methods inherited from interface [org.nongnu.multigraph.Graph](#)

```
add, addObserver, avg_nodal_degree, clear_all_edges, countObservers, deleteObserver,
deleteObservers, edge_outdegree, edge, edges, edges, hasChanged, is_directed,
is_linked, is_simple, link_count, max_nodal_degree, nodal_outdegree, notifyObservers,
notifyObservers, plugObservable, random_edge_iterable, random_node_iterable, remove,
remove, set, set, successors, unplugObservable
```

#### Methods inherited from interface [java.util.Set](#)

```
add, addAll, clear, contains, containsAll, equals, hashCode, isEmpty, iterator,
remove, removeAll, retainAll, size, spliterator, toArray, toArray
```

#### Methods inherited from interface [java.util.Collection](#)

```
add, addAll, clear, contains, containsAll, equals, hashCode, isEmpty, iterator,
parallelStream, remove, removeAll, removeIf, retainAll, size, spliterator, stream,
toArray, toArray
```

#### Methods inherited from interface [java.lang.Iterable](#)

```
forEach, iterator, spliterator
```

## Constructors

### MultiGraph

```
public MultiGraph()
```

## Methods

### remove

```
protected boolean remove(java.lang.Object from,
                         java.lang.Object to,
                         java.lang.Object label)
```

The core, central edge removal method. All other remove methods are filters for this method, meant to check invariants, apply various requirements. etc. Exported to package so that other types of graph may be subclassed from this

### set

```
protected void set(java.lang.Object from,
                    java.lang.Object to,
                    int weight,
                    java.lang.Object label)
```

The core, central set method. All other set methods are filters for this method, meant to check invariants, apply various requirements. etc. Exported to package so that other types of graph may be subclassed from this

(continued from last page)

## **remove**

```
public boolean remove(java.lang.Object o)
```

---

## **is\_directed**

```
public boolean is_directed()
```

## org.nongnu.multigraph Interface NodeLabeler

---

public interface **NodeLabeler**  
extends

Callback interface to forward the labelling of new nodes back to the user.

**Parameters:**

N - The type of the Nodes in the graph, E - The type of the Edges in the graph

---

### Method Summary

abstract java.lang.Object	<a href="#">getNode(java.lang.String n)</a>
------------------------------	---

### Methods

#### **getNode**

public abstract java.lang.Object **getNode**(java.lang.String n)

## org.nongnu.multigraph Class PartitionGraph

```
java.lang.Object
+-org.nongnu.multigraph.PartitionGraph
```

### All Implemented Interfaces:

java.util.Observer, [Graph](#)

```
public class PartitionGraph
extends java.lang.Object
implements Graph, java.util.Observer
```

Partition the nodes of the supplied graph. Assignment of nodes to partitions is controlled by a user supplied callback class, with the PartitionCallbacks interface. Updates to the underlying graph automatically keep the partition updated.

### Nested Class Summary

class	<a href="#">PartitionGraph.PartitionCallbacks</a> PartitionGraph.PartitionCallbacks
-------	--

### Constructor Summary

public	<a href="#">PartitionGraph(PartitionGraph.PartitionCallbacks cb)</a>
--------	--

### Method Summary

boolean	<a href="#">add(java.lang.Object node)</a>
boolean	<a href="#">addAll(java.util.Collection clctn)</a>
void	<a href="#">addObserver(java.util.Observer o)</a>
float	<a href="#">avg_nodal_degree()</a>
void	<a href="#">clear_all_edges()</a>
void	<a href="#">clear()</a>
boolean	<a href="#">contains(java.lang.Object o)</a>
boolean	<a href="#">containsAll(java.util.Collection clctn)</a>
int	<a href="#">countObservers()</a>
void	<a href="#">deleteObserver(java.util.Observer o)</a>
void	<a href="#">deleteObservers()</a>

int	<a href="#"><u>edge_outdegree</u>(java.lang.Object node)</a>
<a href="#"><u>Edge</u></a>	<a href="#"><u>edge</u>(java.lang.Object from, java.lang.Object to)</a>
<a href="#"><u>Edge</u></a>	<a href="#"><u>edge</u>(java.lang.Object from, java.lang.Object to, java.lang.Object label)</a>
java.util.Set	<a href="#"><u>edges</u>(java.lang.Object from)</a>
java.util.Collection	<a href="#"><u>edges</u>(java.lang.Object from, java.lang.Object to)</a>
boolean	<a href="#"><u>hasChanged</u>()</a>
boolean	<a href="#"><u>is_directed</u>()</a>
boolean	<a href="#"><u>is_linked</u>(java.lang.Object from, java.lang.Object to)</a>
boolean	<a href="#"><u>is_simple</u>()</a>
boolean	<a href="#"><u>isEmpty</u>()</a>
java.util.Iterator	<a href="#"><u>iterator</u>()</a>
long	<a href="#"><u>link_count</u>()</a>
int	<a href="#"><u>max_nodal_degree</u>()</a>
int	<a href="#"><u>nodal_outdegree</u>(java.lang.Object node)</a>
void	<a href="#"><u>notifyObservers</u>()</a>
void	<a href="#"><u>notifyObservers</u>(java.lang.Object arg)</a>
java.util.Set	<a href="#"><u>partition</u>(int id)</a> Retrieve a particular partition of the nodes.
int	<a href="#"><u>partitions</u>()</a>
void	<a href="#"><u>plugObservable</u>()</a>
java.lang.Iterable	<a href="#"><u>random_edge_iterable</u>(java.lang.Object n)</a>
java.lang.Iterable	<a href="#"><u>random_node_iterable</u>()</a>
boolean	<a href="#"><u>remove</u>(java.lang.Object from, java.lang.Object to)</a>
boolean	<a href="#"><u>remove</u>(java.lang.Object from, java.lang.Object to, java.lang.Object label)</a>

boolean	<a href="#">remove</a> (java.lang.Object o)
boolean	<a href="#">removeAll</a> (java.util.Collection clctn)
boolean	<a href="#">retainAll</a> (java.util.Collection clctn)
void	<a href="#">set</a> (java.lang.Object from, java.lang.Object to, java.lang.Object label)
void	<a href="#">set</a> (java.lang.Object from, java.lang.Object to, java.lang.Object label, int weight)
int	<a href="#">size()</a>
java.util.Set	<a href="#">successors</a> (java.lang.Object from)
java.lang.Object[]	<a href="#">toArray()</a>
java.lang.Object[]	<a href="#">toArray</a> (java.lang.Object[] ts)
void	<a href="#">unplugObservable</a> ()
void	<a href="#">update</a> (java.util.Observable o, java.lang.Object o1)

**Methods inherited from class** java.lang.Object

clone, equals, finalize, getClass, hashCode, notify, notifyAll, toString, wait, wait

**Methods inherited from interface** [org.nongnu.multigraph.Graph](#)

[add](#), [addObserver](#), [avg\\_nodal\\_degree](#), [clear\\_all\\_edges](#), [countObservers](#), [deleteObserver](#), [deleteObservers](#), [edge\\_outdegree](#), [edge](#), [edges](#), [edges](#), [hasChanged](#), [is\\_directed](#), [is\\_linked](#), [is\\_simple](#), [link\\_count](#), [max\\_nodal\\_degree](#), [nodal\\_outdegree](#), [notifyObservers](#), [notifyObservers](#), [plugObservable](#), [random\\_edge\\_iterable](#), [random\\_node\\_iterable](#), [remove](#), [remove](#), [set](#), [set](#), [successors](#), [unplugObservable](#)

**Methods inherited from interface** java.util.Set

[add](#), [addAll](#), [clear](#), [contains](#), [containsAll](#), [equals](#), [hashCode](#), [isEmpty](#), [iterator](#), [remove](#), [removeAll](#), [retainAll](#), [size](#), [spliterator](#), [toArray](#), [toArray](#)

**Methods inherited from interface** java.util.Collection

[add](#), [addAll](#), [clear](#), [contains](#), [containsAll](#), [equals](#), [hashCode](#), [isEmpty](#), [iterator](#), [parallelStream](#), [remove](#), [removeAll](#), [removeIf](#), [retainAll](#), [size](#), [spliterator](#), [stream](#), [toArray](#), [toArray](#)

**Methods inherited from interface** java.lang.Iterable

[forEach](#), [iterator](#), [spliterator](#)

**Methods inherited from interface** java.util.Observer

```
update
```

## Constructors

### PartitionGraph

```
public PartitionGraph(PartitionGraph.PartitionCallbacks cb)
```

## Methods

### partition

```
public java.util.Set partition(int id)
```

Retrieve a particular partition of the nodes.

**Parameters:**

id - The partition id to retrieve

**Returns:**

A Set of the nodes in this partition,

---

### partitions

```
public int partitions()
```

---

### is\_directed

```
public boolean is_directed()
```

---

### is\_simple

```
public boolean is_simple()
```

---

### set

```
public void set(java.lang.Object from,
               java.lang.Object to,
               java.lang.Object label)
```

(continued from last page)

**set**

```
public void set(java.lang.Object from,
               java.lang.Object to,
               java.lang.Object label,
               int weight)
```

---

**add**

```
public boolean add(java.lang.Object node)
```

---

**remove**

```
public boolean remove(java.lang.Object from,
                     java.lang.Object to,
                     java.lang.Object label)
```

---

**remove**

```
public boolean remove(java.lang.Object from,
                     java.lang.Object to)
```

---

**clear\_all\_edges**

```
public void clear_all_edges()
```

---

**edge\_outdegree**

```
public int edge_outdegree(java.lang.Object node)
```

---

**nodal\_outdegree**

```
public int nodal_outdegree(java.lang.Object node)
```

---

**avg\_nodal\_degree**

```
public float avg_nodal_degree()
```

---

**link\_count**

```
public long link_count()
```

---

**max\_nodal\_degree**

```
public int max_nodal_degree()
```

---

**successors**

```
public java.util.Set successors(java.lang.Object from)
```

---

**edges**

```
public java.util.Set edges(java.lang.Object from)
```

---

**edges**

```
public java.util.Collection edges(java.lang.Object from,
                                  java.lang.Object to)
```

---

**edge**

```
public Edge edge(java.lang.Object from,
                  java.lang.Object to)
```

---

**is\_linked**

```
public boolean is_linked(java.lang.Object from,
                        java.lang.Object to)
```

---

**edge**

```
public Edge edge(java.lang.Object from,
                  java.lang.Object to,
                  java.lang.Object label)
```

---

**random\_node\_iterable**

```
public java.lang.Iterable random_node_iterable()
```

---

**random\_edge\_iterable**

```
public java.lang.Iterable random_edge_iterable(java.lang.Object n)
```

---

**addObserver**

```
public void addObserver(java.util.Observer o)
```

---

**countObservers**

```
public int countObservers()
```

---

**deleteObserver**

```
public void deleteObserver(java.util.Observer o)
```

---

**deleteObservers**

```
public void deleteObservers()
```

---

**hasChanged**

```
public boolean hasChanged()
```

---

**notifyObservers**

```
public void notifyObservers()
```

---

**notifyObservers**

```
public void notifyObservers(java.lang.Object arg)
```

---

**plugObservable**

```
public void plugObservable()
```

---

**unplugObservable**

```
public void unplugObservable()
```

---

**size**

```
public int size()
```

---

(continued from last page)

---

**isEmpty**

```
public boolean isEmpty()
```

---

**contains**

```
public boolean contains(java.lang.Object o)
```

---

**iterator**

```
public java.util.Iterator iterator()
```

---

**toArray**

```
public java.lang.Object[] toArray()
```

---

**toArray**

```
public java.lang.Object[] toArray(java.lang.Object[] ts)
```

---

**remove**

```
public boolean remove(java.lang.Object o)
```

---

**containsAll**

```
public boolean containsAll(java.util.Collection clctn)
```

---

**addAll**

```
public boolean addAll(java.util.Collection clctn)
```

---

**retainAll**

```
public boolean retainAll(java.util.Collection clctn)
```

---

(continued from last page)

**removeAll**

```
public boolean removeAll(java.util.Collection clctn)
```

---

**clear**

```
public void clear()
```

---

**update**

```
public void update(java.util.Observable o,
                  java.lang.Object ol)
```

## org.nongnu.multigraph Interface PartitionGraph.PartitionCallbacks

---

public interface **PartitionGraph.PartitionCallbacks**  
extends

### Method Summary

abstract <a href="#">Graph</a>	<a href="#">create_graph()</a> Create and return the graph to be wrapped and partitioned
abstract int	<a href="#">node2partition(java.lang.Object node)</a>
abstract int	<a href="#">num_partitions()</a>

### Methods

#### **create\_graph**

public abstract [Graph](#) **create\_graph()**

Create and return the graph to be wrapped and partitioned

**Returns:**

The graph to be partitioned

---

#### **num\_partitions**

public abstract int **num\_partitions()**

**Returns:**

The number of partitions, this must not change.

---

#### **node2partition**

public abstract int **node2partition(java.lang.Object node)**

**Returns:**

Map a node to a partition ID.

## org.nongnu.multigraph Class ShortestPathFirst

```
java.lang.Object
+-org.nongnu.multigraph.ShortestPathFirst
```

---

```
public class ShortestPathFirst
extends java.lang.Object
```

Dijkstra's Shortest Path First algoritm, implemented to act on a Graph of N-nodes and Edges, with L-labels  
**Parameters:**

N - The type of the Nodes in the graph, E - The type of the Edges in the graph

### Constructor Summary

<code>public</code>	<a href="#">ShortestPathFirst(Graph g)</a>
---------------------	--

### Method Summary

<code>java.lang.Object</code>	<a href="#">nexthop(java.lang.Object to)</a>
	Return the next-hop node for the shortest path from the root node to the given node.
<code>java.util.List</code>	<a href="#">path(java.lang.Object to)</a>
	Return the path from the root node to the 'to' node, as a List of Edges, in the current SPF tree.
<code>java.lang.Object</code>	<a href="#">root()</a>
<code>void</code>	<a href="#">run(java.lang.Object root)</a>
	Construct the SPF tree rooted at the given node, to be used for subsequent shortest-path query call.

### Methods inherited from class `java.lang.Object`

<code>clone, equals, finalize, getClass, hashCode, notify, notifyAll, toString, wait, wait</code>
---

### Constructors

#### ShortestPathFirst

<code>public ShortestPathFirst(Graph g)</code>
--

### Methods

#### run

<code>public void run(java.lang.Object root)</code>
---

(continued from last page)

Construct the SPF tree rooted at the given node, to be used for subsequent shortest-path query call.

**Parameters:**

`root` - Root node for the Shortest-Path First tree.

---

**path**

```
public java.util.List path(java.lang.Object to)
```

Return the path from the root node to the 'to' node, as a List of Edges, in the current SPF tree. While our SPF supports equal-cost, multiple-paths, we only return one path, for simplicity's sake.

**Parameters:**

`to` - The node to query a path for

**Returns:**

List of Edges forming a path to the given node

---

**nexthop**

```
public java.lang.Object nexthop(java.lang.Object to)
```

Return the next-hop node for the shortest path from the root node to the given node.

**Parameters:**

`to` - Destination node to query path for

**Returns:**

The best next-hop from the root

---

**root**

```
public java.lang.Object root()
```

# org.nongnu.multigraph

## Class SimpleDiGraph

```
java.lang.Object
  +-java.util.Observable
    +-org.nongnu.multigraph.MultiDiGraph
      +-org.nongnu.multigraph.SimpleDiGraph
```

All Implemented Interfaces:  
[Graph](#)

Direct Known Subclasses:  
[SimpleGraph](#)

**public class SimpleDiGraph**  
**extends MultiDiGraph**

Simple, directed edge graph: no self-loop edges allowed and no more than 1 edge between nodes.

**Parameters:**

N - The type of the Nodes in the graph, E - The type of the Edges in the graph

### Constructor Summary

public	<a href="#">SimpleDiGraph()</a>
--------	---------------------------------

### Method Summary

void	<a href="#">_set(java.lang.Object from, java.lang.Object to, int weight, java.lang.Object label)</a>
------	--

boolean	<a href="#">is_simple()</a>
---------	-----------------------------

#### Methods inherited from class [org.nongnu.multigraph.MultiDiGraph](#)

<a href="#">_remove</a> , <a href="#">_set</a> , <a href="#">_add</a> , <a href="#">addAll</a> , <a href="#">avg_nodal_degree</a> , <a href="#">clear_all_edges</a> , <a href="#">clear</a> , <a href="#">contains</a> , <a href="#">containsAll</a> , <a href="#">edge_outdegree</a> , <a href="#">edge</a> , <a href="#">edge</a> , <a href="#">edges</a> , <a href="#">edges</a> , <a href="#">equals</a> , <a href="#">hashCode</a> , <a href="#">is_directed</a> , <a href="#">is_linked</a> , <a href="#">is_simple</a> , <a href="#">isEmpty</a> , <a href="#">iterator</a> , <a href="#">link_count</a> , <a href="#">max_nodal_degree</a> , <a href="#">nodal_outdegree</a> , <a href="#">notifyObservers</a> , <a href="#">notifyObservers</a> , <a href="#">plugObservable</a> , <a href="#">random_edge_iterable</a> , <a href="#">random_node_iterable</a> , <a href="#">remove</a> , <a href="#">remove</a> , <a href="#">remove</a> , <a href="#">removeAll</a> , <a href="#">retainAll</a> , <a href="#">set</a> , <a href="#">set</a> , <a href="#">size</a> , <a href="#">successors</a> , <a href="#">toArray</a> , <a href="#">toString</a> , <a href="#">unplugObservable</a>
--

#### Methods inherited from class [java.util.Observable](#)

<a href="#">addObserver</a> , <a href="#">clearChanged</a> , <a href="#">countObservers</a> , <a href="#">deleteObserver</a> , <a href="#">deleteObservers</a> , <a href="#">hasChanged</a> , <a href="#">notifyObservers</a> , <a href="#">notifyObservers</a> , <a href="#">setChanged</a>
--

#### Methods inherited from class [java.lang.Object](#)

<a href="#">clone</a> , <a href="#">equals</a> , <a href="#">finalize</a> , <a href="#">getClass</a> , <a href="#">hashCode</a> , <a href="#">notify</a> , <a href="#">notifyAll</a> , <a href="#">toString</a> , <a href="#">wait</a> , <a href="#">wait</a>
---

**Methods inherited from interface** [org.nongnu.multigraph.Graph](#)

```
add, addObserver, avg_nodal_degree, clear_all_edges, countObservers, deleteObserver,
deleteObservers, edge_outdegree, edge, edges, edges, hasChanged, is_directed,
is_linked, is_simple, link_count, max_nodal_degree, nodal_outdegree, notifyObservers,
notifyObservers, plugObservable, random_edge_iterable, random_node_iterable, remove,
remove, set, set, successors, unplugObservable
```

**Methods inherited from interface** [java.util.Set](#)

```
add, addAll, clear, contains, containsAll, equals, hashCode, isEmpty, iterator,
remove, removeAll, retainAll, size, spliterator, toArray, toArray
```

**Methods inherited from interface** [java.util.Collection](#)

```
add, addAll, clear, contains, containsAll, equals, hashCode, isEmpty, iterator,
parallelStream, remove, removeAll, removeIf, retainAll, size, spliterator, stream,
toArray, toArray
```

**Methods inherited from interface** [java.lang.Iterable](#)

```
forEach, iterator, spliterator
```

## Constructors

### **SimpleDiGraph**

```
public SimpleDiGraph()
```

## Methods

### **\_set**

```
protected void _set(java.lang.Object from,
                   java.lang.Object to,
                   int weight,
                   java.lang.Object label)
```

The core, central set method. All other set methods are filters for this method, meant to check invariants, apply various requirements. etc. Exported to package so that other types of graph may be subclassed from this

### **is\_simple**

```
public boolean is_simple()
```

# org.nongnu.multigraph

## Class SimpleGraph

```
java.lang.Object
  +-java.util.Observable
    +-org.nongnu.multigraph.MultiDiGraph
      +-org.nongnu.multigraph.SimpleDiGraph
        +-org.nongnu.multigraph.SimpleGraph
```

All Implemented Interfaces:  
[Graph](#)

public class **SimpleGraph**  
 extends [SimpleDiGraph](#)

Simple, undirected graph: No self-loop edges allowed and 0 or 1 edges between nodes.

**Parameters:**

N - The type of the Nodes in the graph, E - The type of the Edges in the graph

**See Also:**

[SimpleDiGraph](#)

### Constructor Summary

public	<a href="#">SimpleGraph()</a>
--------	-------------------------------

### Method Summary

boolean	<a href="#">remove</a> (java.lang.Object from, java.lang.Object to, java.lang.Object label)
void	<a href="#">set</a> (java.lang.Object from, java.lang.Object to, int weight, java.lang.Object label) The core, central set method. All other set methods are filters for this method, meant to check invariants, apply various requirements. etc. Exported to package so that other types of graph may be subclassed from this
boolean	<a href="#">is_directed()</a>

#### Methods inherited from class [org.nongnu.multigraph.SimpleDiGraph](#)

[\\_set](#), [is\\_simple](#)

#### Methods inherited from class [org.nongnu.multigraph.MultiDiGraph](#)

[\\_remove](#), [\\_set](#), [add](#), [addAll](#), [avg\\_nodal\\_degree](#), [clear\\_all\\_edges](#), [clear](#), [contains](#), [containsAll](#), [edge\\_outdegree](#), [edge](#), [edge](#), [edges](#), [edges](#), [equals](#), [hashCode](#), [is\\_directed](#), [is\\_linked](#), [is\\_simple](#), [isEmpty](#), [iterator](#), [link\\_count](#), [max\\_nodal\\_degree](#), [nodal\\_outdegree](#), [notifyObservers](#), [notifyObservers](#), [plugObservable](#), [random\\_edge\\_iterable](#), [random\\_node\\_iterable](#), [remove](#), [remove](#), [remove](#), [removeAll](#), [retainAll](#), [set](#), [set](#), [size](#), [successors](#), [toArray](#), [toArray](#), [toString](#), [unplugObservable](#)

#### Methods inherited from class [java.util.Observable](#)

```
addObserver, clearChanged, countObservers, deleteObserver, deleteObservers,
hasChanged, notifyObservers, notifyObservers, setChanged
```

#### Methods inherited from class java.lang.Object

```
clone, equals, finalize, getClass, hashCode, notify, notifyAll, toString, wait, wait,
```

#### Methods inherited from interface org.nongnu.multigraph.Graph

```
add, addObserver, avg_nodal_degree, clear_all_edges, countObservers, deleteObserver,
deleteObservers, edge_outdegree, edge, edges, edges, hasChanged, is_directed,
is_linked, is_simple, link_count, max_nodal_degree, nodal_outdegree, notifyObservers,
notifyObservers, plugObservable, random_edge_iterable, random_node_iterable, remove,
remove, set, set, successors, unplugObservable
```

#### Methods inherited from interface java.util.Set

```
add, addAll, clear, contains, containsAll, equals, hashCode, isEmpty, iterator,
remove, removeAll, retainAll, size, spliterator, toArray, toArray
```

#### Methods inherited from interface java.util.Collection

```
add, addAll, clear, contains, containsAll, equals, hashCode, isEmpty, iterator,
parallelStream, remove, removeAll, removeIf, retainAll, size, spliterator, stream,
toArray, toArray
```

#### Methods inherited from interface java.lang.Iterable

```
forEach, iterator, spliterator
```

## Constructors

### SimpleGraph

```
public SimpleGraph()
```

## Methods

### \_remove

```
protected boolean _remove(java.lang.Object from,
                        java.lang.Object to,
                        java.lang.Object label)
```

The core, central edge removal method. All other remove methods are filters for this method, meant to check invariants, apply various requirements. etc. Exported to package so that other types of graph may be subclassed from this

(continued from last page)

**\_set**

```
protected void _set(java.lang.Object from,
                   java.lang.Object to,
                   int weight,
                   java.lang.Object label)
```

The core, central set method. All other set methods are filters for this method, meant to check invariants, apply various requirements. etc. Exported to package so that other types of graph may be subclassed from this

---

**is\_directed**

```
public boolean is_directed()
```

# org.nongnu.multigraph Class SyncGraph

```
java.lang.Object
+-org.nongnu.multigraph.SyncGraph
```

## All Implemented Interfaces:

[Graph](#)

```
public class SyncGraph
extends java.lang.Object
implements Graph
```

## Constructor Summary

public	<a href="#">SyncGraph(Graph graph)</a>
--------	--

## Method Summary

boolean	<a href="#">add(java.lang.Object node)</a>
boolean	<a href="#">addAll(java.util.Collection cltn)</a>
void	<a href="#">addObserver(java.util.Observer o)</a>
float	<a href="#">avg_nodal_degree()</a>
void	<a href="#">clear_all_edges()</a>
void	<a href="#">clear()</a>
boolean	<a href="#">contains(java.lang.Object o)</a>
boolean	<a href="#">containsAll(java.util.Collection cltn)</a>
int	<a href="#">countObservers()</a>
void	<a href="#">deleteObserver(java.util.Observer o)</a>
void	<a href="#">deleteObservers()</a>
int	<a href="#">edge_outdegree(java.lang.Object node)</a>
<a href="#">Edge</a>	<a href="#">edge(java.lang.Object from, java.lang.Object to)</a>

<a href="#"><u>Edge</u></a>	<a href="#"><u>edge</u>(java.lang.Object from, java.lang.Object to, java.lang.Object label)</a>
<a href="#"><u>java.util.Set</u></a>	<a href="#"><u>edges</u>(java.lang.Object from)</a>
<a href="#"><u>java.util.Collection</u></a>	<a href="#"><u>edges</u>(java.lang.Object from, java.lang.Object to)</a>
<a href="#"><u>boolean</u></a>	<a href="#"><u>hasChanged()</u></a>
<a href="#"><u>boolean</u></a>	<a href="#"><u>is_directed()</u></a>
<a href="#"><u>boolean</u></a>	<a href="#"><u>is_linked</u>(java.lang.Object from, java.lang.Object to)</a>
<a href="#"><u>boolean</u></a>	<a href="#"><u>is_simple()</u></a>
<a href="#"><u>boolean</u></a>	<a href="#"><u>isEmpty()</u></a>
<a href="#"><u>java.util.Iterator</u></a>	<a href="#"><u>iterator()</u></a>
<a href="#"><u>long</u></a>	<a href="#"><u>link_count()</u></a>
<a href="#"><u>int</u></a>	<a href="#"><u>max_nodal_degree()</u></a>
<a href="#"><u>int</u></a>	<a href="#"><u>nodal_outdegree</u>(java.lang.Object node)</a>
<a href="#"><u>void</u></a>	<a href="#"><u>notifyObservers()</u></a>
<a href="#"><u>void</u></a>	<a href="#"><u>notifyObservers</u>(java.lang.Object arg)</a>
<a href="#"><u>void</u></a>	<a href="#"><u>plugObservable()</u></a>
<a href="#"><u>java.lang.Iterable</u></a>	<a href="#"><u>random_edge_iterable</u>(java.lang.Object n)</a>
<a href="#"><u>java.lang.Iterable</u></a>	<a href="#"><u>random_node_iterable()</u></a>
<a href="#"><u>boolean</u></a>	<a href="#"><u>remove</u>(java.lang.Object from, java.lang.Object to)</a>
<a href="#"><u>boolean</u></a>	<a href="#"><u>remove</u>(java.lang.Object from, java.lang.Object to, java.lang.Object label)</a>
<a href="#"><u>boolean</u></a>	<a href="#"><u>remove</u>(java.lang.Object o)</a>
<a href="#"><u>boolean</u></a>	<a href="#"><u>removeAll</u>(java.util.Collection clctn)</a>
<a href="#"><u>boolean</u></a>	<a href="#"><u>retainAll</u>(java.util.Collection clctn)</a>
<a href="#"><u>void</u></a>	<a href="#"><u>set</u>(java.lang.Object from, java.lang.Object to, java.lang.Object label)</a>

void	<a href="#">set</a> (java.lang.Object from, java.lang.Object to, java.lang.Object label, int weight)
int	<a href="#">size()</a>
java.util.Set	<a href="#">successors</a> (java.lang.Object from)
java.lang.Object[]	<a href="#">toArray()</a>
java.lang.Object[]	<a href="#">toArray</a> (java.lang.Object[] ts)
void	<a href="#">unplugObservable()</a>

**Methods inherited from class** java.lang.Object

clone, equals, finalize, getClass, hashCode, notify, notifyAll, toString, wait, wait, wait

**Methods inherited from interface** [org.nongnu.multigraph.Graph](#)

[add](#), [addObserver](#), [avg\\_nodal\\_degree](#), [clear\\_all\\_edges](#), [countObservers](#), [deleteObserver](#), [deleteObservers](#), [edge\\_outdegree](#), [edge](#), [edges](#), [edges](#), [hasChanged](#), [is\\_directed](#), [is\\_linked](#), [is\\_simple](#), [link\\_count](#), [max\\_nodal\\_degree](#), [nodal\\_outdegree](#), [notifyObservers](#), [notifyObservers](#), [plugObservable](#), [random\\_edge\\_iterable](#), [random\\_node\\_iterable](#), [remove](#), [remove](#), [set](#), [set](#), [successors](#), [unplugObservable](#)

**Methods inherited from interface** java.util.Set

[add](#), [addAll](#), [clear](#), [contains](#), [containsAll](#), [equals](#), [hashCode](#), [isEmpty](#), [iterator](#), [remove](#), [removeAll](#), [retainAll](#), [size](#), [spliterator](#), [toArray](#), [toArray](#)

**Methods inherited from interface** java.util.Collection

[add](#), [addAll](#), [clear](#), [contains](#), [containsAll](#), [equals](#), [hashCode](#), [isEmpty](#), [iterator](#), [parallelStream](#), [remove](#), [removeAll](#), [removeIf](#), [retainAll](#), [size](#), [spliterator](#), [stream](#), [toArray](#), [toArray](#)

**Methods inherited from interface** java.lang.Iterable

[forEach](#), [iterator](#), [spliterator](#)

## Constructors

### SyncGraph

```
public SyncGraph(Graph graph)
```

## Methods

(continued from last page)

**is\_directed**

```
public boolean is_directed()
```

---

**is\_simple**

```
public boolean is_simple()
```

---

**set**

```
public void set(java.lang.Object from,  
               java.lang.Object to,  
               java.lang.Object label)
```

---

**set**

```
public void set(java.lang.Object from,  
               java.lang.Object to,  
               java.lang.Object label,  
               int weight)
```

---

**add**

```
public boolean add(java.lang.Object node)
```

---

**remove**

```
public boolean remove(java.lang.Object from,  
                     java.lang.Object to,  
                     java.lang.Object label)
```

---

**remove**

```
public boolean remove(java.lang.Object from,  
                     java.lang.Object to)
```

---

**clear\_all\_edges**

```
public void clear_all_edges()
```

(continued from last page)

**edge\_outdegree**

```
public int edge_outdegree(java.lang.Object node)
```

---

**nodal\_outdegree**

```
public int nodal_outdegree(java.lang.Object node)
```

---

**avg\_nodal\_degree**

```
public float avg_nodal_degree()
```

---

**link\_count**

```
public long link_count()
```

---

**max\_nodal\_degree**

```
public int max_nodal_degree()
```

---

**successors**

```
public java.util.Set successors(java.lang.Object from)
```

---

**edges**

```
public java.util.Set edges(java.lang.Object from)
```

---

**edges**

```
public java.util.Collection edges(java.lang.Object from,  
                                 java.lang.Object to)
```

---

**edge**

```
public Edge edge(java.lang.Object from,  
                java.lang.Object to)
```

(continued from last page)

**is\_linked**

```
public boolean is_linked(java.lang.Object from,  
                      java.lang.Object to)
```

---

**edge**

```
public Edge edge(java.lang.Object from,  
                  java.lang.Object to,  
                  java.lang.Object label)
```

---

**random\_node\_iterable**

```
public java.lang.Iterable random_node_iterable()
```

---

**random\_edge\_iterable**

```
public java.lang.Iterable random_edge_iterable(java.lang.Object n)
```

---

**addObserver**

```
public void addObserver(java.util.Observer o)
```

---

**countObservers**

```
public int countObservers()
```

---

**deleteObserver**

```
public void deleteObserver(java.util.Observer o)
```

---

**deleteObservers**

```
public void deleteObservers()
```

---

**hasChanged**

```
public boolean hasChanged()
```

(continued from last page)

**notifyObservers**

```
public void notifyObservers()
```

**notifyObservers**

```
public void notifyObservers(java.lang.Object arg)
```

**plugObservable**

```
public void plugObservable()
```

**unplugObservable**

```
public void unplugObservable()
```

**size**

```
public int size()
```

**isEmpty**

```
public boolean isEmpty()
```

**contains**

```
public boolean contains(java.lang.Object o)
```

**iterator**

```
public java.util.Iterator iterator()
```

**toArray**

```
public java.lang.Object[] toArray()
```

**toArray**

```
public java.lang.Object[] toArray(java.lang.Object[] ts)
```

(continued from last page)

---

**remove**

```
public boolean remove(java.lang.Object o)
```

---

**containsAll**

```
public boolean containsAll(java.util.Collection clctn)
```

---

**addAll**

```
public boolean addAll(java.util.Collection clctn)
```

---

**retainAll**

```
public boolean retainAll(java.util.Collection clctn)
```

---

**removeAll**

```
public boolean removeAll(java.util.Collection clctn)
```

---

**clear**

```
public void clear()
```

---

**Package**

**org.nongnu.multigraph.layout**

## org.nongnu.multigraph.layout Class AbstractPositionableNode

```
java.lang.Object
+-org.nongnu.multigraph.layout.AbstractPositionableNode
```

### All Implemented Interfaces:

[PositionableNode](#)

```
public abstract class AbstractPositionableNode
extends java.lang.Object
implements PositionableNode
```

## Constructor Summary

public	<a href="#">AbstractPositionableNode()</a>
--------	--

## Method Summary

float	<a href="#">getMass()</a>
<a href="#">Vector2D</a>	<a href="#">getPosition()</a>
float	<a href="#">getSize()</a>
<a href="#">Vector2D</a>	<a href="#">getVelocity()</a>
boolean	<a href="#">isMovable()</a>
void	<a href="#">setMass(float m)</a>
void	<a href="#">setPosition(<a href="#">Vector2D</a> p)</a>
void	<a href="#">setSize(float s)</a>

### Methods inherited from class `java.lang.Object`

`clone`, `equals`, `finalize`, `getClass`, `hashCode`, `notify`, `notifyAll`, `toString`, `wait`, `wait`, `wait`

### Methods inherited from interface [org.nongnu.multigraph.layout.PositionableNode](#)

[getMass](#), [getPosition](#), [getSize](#), [getVelocity](#), [isMovable](#), [setMass](#), [setPosition](#), [setSize](#)

## Constructors

(continued from last page)

## AbstractPositionableNode

```
public AbstractPositionableNode()
```

### Methods

#### getMass

```
public float getMass()
```

---

#### getPosition

```
public Vector2D getPosition()
```

---

#### setPosition

```
public void setPosition(Vector2D p)
```

---

#### getVelocity

```
public Vector2D getVelocity()
```

---

#### setMass

```
public void setMass(float m)
```

---

#### getSize

```
public float getSize()
```

---

#### setSize

```
public void setSize(float s)
```

---

#### isMovable

```
public boolean isMovable()
```

## org.nongnu.multigraph.layout Class ForceLayout

```
java.lang.Object
  +-org.nongnu.multigraph.layout.Layout
    +-org.nongnu.multigraph.layout.ForceLayout
```

**public class ForceLayout**  
**extends Layout**

See "Graph Drawing by Force-directed Placement", Fruchterman & Reingold.

This algorithm tries to layout a graph as if the nodes are repelled by each other, exponentially more so as they get closer to each other, while at the same time the edges act like springs to pull nodes together. These are combined to give each node a velocity and momentum, acting in tension on other nodes. If node's have a mass, this is taken into account.

The algorithm takes a number of iterations to reach equilibrium, presuming there are no other forces acting on the graph.

The behaviour of the algorithm is can be tuned by a number of parameters. Unfortunately, different graphs may require different values for certain parameters for best effect.

**Parameters:**

N - The type of the Nodes in the graph, E - The type of the Edges in the graph

### Constructor Summary

public	<a href="#">ForceLayout(Graph graph, java.awt.Dimension bound, int maxiterations)</a>
public	<a href="#">ForceLayout(Graph graph, java.awt.Dimension bound, int maxiterations, double initial_temperature)</a> Create a ForceLayout instance, with the initial temperature set as given.
public	<a href="#">ForceLayout(Graph graph, java.awt.Dimension bound, int maxiterations, double initial_temperature, double C)</a> Create a ForceLayout instance, with the initial temperature set as given, and using the specific C scalar for the k parameter of the algorithm.

### Method Summary

boolean	<a href="#">layout(float interval)</a>
<a href="#">ForceLayout</a>	<a href="#">setC(double C)</a>
<a href="#">ForceLayout</a>	<a href="#">setDecay(double decay)</a> The velocity on each iteration is scaled by a 'temperature' factor, which is decayed according to temperature(t+1) = temperature(t) * decay, or temperature (t) = decay <sup>t</sup> .
void	<a href="#">setJiggle(double jiggle)</a>
<a href="#">ForceLayout</a>	<a href="#">setMinkve(double minkve)</a>
<a href="#">ForceLayout</a>	<a href="#">setMintemp(double mintemp)</a> Set the minimum temperature possible.

**Methods inherited from class** [org.nongnu.multigraph.layout.Layout](#)[factory](#), [isaLayout](#), [layout](#), [maxiterations](#), [maxiterations](#)**Methods inherited from class** [java.lang.Object](#)[clone](#), [equals](#), [finalize](#), [getClass](#), [hashCode](#), [notify](#), [notifyAll](#), [toString](#), [wait](#), [wait](#)

## Constructors

### ForceLayout

```
public ForceLayout(Graph graph,
                  java.awt.Dimension bound,
                  int maxiterations)
```

### ForceLayout

```
public ForceLayout(Graph graph,
                  java.awt.Dimension bound,
                  int maxiterations,
                  double initial_temperature)
```

Create a ForceLayout instance, with the initial temperature set as given.

**Parameters:**

- `graph` - The Graph to act on
- `bound` - The boundary to apply to the layout algorithm
- `maxiterations` - The maximum number of iterations to run for.
- `initial_temperature` - The initial temperature scale factor to apply.

**See Also:**

[for further discussion of temperature.](#)

### ForceLayout

```
public ForceLayout(Graph graph,
                  java.awt.Dimension bound,
                  int maxiterations,
                  double initial_temperature,
                  double C)
```

Create a ForceLayout instance, with the initial temperature set as given, and using the specific C scalar for the k parameter of the algorithm.

C defaults to 1.

**Parameters:**

- `graph`
- `bound`
- `maxiterations`
- `initial_temperature`
- `C` - scalar to apply to the k constant, which is used to calculate the attractive and repulsive forces.

## Methods

(continued from last page)

## setMintemp

```
public ForceLayout setMintemp(double mintemp)
```

Set the minimum temperature possible.

The velocity calculated for a node on each iteration is scaled by a 'temperature' factor. This temperature decays on each iteration, to simulate the algorithm getting 'colder' - the idea being to allow the algorithm to stabilise.

Generally, you want this parameter set so that in later iterations of the algorithm there is still some amount of energy attainable to allow nodes to move around, while being low enough to damp out any wild movement of nodes.

On dense graphs, and/or graphs with very well connected nodes, you will want this value to be lower, to prevent oscillations. On more evenly distributed graphs, this value can be set higher.

The default is 0.001

**Parameters:**

`mintemp` - The minimum temperature that can apply. Generally this should be 1 or less.

## setDecay

```
public ForceLayout setDecay(double decay)
```

The velocity on each iteration is scaled by a 'temperature' factor, which is decayed according to  $\text{temperature}(t+1) = \text{temperature}(t) * \text{decay}$ , or  $\text{temperature}(t) = \text{decay}^t$ .

This defaults to 0.94.

**Parameters:**

`decay` - The decay factor to apply to the temperature. Generally it should be  $\leq 1$ . Higher values lead to a more linear decay. Lower values to a more exponential and initially rapid decay.

**See Also:**

[setMintemp](#)

## setMinkve

```
public ForceLayout setMinkve(double minkve)
```

**Parameters:**

`minkve` - Sets the minimum sum of kinetic energy values, below which the algorithm will be considered to no longer have any movement.

Default: 0.1

## setC

```
public ForceLayout setC(double C)
```

**Parameters:**

`C` - Sets the C parameter of the algorithm, which is used to scale the k factor of the algorithm. Increasing this factor will magnify the repulsive and attractive forces, decreasing will minimise them. Values around 1 will try balance nodes around the area. Values above will tend to force many nodes against the boundary. Values below 1 will tend to cause nodes to cluster more toward centre of the area.

Default: 1

## setJiggle

```
public void setJiggle(double jiggle)
```

### Parameters:

jiggle - Adds some entropy to the algorithm. Higher values add more jiggle. Recommended value is between 0 and 1. Defaults to 0.1. Note that increasing this value will increase the background 'heat' of the algorithm, and you may need to increase maxIterations if you're depending on it rather than minkve.

---

## layout

```
public boolean layout(float interval)
```

returns true if layout can still change. If false, then there is no more the layout can do to the graph. This default implementation does nothing except enforce the maximum iterations constraint.

# org.nongnu.multigraph.layout

## Class Layout

```
java.lang.Object
+-org.nongnu.multigraph.layout.Layout
```

**Direct Known Subclasses:**

[RadialLayout](#), [ForceLayout](#), [NullLayout](#), [RandomLayout](#)

public abstract class **Layout**  
extends java.lang.Object

Abstract implementation of a layout algorithm. To be used something like:

```
new Layout l = Layout.factory ("layout_name", graph, bound_dimension, 10);
while (l.layout (time_passed));
<do whatever other work>
```

In order to be able to apply a layout algorithm to a graph, the graph's N-type nodes must implement the [PositionableNode](#) interface.

**Parameters:**

N - The type of the Nodes in the graph, which must implement PositionableNode, E - The type of the Edges in the graph

### Constructor Summary

public	<a href="#">Layout</a> ( <a href="#">Graph</a> graph, java.awt.Dimension bound)
public	<a href="#">Layout</a> ( <a href="#">Graph</a> graph, java.awt.Dimension bound, int maxiterations)

### Method Summary

static <a href="#">Layout</a>	<a href="#">factory</a> (java.lang.String algnname, <a href="#">Graph</a> graph, java.awt.Dimension bound, int maxiterations)
static boolean	<a href="#">isaLayout</a> (java.lang.String name) Query whether the given string is an implemented layout algorithm.
boolean	<a href="#">layout</a> (float interval) returns true if layout can still change.
int	<a href="#">maxiterations()</a>
<a href="#">Layout</a>	<a href="#">maxiterations</a> (int maxiterations)

### Methods inherited from class java.lang.Object

[clone](#), [equals](#), [finalize](#), [getClass](#), [hashCode](#), [notify](#), [notifyAll](#), [toString](#), [wait](#), [wait](#)

## Constructors

### Layout

```
public Layout(Graph graph,  
             java.awt.Dimension bound)
```

---

### Layout

```
public Layout(Graph graph,  
             java.awt.Dimension bound,  
             int maxiterations)
```

## Methods

### maxiterations

```
public int maxiterations()
```

---

### maxiterations

```
public Layout maxiterations(int maxiterations)
```

---

### layout

```
public boolean layout(float interval)
```

returns true if layout can still change. If false, then there is no more the layout can do to the graph. This default implementation does nothing except enforce the maximum iterations constraint.

---

### isALayout

```
public static boolean isALayout(java.lang.String name)
```

Query whether the given string is an implemented layout algorithm.

**Parameters:**

name - Layout algorith name

**Returns:**

Whether a layout implementation exists for the given algorithm name

---

### factory

```
public final static Layout factory(java.lang.String alname,  
                                   Graph graph,  
                                   java.awt.Dimension bound,  
                                   int maxiterations)
```

(continued from last page)

## org.nongnu.multigraph.layout Class NullLayout

```
java.lang.Object
+-org.nongnu.multigraph.layout.Layout
  +-org.nongnu.multigraph.layout.NullLayout
```

---

**public class NullLayout**  
extends [Layout](#)

---

### Constructor Summary

public	<a href="#">NullLayout(Graph graph, java.awt.Dimension bound, int maxiterations)</a>
--------	--

### Methods inherited from class org.nongnu.multigraph.layout.Layout

<a href="#">factory</a> , <a href="#">isaLayout</a> , <a href="#">layout</a> , <a href="#">maxiterations</a> , <a href="#">maxiterations</a>
--

### Methods inherited from class java.lang.Object

clone, equals, finalize, getClass, hashCode, notify, notifyAll, toString, wait, wait
--

## Constructors

### NullLayout

```
public NullLayout(Graph graph,
                  java.awt.Dimension bound,
                  int maxiterations)
```

## org.nongnu.multigraph.layout Interface PositionableNode

All Known Implementing Classes:  
[AbstractPositionableNode](#)

---

public interface **PositionableNode**  
extends

Interface that is required to be implemented by nodes of a graph, if any Layout algorithm is to be able to act on them.

### Method Summary

abstract float	<a href="#">getMass()</a>
abstract <a href="#">Vector2D</a>	<a href="#">getPosition()</a>
abstract float	<a href="#">getSize()</a>
abstract <a href="#">Vector2D</a>	<a href="#">getVelocity()</a>
abstract boolean	<a href="#">isMovable()</a>
abstract void	<a href="#">setMass(float m)</a>
abstract void	<a href="#">setPosition(<a href="#">Vector2D</a> pos)</a>
abstract void	<a href="#">setSize(float s)</a>

### Methods

#### getPosition

public abstract [Vector2D](#) [getPosition\(\)](#)

---

#### setPosition

public abstract void [setPosition\(\[Vector2D\]\(#\) pos\)](#)

---

#### getVelocity

public abstract [Vector2D](#) [getVelocity\(\)](#)

---

### **getSize**

```
public abstract float getSize()
```

---

### **setSize**

```
public abstract void setSize(float s)
```

---

### **getMass**

```
public abstract float getMass()
```

---

### **setMass**

```
public abstract void setMass(float m)
```

---

### **isMovable**

```
public abstract boolean isMovable()
```

# org.nongnu.multigraph.layout

## Class RadialLayout

```
java.lang.Object
+-org.nongnu.multigraph.layout.Layout
  +-org.nongnu.multigraph.layout.RadialLayout
```

**public class RadialLayout**  
**extends Layout**

Layout the graph's nodes radially, in a nice circle. It tries to scale the node size according to the graph.

**Parameters:**

N - The type of the Nodes in the graph, E - The type of the Edges in the graph

### Constructor Summary

public	<a href="#">RadialLayout(Graph graph, java.awt.Dimension bound, int maxiterations)</a>
--------	--

### Method Summary

boolean	<a href="#">layout(float interval)</a>
---------	--

#### Methods inherited from class [org.nongnu.multigraph.layout.Layout](#)

[factory](#), [isarLayout](#), [layout](#), [maxiterations](#), [maxiterations](#)

#### Methods inherited from class [java.lang.Object](#)

[clone](#), [equals](#), [finalize](#), [getClass](#), [hashCode](#), [notify](#), [notifyAll](#), [toString](#), [wait](#), [wait](#), [wait](#)

### Constructors

#### RadialLayout

```
public RadialLayout(Graph graph,
                    java.awt.Dimension bound,
                    int maxiterations)
```

### Methods

#### layout

```
public boolean layout(float interval)
```

returns true if layout can still change. If false, then there is no more the layout can do to the graph. This default implementation does nothing except enforce the maximum iterations constraint.

# org.nongnu.multigraph.layout

## Class RandomLayout

```
java.lang.Object
+-org.nongnu.multigraph.layout.Layout
  +-org.nongnu.multigraph.layout.RandomLayout
```

**public class RandomLayout**  
**extends Layout**

Random layout of a graph. This layout can be interesting to apply prior to a ForceLayout.

**Parameters:**

N - The type of the Nodes in the graph, E - The type of the Edges in the graph

### Constructor Summary

public	<a href="#">RandomLayout(Graph graph, java.awt.Dimension bound, int maxiterations)</a>
--------	--

### Method Summary

boolean	<a href="#">layout(float interval)</a>
---------	--

#### Methods inherited from class [org.nongnu.multigraph.layout.Layout](#)

[factory](#), [isalayout](#), [layout](#), [maxiterations](#), [maxiterations](#)

#### Methods inherited from class [java.lang.Object](#)

[clone](#), [equals](#), [finalize](#), [getclass](#), [hashcode](#), [notify](#), [notifyall](#), [tostring](#), [wait](#), [wait](#), [wait](#)

### Constructors

#### RandomLayout

```
public RandomLayout(Graph graph,
                    java.awt.Dimension bound,
                    int maxiterations)
```

### Methods

#### layout

```
public boolean layout(float interval)
```

returns true if layout can still change. If false, then there is no more the layout can do to the graph. This default implementation does nothing except enforce the maximum iterations constraint.

## org.nongnu.multigraph.layout Class Vector2D

```
java.lang.Object
  +-java.awt.geom.Point2D
    +-java.awt.geom.Point2D.Double
      +-org.nongnu.multigraph.layout.Vector2D
```

### All Implemented Interfaces:

java.lang.Cloneable, java.io.Serializable

```
public class Vector2D
extends java.awt.geom.Point2D.Double
```

### Fields inherited from class java.awt.geom.Point2D.Double

x, y

## Constructor Summary

public	<a href="#">Vector2D()</a>
public	<a href="#">Vector2D(double arg0, double arg1)</a>
public	<a href="#">Vector2D(java.awt.geom.Point2D p)</a>

## Method Summary

double	<a href="#">length()</a>
double	<a href="#">magnitude()</a>
<a href="#">Vector2D</a>	<a href="#">minus(double x, double y)</a>
<a href="#">Vector2D</a>	<a href="#">minus(java.awt.geom.Point2D v)</a>
void	<a href="#">normalise()</a>
<a href="#">Vector2D</a>	<a href="#">normalise(<a href="#">Vector2D</a> v)</a>
<a href="#">Vector2D</a>	<a href="#">plus(double x, double y)</a>
<a href="#">Vector2D</a>	<a href="#">plus(java.awt.geom.Point2D v)</a>
<a href="#">Vector2D</a>	<a href="#">rotate(double n)</a>

<a href="#">Vector2D</a>	<a href="#">times</a> (double n)
--------------------------	----------------------------------

**Methods inherited from class** java.awt.geom.Point2D.Double

getX, getY, setLocation, toString

**Methods inherited from class** java.awt.geom.Point2D

clone, distance, distance, distance, distanceSq, distanceSq, distanceSq, equals, getX, getY, hashCode, setLocation, setLocation

**Methods inherited from class** java.lang.Object

clone, equals, finalize, getClass, hashCode, notify, notifyAll, toString, wait, wait, wait

## Constructors

**Vector2D**public **Vector2D**()**Vector2D**public **Vector2D**(double arg0,  
double arg1)**Vector2D**public **Vector2D**(java.awt.geom.Point2D p)

## Methods

**normalise**public void **normalise**()**magnitude**public double **magnitude**()**normalise**public [Vector2D](#) **normalise**([Vector2D](#) v)

(continued from last page)

---

**length**

```
public double length()
```

---

---

**plus**

```
public Vector2D plus(java.awt.geom.Point2D v)
```

---

---

**plus**

```
public Vector2D plus(double x,  
                     double y)
```

---

---

**minus**

```
public Vector2D minus(java.awt.geom.Point2D v)
```

---

---

**minus**

```
public Vector2D minus(double x,  
                     double y)
```

---

---

**times**

```
public Vector2D times(double n)
```

---

---

**rotate**

```
public Vector2D rotate(double n)
```

---

---

**Package**

## **org.nongnu.multigraph.metrics**

## org.nongnu.multigraph.metrics Class dmap

```
java.lang.Object
+-org.nongnu.multigraph.metrics.dmap
```

---

```
public class dmap
extends java.lang.Object
```

Distance maps, from each node to every other node. Suitable for sparse graphs.

**Parameters:**

N

---

### Constructor Summary

public	<a href="#">dmap()</a>
--------	------------------------

### Method Summary

int	<a href="#">dist(java.lang.Object from, java.lang.Object to)</a>
-----	--

java.util.Set	<a href="#">entrySet()</a>
---------------	----------------------------

void	<a href="#">set(java.lang.Object from, java.lang.Object to, int w)</a>
------	--

### Methods inherited from class java.lang.Object

clone, equals, finalize, getClass, hashCode, notify, notifyAll, toString, wait, wait
--

---

### Constructors

#### dmap

```
public dmap()
```

---

### Methods

#### dist

```
public int dist(java.lang.Object from,
               java.lang.Object to)
```

---

(continued from last page)

## **set**

```
public void set(java.lang.Object from,  
                 java.lang.Object to,  
                 int w)
```

---

## **entrySet**

```
public java.util.Set entrySet()
```

## org.nongnu.multigraph.metrics Class TraversalMetrics

```
java.lang.Object
+-org.nongnu.multigraph.metrics.TraversalMetrics
```

---

```
public class TraversalMetrics
extends java.lang.Object
```

Return

### Nested Class Summary

class	<a href="#">TraversalMetrics.graph_traversor</a> TraversalMetrics.graph_traversor
class	<a href="#">TraversalMetrics.node_test</a> TraversalMetrics.node_test
class	<a href="#">TraversalMetrics.traversor_degree_distribution</a> TraversalMetrics.traversor_degree_distribution

### Constructor Summary

public	<a href="#">TraversalMetrics()</a>
--------	------------------------------------

### Method Summary

static int	<a href="#">count(Graph graph, TraversalMetrics.node_test t)</a> Traverse the graph and count those nodes which are accepted by the node_test callback.
static int[]	<a href="#">degree_distribution(Graph graph)</a> Traverse the Graph and create a histogram of the distribution of nodal out-degree.
static int	<a href="#">edges(Graph graph)</a> Traverse the graph and count the number of edges.
static <a href="#">dmap</a>	<a href="#">FloydWarshal(Graph graph)</a>
static float[]	<a href="#">norm_degree_distribution(Graph graph)</a> Traverse the Graph and create a histogram of the normalised distribution of nodal out-degree.
static java.util.Map	<a href="#">stats(dmap dmap, Graph graph)</a>
static java.util.Map	<a href="#">stats(Graph graph)</a>
static void	<a href="#">traverse_graph(Graph graph, TraversalMetrics.graph_traversor gt)</a> Simple convenience function to traverse a graph with the given traversor.
static void	<a href="#">traverse_graph(Graph graph, TraversalMetrics.graph_traversor[] gts)</a> Simple convenience function to traverse a graph for each of the given traversors

**Methods inherited from class** java.lang.Object

```
clone, equals, finalize, getClass, hashCode, notify, notifyAll, toString, wait, wait,
```

## Constructors

### TraversalMetrics

```
public TraversalMetrics()
```

## Methods

### traverse\_graph

```
public static void traverse_graph(Graph graph,
    TraversalMetrics.graph_traversor gt)
```

Simple convenience function to traverse a graph with the given traversor.

**Parameters:**

graph  
gt

### traverse\_graph

```
public static void traverse_graph(Graph graph,
    TraversalMetrics.graph_traversor[] gts)
```

Simple convenience function to traverse a graph for each of the given traversors

### degree\_distribution

```
public static int[] degree_distribution(Graph graph)
```

Traverse the Graph and create a histogram of the distribution of nodal out-degree.

**Parameters:**

graph - The graph to traverse

**Returns:**

An integer array of the degree distribution, where the array indices correspond to the degree.

### norm\_degree\_distribution

```
public static float[] norm_degree_distribution(Graph graph)
```

Traverse the Graph and create a histogram of the normalised distribution of nodal out-degree. I.e. the probability of a given degree, for that degree.

**Parameters:**

graph - The graph to traverse

**Returns:**

(continued from last page)

An integer array of the degree distribution, where the array indices correspond to the degree.

---

## count

```
public static int count(Graph graph,  
                      TraversalMetrics.node\_test t)
```

Traverse the graph and count those nodes which are accepted by the node\_test callback.

### Parameters:

graph - The graph to traverse

t - The boolean callback to apply to decide whether a node should be counted or not.

### Returns:

The number of nodes which match the provided test.

---

## edges

```
public static int edges(Graph graph)
```

Traverse the graph and count the number of edges.

---

## FloydWarshal

```
public static dmap FloydWarshal(Graph graph)
```

---

## stats

```
public static java.util.Map stats(Graph graph)
```

---

## stats

```
public static java.util.Map stats(dmap dmap,  
                                 Graph graph)
```

## org.nongnu.multigraph.metrics Interface TraversalMetrics.graph\_traversor

All Known Implementing Classes:  
[traversor\\_degree\\_distribution](#)

---

public interface TraversalMetrics.graph\_traversor  
extends

action callback interface, for each node of a graph

---

### Method Summary

abstract void	<a href="#">node</a> (java.lang.Object node)
---------------	--

### Methods

#### node

public abstract void **node**(java.lang.Object node)

## org.nongnu.multigraph.metrics Class TraversalMetrics.traversor\_degree\_distribution

```
java.lang.Object
+-org.nongnu.multigraph.metrics.TraversalMetrics.traversor_degree_distribution
```

All Implemented Interfaces:

[graph\\_traversor](#)

---

```
public class TraversalMetrics.traversor_degree_distribution
extends java.lang.Object
implements graph\_traversor
```

### Constructor Summary

public	<a href="#">traversor_degree_distribution()</a>
--------	---

### Method Summary

void	<a href="#">node(java.lang.Object node)</a>
------	---

#### Methods inherited from class java.lang.Object

```
clone, equals, finalize, getClass, hashCode, notify, notifyAll, toString, wait, wait,
```

#### Methods inherited from interface [org.nongnu.multigraph.metrics.TraversalMetrics.graph\\_traversor](#)

<a href="#">node</a>
----------------------

### Constructors

#### traversor\_degree\_distribution

```
public traversor\_degree\_distribution\(\)
```

### Methods

#### node

```
public void node\(java.lang.Object node\)
```

## org.nongnu.multigraph.metrics Interface TraversalMetrics.node\_test

---

public interface **TraversalMetrics.node\_test**  
extends

### Method Summary

abstract boolean	<a href="#"><u>test</u></a> (java.lang.Object node)
------------------	---

### Methods

#### **test**

public abstract boolean **test**(java.lang.Object node)

---

**Package**

## **org.nongnu.multigraph.perturb**

# org.nongnu.multigraph.perturb

## Interface perturber

All Known Implementing Classes:  
[RemoveAddEach](#), [RandomRemove](#)

### public interface perturber

extends

Operations common to graph perturbation implementations. These are used to iteratively apply changes to a graph according to some cohesive policy, provided by a concrete implementation.

## Method Summary

abstract int	<a href="#"><u>clear_removed_edges()</u></a> Clear state held for any removed edges.
abstract java.util.List	<a href="#"><u>perturb()</u></a> Carry out 1 iteration of the perturbation of the graph, according to the policy of the implementation.
abstract java.util.List	<a href="#"><u>removed_edges()</u></a>
abstract int	<a href="#"><u>restore()</u></a> Restore edges removed previously, up until the last call to <a href="#"><u>clear_removed_edges()</u></a> .

## Methods

### **perturb**

```
public abstract java.util.List perturb()
```

Carry out 1 iteration of the perturbation of the graph, according to the policy of the implementation. Edges may be added or removed.

**Returns:**

An unmodifiable list of any edges removed in this iteration.

### **restore**

```
public abstract int restore()
```

Restore edges removed previously, up until the last call to [clear\\_removed\\_edges\(\)](#). This also clears the remember, removed edges.

**Returns:**

The number of restored edges.

### **removed\_edges**

```
public abstract java.util.List removed_edges()
```

(continued from last page)

**Returns:**

An unmodifiable list view of any remembered, removed edges.

---

**clear\_removed\_edges**

```
public abstract int clear_removed_edges()
```

Clear state held for any removed edges. Future [restore\(\)](#) calls will not be able to add back any of the edges held before the call to this method.

**Returns:**

The number of edges cleared.

## org.nongnu.multigraph.perturb Class RandomMove

```
java.lang.Object
  +-org.nongnu.multigraph.rewire.Rewire
    +-org.nongnu.multigraph.perturb.RandomMove
```

**public class RandomMove**  
**extends Rewire**

A graph perturbing class which emulates a mobile network, by moving PositionableNode nodes according to their velocity, within the giving bounding area, and then applying cartesian rewire. Nodes that reach the bound have their velocity changed by 180° ± a random amount in a Gaussian distribution. This perturb class is intended to be called in a loop.

**Parameters:**

N, L

**Fields inherited from class org.nongnu.multigraph.rewire.Rewire**

el, graph

### Constructor Summary

public	<code>RandomMove(Graph graph, EdgeLabeler el, java.awt.Dimension bound, float default_speed, float maxrange)</code>
public	<code>RandomMove(Graph graph, EdgeLabeler el, java.awt.Dimension bound, float default_speed, float maxrange, int angle_dev)</code>

### Method Summary

void	<code>rewire()</code>
------	-----------------------

**Methods inherited from class org.nongnu.multigraph.rewire.Rewire**

add, factory, isaRewire, rewire

**Methods inherited from class java.lang.Object**

`clone`, `equals`, `finalize`, `getClass`, `hashCode`, `notify`, `notifyAll`, `toString`, `wait`, `wait`

### Constructors

#### RandomMove

```
public RandomMove(Graph graph,
                  EdgeLabeler el,
                  java.awt.Dimension bound,
                  float default_speed,
                  float maxrange)
```

(continued from last page)

---

## RandomMove

```
public RandomMove(Graph graph,
                  EdgeLabeler el,
                  java.awt.Dimension bound,
                  float default_speed,
                  float maxrange,
                  int angle_dev)
```

## Methods

### rewire

```
public void rewire()
```

Rewire the whole graph. All edges potentially are first cleared. Then some edges added back, according to the specified algorithm.

## org.nongnu.multigraph.perturb

### Class RandomRemove

```
java.lang.Object
+-org.nongnu.multigraph.perturb.RandomRemove
```

#### All Implemented Interfaces:

[perturber](#)

```
public class RandomRemove
extends java.lang.Object
implements perturber
```

Perturb the graph by removing a randomly chosen set of edges from the graph. Methods are also given to restore removed edges, and clear the state kept for removed edges.

## Constructor Summary

public	<a href="#">RandomRemove(Graph graph)</a> Create new instance to randomly remove 1 edge on each call to
public	<a href="#">RandomRemove(Graph graph, float remove_edges)</a> Create new instance to iterate over the specified graph and remove then add up to remove_edges # of edges on each call to <a href="#">remove()</a> ,
public	<a href="#">RandomRemove(Graph graph, float remove_edges, int max_perturbs)</a> Create new instance to iterate over the specified graph and remove then add up to remove_edges # of edges on each call to <a href="#">remove()</a> ,

## Method Summary

int	<a href="#">clear_removed_edges()</a> Clear state held for the removed edges.
boolean	<a href="#">combine()</a>
void	<a href="#">combine(boolean combine)</a>
java.util.List	<a href="#">perturb()</a> Carry out another iteration of random-removes by restoring any previously removed edges, clearing the removed_edge set, and then removing the next random set of edges.
java.util.List	<a href="#">remove()</a> Randomly remove edges from the graph, up to the specified number of edges for the class.
java.util.List	<a href="#">removed_edges()</a>
int	<a href="#">restore()</a> Restore edges removed previously, up until the last call to <a href="#">clear_removed_edges()</a> .

### Methods inherited from class java.lang.Object

```
clone, equals, finalize, getClass, hashCode, notify, notifyAll, toString, wait, wait,
wait
```

#### Methods inherited from interface [org.nongnu.multigraph.perturb.Perturber](#)

```
clear\_removed\_edges, perturb, removed\_edges, restore
```

## Constructors

### RandomRemove

```
public RandomRemove(Graph graph)
```

Create new instance to randomly remove 1 edge on each call to

#### Parameters:

graph - The graph to act on.

### RandomRemove

```
public RandomRemove(Graph graph,
                    float remove_edges)
```

Create new instance to iterate over the specified graph and remove then add up to remove\_edges # of edges on each call to [remove\(\)](#),

#### Parameters:

graph - The graph to act on

remove\_edges - The number of edges to remove/add at a time, on each call to remove. This may be specified either as an absolute number of edges ( $> 1$ ), or else as a proportion of the number of nodes, ( $0 < x < 1$ ).

### RandomRemove

```
public RandomRemove(Graph graph,
                    float remove_edges,
                    int max_perturbs)
```

Create new instance to iterate over the specified graph and remove then add up to remove\_edges # of edges on each call to [remove\(\)](#),

#### Parameters:

graph - The graph to act on

remove\_edges - The number of edges to remove/add at a time, on each call to remove. This may be specified either as an absolute number of edges ( $> 1$ ), or else as a proportion of the number of nodes, ( $0 < x < 1$ ).

max\_perturbs - Limit the number of times groups of edges will be removed. A value  $\leq 0$  means no limit.

## Methods

### combine

```
public void combine(boolean combine)
```

(continued from last page)

## combine

```
public boolean combine()
```

---

## remove

```
public java.util.List remove()
```

Randomly remove edges from the graph, up to the specified number of edges for the class.

### Returns:

An unmodifiable list of the edges removed in this particular iteration.

---

## restore

```
public int restore()
```

Restore edges removed previously, up until the last call to [clear\\_removed\\_edges\(\)](#).

---

## removed\_edges

```
public java.util.List removed_edges()
```

### Returns:

An unmodifiable list view of the remembered, removed edges.

---

## clear\_removed\_edges

```
public int clear_removed_edges()
```

Clear state held for the removed edges. Future [restore\(\)](#) calls will not be able to add back any of the edges held before the call to this method.

---

## perturb

```
public java.util.List perturb()
```

Carry out another iteration of random-removes by restoring any previously removed edges, clearing the removed\_edge set, and then removing the next random set of edges.

### Returns:

## org.nongnu.multigraph.perturb Class RemoveAddEach

```
java.lang.Object
+-org.nongnu.multigraph.perturb.RemoveAddEach
```

### All Implemented Interfaces:

[perturber](#)

public class **RemoveAddEach**

extends java.lang.Object

implements [perturber](#)

Perturb the graph by removing and adding back a subset of edges from the graph, on each call to [perturb\(\)](#), until all the edges have been removed and added back.

### Constructor Summary

public	<a href="#">RemoveAddEach(Graph graph)</a> Create new instance to iterate over the specified graph and remove then add 1 edge on each call to perturb, until all edges have been removed and added.
public	<a href="#">RemoveAddEach(Graph graph, float remove_edges, int max_perturbs)</a> Create new instance to iterate over the specified graph and remove then add up to remove_edges # of edges on each call to perturb, up to a maximum number of perturbations.

### Method Summary

int	<a href="#">clear_removed_edges()</a>
java.util.List	<a href="#">last_removed_edges()</a>
java.util.List	<a href="#">perturb()</a> Carry out 1 perturbation of the graph, either removing the next group of edges from the graph, or adding back the group of edges removed in the previous iteration.
java.util.List	<a href="#">removed_edges()</a>
int	<a href="#">restore()</a>

#### Methods inherited from class java.lang.Object

[clone](#), [equals](#), [finalize](#), [getClass](#), [hashCode](#), [notify](#), [notifyAll](#), [toString](#), [wait](#), [wait](#)

#### Methods inherited from interface org.nongnu.multigraph.perturb.perturber

[clear\\_removed\\_edges](#), [perturb](#), [removed\\_edges](#), [restore](#)

(continued from last page)

## Constructors

### RemoveAddEach

```
public RemoveAddEach(Graph graph)
```

Create new instance to iterate over the specified graph and remove then add 1 edge on each call to perturb, until all edges have been removed and added.

**Parameters:**

graph - The graph to act on.

### RemoveAddEach

```
public RemoveAddEach(Graph graph,  
                     float remove_edges,  
                     int max_perturbs)
```

Create new instance to iterate over the specified graph and remove then add up to remove\_edges # of edges on each call to perturb, up to a maximum number of perturbations.

**Parameters:**

graph - The graph to act on

remove\_edges - The number of edges to remove/add at a time, on each call to perturb. This may be specified either as an absolute number of edges ( $> 1$ ), or else as a proportion of the number of nodes, ( $0 < x < 1$ ).

max\_perturbs - Limit the number of times groups of edges will be removed. A value  $\leq 0$  means no limit.

## Methods

### last\_removed\_edges

```
public java.util.List last_removed_edges()
```

### perturb

```
public java.util.List perturb()
```

Carry out 1 perturbation of the graph, either removing the next group of edges from the graph, or adding back the group of edges removed in the previous iteration.

**Returns:**

An unmodifiable list of the edges removed in this iteration.

### restore

```
public int restore()
```

### removed\_edges

```
public java.util.List removed_edges()
```

(continued from last page)

## **clear\_removed\_edges**

```
public int clear_removed_edges()
```

---

**Package**

## **org.nongnu.multigraph.rewire**

## org.nongnu.multigraph.rewire Class CartesianRewire

```
java.lang.Object
+-org.nongnu.multigraph.rewire.Rewire
  +-org.nongnu.multigraph.rewire.CartesianRewire
```

### public class **CartesianRewire** extends [Rewire](#)

Wire up nodes in the graph with each other according to their cartesian distance from each other, applying a 'range' constraint. The user can apply further constraints using the EdgeLabeler.

This rewiring algorithm only considers edges once, at this time. If the graph is directed, and the user wishes both directions to be set, they must do so themselves in their EdgeLabeler.

#### Parameters:

N - The type of the Nodes in the graph, E - The type of the Edges in the graph

#### Fields inherited from class [org.nongnu.multigraph.rewire.Rewire](#)

[el](#), [graph](#)

## Constructor Summary

public	<a href="#">CartesianRewire(Graph graph, EdgeLabeler el, java.awt.Dimension bound, float range)</a> Create a new CartesianRewire instance, for the given graph, wiring up nodes that are within the given distance.
public	<a href="#">CartesianRewire(Graph graph, EdgeLabeler el, float range)</a> Create a new CartesianRewire instance, for the given graph, wiring up nodes that are within the given distance.

## Method Summary

void	<a href="#">rewire()</a>
------	--------------------------

#### Methods inherited from class [org.nongnu.multigraph.rewire.Rewire](#)

[add](#), [factory](#), [isaRewire](#), [rewire](#)

#### Methods inherited from class [java.lang.Object](#)

[clone](#), [equals](#), [finalize](#), [getClass](#), [hashCode](#), [notify](#), [notifyAll](#), [toString](#), [wait](#), [wait](#)

## Constructors

(continued from last page)

## CartesianRewire

```
public CartesianRewire(Graph graph,
                      EdgeLabeler el,
                      java.awt.Dimension bound,
                      float range)
```

Create a new CartesianRewire instance, for the given graph, wiring up nodes that are within the given distance. Note that the EdgeLabeler callback may apply its own, further constraints, by returning a null label. This instance of the algorithm uses the supplied boundary to create a grid index to speed things. Certain extreme cases may be slower with grid-indexing e.g. where nodes are extremely tightly bunched, positionally.

### Parameters:

- graph - The graph to rewire.
- el - The EdgeLabeler to callback to create labels.
- bound - The positional boundary for nodes, used for grid-indexing.
- range - The maximum range for links between nodes.

## CartesianRewire

```
public CartesianRewire(Graph graph,
                      EdgeLabeler el,
                      float range)
```

Create a new CartesianRewire instance, for the given graph, wiring up nodes that are within the given distance. Note that the EdgeLabeler callback may apply its own, further constraints, by returning a null label.

### Parameters:

- graph - The graph to rewire.
- el - The EdgeLabeler to callback to create labels.
- range - The maximum range for links between nodes.

## Methods

### rewire

```
public void rewire()
```

Rewire the whole graph. All edges potentially are first cleared. Then some edges added back, according to the specified algorithm.

## org.nongnu.multigraph.rewire Class LatticeRewire

```
java.lang.Object
+-org.nongnu.multigraph.rewire.Rewire
  +-org.nongnu.multigraph.rewire.LatticeRewire
```

**public class LatticeRewire**  
**extends Rewire**

Wire up the nodes in a 2D lattice. The algorithm defaults to floor ( $\sqrt{|V|}$ ) columns, unless a columns value greater than 0 is specified.

**Parameters:**

N - The type of the nodes of the graph., E - The type of the edge labels of the graph.

**Fields inherited from class org.nongnu.multigraph.rewire.Rewire**

el, graph

### Constructor Summary

public	<u>LatticeRewire</u> ( <u>Graph</u> graph, <u>EdgeLabeler</u> el, int cols)
public	<u>LatticeRewire</u> ( <u>Graph</u> graph, <u>EdgeLabeler</u> el)

### Method Summary

void	<u>rewire</u> ()
------	------------------

**Methods inherited from class org.nongnu.multigraph.rewire.Rewire**

add, factory, isaRewire, rewire

**Methods inherited from class java.lang.Object**

clone, equals, finalize, getClass, hashCode, notify, notifyAll, toString, wait, wait, wait

### Constructors

#### LatticeRewire

```
public LatticeRewire(Graph graph,
                     EdgeLabeler el,
                     int cols)
```

(continued from last page)

## LatticeRewire

```
public LatticeRewire(Graph graph,  
                     EdgeLabeler el)
```

## Methods

### rewire

```
public void rewire()
```

Rewire the whole graph. All edges potentially are first cleared. Then some edges added back, according to the specified algorithm.

## org.nongnu.multigraph.rewire Class MultiClassScaleFreeRewire

```
java.lang.Object
  +-org.nongnu.multigraph.rewire.Rewire
    +-org.nongnu.multigraph.rewire.ScaleFreeRewire
      +-org.nongnu.multigraph.rewire.MultiClassScaleFreeRewire
```

public class **MultiClassScaleFreeRewire**

extends [ScaleFreeRewire](#)

**Parameters:**

N - The label type of nodes in the graph., E - The label type of edges in the graph.

### Field Summary

protected	<a href="#">p</a>
-----------	-------------------

**Fields inherited from class** [org.nongnu.multigraph.rewire.ScaleFreeRewire](#)

<a href="#">a</a> , <a href="#">m</a> , <a href="#">nodes</a> , <a href="#">r</a>
---

**Fields inherited from class** [org.nongnu.multigraph.rewire.Rewire](#)

<a href="#">el</a> , <a href="#">graph</a>
--

### Constructor Summary

public	<a href="#">MultiClassScaleFreeRewire</a> ( <a href="#">Graph</a> graph, <a href="#">EdgeLabeler</a> el)
--------	--

### Method Summary

int	<a href="#">add_like_links</a> (int split, int numlinks)
boolean	<a href="#">consider_similar_link</a> (java.lang.Object vi, java.lang.Object vj, int numnodes, int numlinks)
int	<a href="#">p()</a>
<a href="#">MultiClassScaleFreeRewire</a>	<a href="#">p</a> (int p) The 'p' parameter is the number of links to add between existing, sufficiently alike nodes on each time-step.
int	<a href="#">rewire_callback</a> (int split, int numlinks)

**Methods inherited from class** [org.nongnu.multigraph.rewire.ScaleFreeRewire](#)

<a href="#">_init_nodes</a> , <a href="#">a</a> , <a href="#">a</a> , <a href="#">add_link</a> , <a href="#">add</a> , <a href="#">add</a> , <a href="#">consider_link</a> , <a href="#">link_added</a> , <a href="#">m_mode_stop</a> , <a href="#">m_mode</a> , <a href="#">m_mode</a> , <a href="#">m_mode</a> , <a href="#">m</a> , <a href="#">m</a> , <a href="#">m0</a> , <a href="#">nodes_iterator</a> , <a href="#">rewire_callback</a> , <a href="#">rewire</a>
---

**Methods inherited from class** [org.nongnu.multigraph.rewire.Rewire](#)[add](#), [factory](#), [isaRewire](#), [rewire](#)**Methods inherited from class** [java.lang.Object](#)[clone](#), [equals](#), [finalize](#), [getClass](#), [hashCode](#), [notify](#), [notifyAll](#), [toString](#), [wait](#), [wait](#)

## Fields

**p**

protected int p

## Constructors

### MultiClassScaleFreeRewire

public [MultiClassScaleFreeRewire](#)([Graph](#) graph,  
[EdgeLabeler](#) el)

## Methods

**p**

public int p()

**Returns:**

The number of links to consider adding between alike nodes on each time-step.

**See Also:**[p\(int\)](#)**p**public [MultiClassScaleFreeRewire](#) p(int p)

The 'p' parameter is the number of links to add between existing, sufficiently alike nodes on each time-step. It defaults to 1, and must be greater than or equal to 0.

**Parameters:**

p - The number of links to consider adding between alike nodes on each time-step.

**Returns:**

reference to this class.

(continued from last page)

## **consider\_similar\_link**

```
protected boolean consider_similar_link(java.lang.Object vi,
    java.lang.Object vj,
    int numnodes,
    int numlinks)
```

---

## **add\_like\_links**

```
protected int add_like_links(int split,
    int numlinks)
```

---

## **rewire\_callback**

```
protected int rewire_callback(int split,
    int numlinks)
```

## org.nongnu.multigraph.rewire Class RandomRewire

```
java.lang.Object
  +-org.nongnu.multigraph.rewire.Rewire
    +-org.nongnu.multigraph.rewire.RandomRewire
```

**public class RandomRewire**  
**extends Rewire**

Randomly wire up nodes of a graph, with each node having at least the mindegree number of outgoing edges. Note that the graph need not be continuous.

**Parameters:**

N - The type of the Nodes in the graph, E - The type of the Edges in the graph

**Fields inherited from class org.nongnu.multigraph.rewire.Rewire**

el, graph

### Constructor Summary

public	<u>RandomRewire</u> ( <u>Graph</u> graph, <u>EdgeLabeler</u> el)
public	<u>RandomRewire</u> ( <u>Graph</u> graph, <u>EdgeLabeler</u> el, int mindegree) Create a RandomRewire graph rewirer, with a minimum out-degree which nodes should have after the graph is rewired.

### Method Summary

void	<u>rewire</u> ()
<u>RandomRewire</u>	<u>set_mindegree</u> (int mindegree) Set the minimum out-degree which nodes should have after the graph is rewired.

**Methods inherited from class org.nongnu.multigraph.rewire.Rewire**

add, factory, isaRewire, rewire

**Methods inherited from class java.lang.Object**

clone, equals, finalize, getClass, hashCode, notify, notifyAll, toString, wait, wait, wait

### Constructors

#### RandomRewire

```
public RandomRewire(Graph graph,  
                    EdgeLabeler el)
```

## RandomRewire

```
public RandomRewire(Graph graph,  
                    EdgeLabeler el,  
                    int mindegree)
```

Create a RandomRewire graph rewirer, with a minimum out-degree which nodes should have after the graph is rewired.

### Parameters:

- graph - The graph to rewire.
- el - An EdgeLabeler callback, to allow the user to create Labels for new Edges.
- mindegree - The minimum out degree (edges to other nodes)

## Methods

### set\_mindegree

```
public RandomRewire set_mindegree(int mindegree)
```

Set the minimum out-degree which nodes should have after the graph is rewired.

### Parameters:

- mindegree - The minimum out degree (edges to other nodes)

### Returns:

This RandomRewire instance.

---

### rewire

```
public void rewire()
```

Rewire the whole graph. All edges potentially are first cleared. Then some edges added back, according to the specified algorithm.

# org.nongnu.multigraph.rewire

## Class Rewire

```
java.lang.Object
+-org.nongnu.multigraph.rewire.Rewire
```

### Direct Known Subclasses:

[ScaleFreeRewire](#), [RandomRewire](#), [LatticeRewire](#), [CartesianRewire](#), [RandomMove](#)

public abstract class **Rewire**  
extends java.lang.Object

Abstract interface for algorithms to rewire the edges of a graph. All edges may be initially cleared from the graph.

#### Parameters:

N - The type of the Nodes in the graph, E - The type of the Edges in the graph

## Field Summary

protected	<a href="#">el</a>
protected	<a href="#">graph</a>

## Constructor Summary

public	<a href="#">Rewire(Graph graph, EdgeLabeler el)</a>
--------	---

## Method Summary

void	<a href="#">add(java.lang.Object node)</a> Add a single node to the graph.
static <a href="#">Rewire</a>	<a href="#">factory(java.lang.String alname, Graph graph, EdgeLabeler el)</a>
static boolean	<a href="#">isaRewire(java.lang.String name)</a> Query whether the given string is an implemented rewire algorithm.
abstract void	<a href="#">rewire()</a> Rewire the whole graph.

## Methods inherited from class java.lang.Object

`clone`, `equals`, `finalize`, `getClass`, `hashCode`, `notify`, `notifyAll`, `toString`, `wait`, `wait`

## Fields

### graph

protected org.nongnu.multigraph.Graph **graph**

(continued from last page)

---

**el**

```
protected org.nongnu.multigraph.EdgeLabeler el
```

## Constructors

### Rewire

```
public Rewire(Graph graph,  
             EdgeLabeler el)
```

## Methods

### rewire

```
public abstract void rewire()
```

Rewire the whole graph. All edges potentially are first cleared. Then some edges added back, according to the specified algorithm.

---

### add

```
public void add(java.lang.Object node)
```

Add a single node to the graph.

This may (or may not) have different resource usage relative to rewire(). E.g. it may use less memory than rewire(), but more CPU over all if called for a significant number of nodes.

This method is optional, and not all implementations support it.

**Parameters:**

node

**Throws:**

UnsupportedOperationException - If this method is not supported.

---

### isaRewire

```
public static boolean isaRewire(java.lang.String name)
```

Query whether the given string is an implemented rewire algorithm.

**Parameters:**

name - rewiring algorithm name

**Returns:**

Whether a rewiring implementation exists for the given algorithm name

(continued from last page)

**factory**

```
public final static Rewire factory(java.lang.String algname,  
        Graph graph,  
        EdgeLabeler el)
```

## org.nongnu.multigraph.rewire Class ScaleFreeRewire

```
java.lang.Object
+-org.nongnu.multigraph.rewire.Rewire
  +-org.nongnu.multigraph.rewire.ScaleFreeRewire
```

### Direct Known Subclasses:

[MultiClassScaleFreeRewire](#)

public class **ScaleFreeRewire**

extends [Rewire](#)

Rewire a graph such that the connectivity of its nodes have a scale-free distribution, following the model given in the Barabasi, Albert paper "Emergence of Scaling in Random Networks" paper.

The model uses an `m` parameter, which specifies the number of edges to add from each new node to existing nodes, on each time-step. The initial seed size of the graph, (`m_0` in the paper) will be taken as `m + 1`. The default for `m` is 1.

In the BA model, `m` is very rigidly the number of links added. This may be useful for analytical purposes, but does not lead to much variation in the structure of the generated graphs. Other interpretations are possible for `m`, which deviate slightly from the BA model. E.g. it could be taken as a minimum number of links to add, while considering adding links from the new node to every existing node; or as a maximum number to add. This can be controlled with the "`m_mode`" parameter (@see `m_modes`).

An additional '`a`' parameter allows an additive bias to be introduced, which gives low-degree (i.e. younger) nodes a better chance - in the spirit of the Dorogovtsev, et al, paper. @see `#a`

### Parameters:

`N` - The label type of nodes in the graph., `E` - The label type of edges in the graph.

## Nested Class Summary

class	<a href="#">ScaleFreeRewire.m_modes</a> ScaleFreeRewire.m_modes
-------	--

## Field Summary

protected	<a href="#">a</a>
protected	<a href="#">m</a>
protected	<a href="#">nodes</a>
protected	<a href="#">r</a>

## Fields inherited from class [org.nongnu.multigraph.rewire.Rewire](#)

[el](#), [graph](#)

## Constructor Summary

public	<a href="#">ScaleFreeRewire(Graph graph, EdgeLabeler el)</a> Create a ScaleFreeRewiring instancing, with the default m value.
--------	--

## Method Summary

void	<a href="#">_init_nodes()</a>
int	<a href="#">a()</a>
<a href="#">ScaleFreeRewire</a>	<a href="#">a(int a)</a> The 'a' parameter controls the initial attractiveness of a node, along the lines of the A parameter in the Dorogovtsev, et al, "Structure of Growing Networks with Preferential Linking", Phys.
boolean	<a href="#">add_link(java.lang.Object to_add, java.lang.Object to)</a>
void	<a href="#">add(java.lang.Object to_add)</a>
int	<a href="#">add(java.lang.Object to_add, long numlinks, java.lang.Iterable iter)</a>
boolean	<a href="#">consider_link(java.lang.Object to_add, java.lang.Object vi, long numlinks)</a> Consider whether to add a link between the new node and the given existing node.
void	<a href="#">link_added(java.lang.Object added, java.lang.Object vi)</a> Called when a link is created between 2 nodes.
static boolean	<a href="#">m_mode_stop(<a href="#">ScaleFreeRewire.m_modes</a> m_mode, int m, int added, int pass)</a>
<a href="#">ScaleFreeRewire.m_modes</a>	<a href="#">m_mode()</a>
<a href="#">ScaleFreeRewire</a>	<a href="#">m_mode(<a href="#">ScaleFreeRewire.m_modes</a> m_mode)</a>
<a href="#">ScaleFreeRewire</a>	<a href="#">m_mode(java.lang.String m_mode)</a>
int	<a href="#">m()</a>
<a href="#">ScaleFreeRewire</a>	<a href="#">m(int m)</a> The 'm' parameter is the number of links each new node will get to the existing nodes in the graph.
void	<a href="#">m0()</a>
java.util.Iterator	<a href="#">nodes_iterator(int split)</a>
int	<a href="#">rewire_callback(int split, int numlinks)</a>
void	<a href="#">rewire()</a>

### Methods inherited from class [org.nongnu.multigraph.rewire.Rewire](#)

[add](#), [factory](#), [isaRewire](#), [rewire](#)

**Methods inherited from class** java.lang.Object

```
clone, equals, finalize, getClass, hashCode, notify, notifyAll, toString, wait, wait,
```

**Fields****r**

```
protected java.util.Random r
```

**nodes**

```
protected java.lang.Object nodes
```

**m**

```
protected int m
```

**a**

```
protected int a
```

**Constructors****ScaleFreeRewire**

```
public ScaleFreeRewire(Graph graph,  
EdgeLabeler el)
```

Create a ScaleFreeRewiring instancing, with the default m value.

**Parameters:**

graph - Graph to act on

el - EdgeLabeler callback, whose getLabel method will be called when an edge is added.

**Methods****m**

```
public int m()
```

**m**

```
public ScaleFreeRewire m(int m)
```

(continued from last page)

The 'm' parameter is the number of links each new node will get to the existing nodes in the graph. It defaults to 1, and must be greater than or equal to 1.

**Parameters:**

`m` - The number of links to add from a new node on every time-step.

**Returns:**

reference to this class.

**a**

```
public int a()
```

**a**

```
public ScaleFreeRewire a(int a)
```

The 'a' parameter controls the initial attractiveness of a node, along the lines of the A parameter in the Dorogovtsev, et al, "Structure of Growing Networks with Preferential Linking", Phys. Rev. Lett., 2000, model. The default is 0, in which case the behaviour is identical to the BA model. The value must be  $\geq 0$ .

**Parameters:**

`a` - The initial attractiveness of new nodes.

**Returns:****m\_mode**

```
public ScaleFreeRewire.m\_modes m_mode()
```

**m\_mode**

```
public ScaleFreeRewire m_mode(ScaleFreeRewire.m\_modes m_mode)
```

**Parameters:**

`m_mode` - Whether to interpret `m` as a maximum, a minimum or strictly according to BA model, as a precise number of links to add.

**Returns:**

reference to this class.

**See Also:**

`m_modes`.

**m\_mode**

```
public ScaleFreeRewire m_mode(java.lang.String m_mode)
```

**Parameters:**

(continued from last page)

`m_mode` - Whether to interpret `m` as a maximum, a minimum or strictly according to BA model, as a precise number of links to add.

**Returns:**

reference to this class.

**See Also:**

`m_modes`.

**\_init\_nodes**

```
protected void _init_nodes()
```

**m\_mode\_stop**

```
protected static boolean m_mode_stop(ScaleFreeRewire.m\_modes m_mode,
    int m,
    int added,
    int pass)
```

**consider\_link**

```
protected boolean consider_link(java.lang.Object to_add,
    java.lang.Object vi,
    long numlinks)
```

Consider whether to add a link between the new node and the given existing node.

**Parameters:**

`to_add` - The vertex to be added to the graph  
`vi` - The vertex being considered, `v_i` in the paper.  
`numlinks` - The number of links that were in the graph, before any links were added to this new node.

**Returns:**

True if a link should be created

**link\_added**

```
protected void link_added(java.lang.Object added,
    java.lang.Object vi)
```

Called when a link is created between 2 nodes. To facilitate the maintenance of any state. For the standard BA-model, there is no state beside the number of links, used to calculate the sum of the degrees, which the class already tracks.

**m0**

```
protected void m0()
```

**nodes\_iterator**

```
protected java.util.Iterator nodes_iterator(int split)
```

## rewire\_callback

```
protected int rewire_callback(int split,  
    int numlinks)
```

---

## rewire

```
public void rewire()
```

Rewire the whole graph. All edges potentially are first cleared. Then some edges added back, according to the specified algorithm.

---

## add\_link

```
protected boolean add_link(java.lang.Object to_add,  
    java.lang.Object to)
```

---

## add

```
protected int add(java.lang.Object to_add,  
    long numlinks,  
    java.lang.Iterable iter)
```

---

## add

```
public void add(java.lang.Object to_add)
```

Add a single node to the graph.

This may (or may not) have different resource usage relative to rewire(). E.g. it may use less memory than rewire(), but more CPU over all if called for a significant number of nodes.

This method is optional, and not all implementations support it.

## org.nongnu.multigraph.rewire Class ScaleFreeRewire.m\_modes

```
java.lang.Object
  +-java.lang.Enum
    +-org.nongnu.multigraph.rewire.ScaleFreeRewire.m_modes
```

### All Implemented Interfaces:

java.io.Serializable, java.lang.Comparable

public static final class **ScaleFreeRewire.m\_modes**

extends java.lang.Enum

## Field Summary

public static final	<a href="#">MAX</a>	The m-value is interpreted as a maximum value.
public static final	<a href="#">MIN</a>	The m-value is interpreted as a minimum value.
public static final	<a href="#">STRICT</a>	Strict interpretation of the m-value used, according to the Barabasi-Albert model.

## Method Summary

static <a href="#">ScaleFreeRewire.m_mod es</a>	<a href="#">valueOf(java.lang.String name)</a>
static <a href="#">ScaleFreeRewire.m_mod es[]</a>	<a href="#">values()</a>

### Methods inherited from class java.lang.Enum

clone, compareTo, equals, finalize, getDeclaringClass, hashCode, name, ordinal, toString, valueOf

### Methods inherited from class java.lang.Object

clone, equals, finalize, getClass, hashCode, notify, notifyAll, toString, wait, wait

### Methods inherited from interface java.lang.Comparable

compareTo

## Fields

(continued from last page)

## STRICT

```
public static final org.nongnu.multigraph.rewire.ScaleFreeRewire.m_modes STRICT
```

Strict interpretation of the m-value used, according to the Barabasi-Albert model. I.e. exactly m links are added on each time-step. This means not all pairs of nodes may be considered.

## MIN

```
public static final org.nongnu.multigraph.rewire.ScaleFreeRewire.m_modes MIN
```

The m-value is interpreted as a minimum value. At every time-step, the creation of links will be considered between \*every\* existing node and the new node, AND at least m-links will be added.

## MAX

```
public static final org.nongnu.multigraph.rewire.ScaleFreeRewire.m_modes MAX
```

The m-value is interpreted as a maximum value. At every time-step, the creation of links will be considered between \*every\* existing node and the new node, OR at least m-links have been added. A minimum of 1 link is also enforced.

## Methods

### values

```
public static ScaleFreeRewire.m_modes[] values()
```

### valueOf

```
public static ScaleFreeRewire.m_modes valueOf(java.lang.String name)
```

---

**Package**

## **org.nongnu.multigraph.structure**

## org.nongnu.multigraph.structure Class graph\_diff

```
java.lang.Object
+-org.nongnu.multigraph.structure.graph_diff
```

---

```
public class graph_diff
extends java.lang.Object
```

Compare two graphs and call the given specified user actions accordingly.

### Nested Class Summary

class	<a href="#">graph_diff.change_state</a> graph_diff.change_state
class	<a href="#">graph_diff.edge_callback</a> graph_diff.edge_callback
class	<a href="#">graph_diff.node_callback</a> graph_diff.node_callback

### Constructor Summary

public	<a href="#">graph_diff(Graph old_graph, Graph new_graph, graph_diff.node_callback node_cb, graph_diff.edge_callback edge_cb)</a>
--------	--

### Method Summary

boolean	<a href="#">compare_next()</a>
void	<a href="#">compare()</a>

#### Methods inherited from class java.lang.Object

clone, equals, finalize, getClass, hashCode, notify, notifyAll, toString, wait, wait, wait
--

### Constructors

#### graph\_diff

```
public graph_diff(Graph old_graph,
                  Graph new_graph,
                  graph_diff.node_callback node_cb,
                  graph_diff.edge_callback edge_cb)
```

### Methods

(continued from last page)

## **compare\_next**

```
public boolean compare_next()
```

---

## **compare**

```
public void compare()
```

## org.nongnu.multigraph.structure Class graph\_diff.change\_state

```
java.lang.Object
  +-java.lang.Enum
    +-org.nongnu.multigraph.structure.graph_diff.change_state
```

### All Implemented Interfaces:

java.io.Serializable, java.lang.Comparable

public static final class **graph\_diff.change\_state**  
extends java.lang.Enum

## Field Summary

public static final	<a href="#">added</a>
public static final	<a href="#">removed</a>

## Method Summary

static <a href="#">graph_diff.change_state</a>	<a href="#">valueOf(java.lang.String name)</a>
static <a href="#">graph_diff.change_state[]</a>	<a href="#">values()</a>

### Methods inherited from class java.lang.Enum

clone, compareTo, equals, finalize, getDeclaringClass, hashCode, name, ordinal, toString, valueOf

### Methods inherited from class java.lang.Object

clone, equals, finalize, getClass, hashCode, notify, notifyAll, toString, wait, wait, wait

### Methods inherited from interface java.lang.Comparable

compareTo

## Fields

### added

public static final org.nongnu.multigraph.structure.graph\_diff.change\_state **added**

## removed

```
public static final org.nongnu.multigraph.structure.graph_diff.change_state removed
```

## Methods

### values

```
public static graph\_diff.change\_state\[\] values()
```

---

### valueOf

```
public static graph\_diff.change\_state valueOf(java.lang.String name)
```

## org.nongnu.multigraph.structure Interface graph\_diff.node\_callback

---

public interface **graph\_diff.node\_callback**  
extends

### Method Summary

abstract void	<a href="#"><u>action</u></a> (java.lang.Object node, <a href="#"><u>graph_diff.change_state</u></a> s)
---------------	---

### Methods

#### **action**

public abstract void **action**(java.lang.Object node,  
[graph\\_diff.change\\_state](#) s)

## org.nongnu.multigraph.structure Interface graph\_diff.edge\_callback

---

public interface graph\_diff.edge\_callback  
extends

### Method Summary

abstract void	<a href="#">action(Edge edge, graph_diff.change_state s)</a>
---------------	--

### Methods

#### action

public abstract void [action\(Edge edge,](#)  
[graph\\_diff.change\\_state s\)](#)

## org.nongnu.multigraph.structure Class kshell

```
java.lang.Object
+-org.nongnu.multigraph.structure.kshell
```

---

**public class kshell**  
extends java.lang.Object

Calculate the maximum k-core membership of each vertex, according to the k-core definition from Seidman, "Network structure and minimum degree", Social Networks, 1983.

### Constructor Summary

public	<a href="#">kshell()</a>
--------	--------------------------

### Method Summary

static int	<a href="#">calc(Graph graph)</a>
------------	-----------------------------------

Calculate the maximum k-shell membership for each node in the Graph, storing the result in the gkc().k field of the node.

### Methods inherited from class java.lang.Object

clone, equals, finalize, getClass, hashCode, notify, notifyAll, toString, wait, wait
--

---

### Constructors

#### kshell

public [kshell\(\)](#)

---

### Methods

#### calc

public static int [calc\(Graph graph\)](#)

Calculate the maximum k-shell membership for each node in the Graph, storing the result in the gkc().k field of the node.  
The method returns the maximum k-shell in the graph.

**Parameters:**

graph - The graph to act on.

**Returns:**

The k-value of the maximum k-shell present in the graph.

## org.nongnu.multigraph.structure Interface kshell\_node

---

public interface **kshell\_node**  
extends

### Method Summary

abstract <a href="#"><u>kshell_node_data</u></a>	<a href="#"><u>gkc()</u></a>
---	------------------------------

### Methods

#### **gkc**

public abstract [kshell\\_node\\_data](#) [gkc\(\)](#)

## org.nongnu.multigraph.structure Class kshell\_node\_data

```
java.lang.Object
+-org.nongnu.multigraph.structure.kshell_node_data
```

```
public class kshell_node_data
extends java.lang.Object
```

### Field Summary

public	<a href="#">k</a>
public	<a href="#">removed</a>

### Constructor Summary

public	<a href="#">kshell_node_data()</a>
--------	------------------------------------

### Method Summary

void	<a href="#">reset()</a>
------	-------------------------

#### Methods inherited from class java.lang.Object

```
clone, equals, finalize, getClass, hashCode, notify, notifyAll, toString, wait, wait,
```

### Fields

#### k

```
public int k
```

#### removed

```
public boolean removed
```

### Constructors

#### kshell\_node\_data

```
public kshell_node_data()
```

(continued from last page)

## Methods

### **reset**

```
public void reset( )
```