

API Documentation

API Documentation

December 10, 2015

Contents

Contents	1
1 Package datk	2
1.1 Modules	2
1.2 Variables	2
2 Package datk.core	3
2.1 Modules	3
2.2 Variables	3
3 Module datk.core.algs	4
3.1 Variables	4
3.2 Class LCR	4
3.2.1 Methods	4
3.3 Class AsyncLCR	5
3.3.1 Methods	5
3.4 Class FloodMax	6
3.4.1 Methods	6
3.5 Class SynchBFS	7
3.5.1 Methods	7
3.6 Class SynchBFSAck	8
3.6.1 Methods	8
3.7 Class SynchConvergecast	9
3.7.1 Methods	9
3.8 Class AsyncConvergecast	10
3.8.1 Methods	10
3.9 Class SynchConvergeHeight	11
3.9.1 Methods	11
3.10 Class AsyncConvergeHeight	12
3.10.1 Methods	12
3.11 Class SynchBroadcast	13
3.11.1 Methods	13
3.12 Class SynchLubyMIS	14
3.12.1 Methods	15
4 Module datk.core.distalgs	16
4.1 Variables	16

4.2	Class Message	16
4.2.1	Methods	16
4.3	Class Process	16
4.3.1	Methods	16
4.4	Class Network	17
4.4.1	Methods	17
4.5	Class Algorithm	18
4.5.1	Methods	18
4.6	Class Synchronous_Algorithm	19
4.6.1	Methods	19
4.7	Class Do_Nothing	20
4.7.1	Methods	20
4.8	Class Asynchronous_Algorithm	20
4.8.1	Methods	20
4.9	Class Compose	21
4.9.1	Methods	21
4.10	Class Chain	22
4.10.1	Methods	22
5	Module datk.core.imports	23
5.1	Variables	23
6	Module datk.core.networks	24
6.1	Variables	24
6.2	Class Unidirectional_Ring	24
6.2.1	Methods	24
6.3	Class Bidirectional_Ring	24
6.3.1	Methods	25
6.4	Class Unidirectional_Line	25
6.4.1	Methods	25
6.5	Class Bidirectional_Line	25
6.5.1	Methods	26
6.6	Class Complete_Graph	26
6.6.1	Methods	26
6.7	Class Random_Line_Network	26
6.7.1	Methods	27
7	Module datk.core.testers	28
7.1	Functions	28
7.2	Variables	28
8	Package datk.tests	29
8.1	Modules	29
8.2	Variables	29
9	Module datk.tests.helpers	30
9.1	Functions	30
9.2	Variables	30
10	Module datk.tests.networks_tests	31
10.1	Variables	31
11	Module datk.tests.tests	32

11.1 Functions	32
11.2 Variables	33

1 Package datk

1.1 Modules

- **core**: A Python Toolkit for Distributed Algorithms
(Section 2, p. 3)
 - **algs** (Section 3, p. 4)
 - **distalgs** (Section 4, p. 16)
 - **imports** (Section 5, p. 23)
 - **networks** (Section 6, p. 24)
 - **tester** (Section 7, p. 28)
- **tests** (Section 8, p. 29)
 - **helpers**: Helper functions for tests in tests.py
(Section 9, p. 30)
 - **networks_tests**: Network Test Suite
(Section 10, p. 31)
 - **tests**: Algorithm Test Suite
(Section 11, p. 32)

1.2 Variables

Name	Description
<code>__package__</code>	Value: None

2 Package datk.core

A Python Toolkit for Distributed Algorithms

Authors:

- Amin Manna <manna@mit.edu>
- Mayuri Sridhar <mayuri@mit.edu>

2.1 Modules

- **algs** (*Section 3, p. 4*)
- **distalgs** (*Section 4, p. 16*)
- **imports** (*Section 5, p. 23*)
- **networks** (*Section 6, p. 24*)
- **tester** (*Section 7, p. 28*)

2.2 Variables

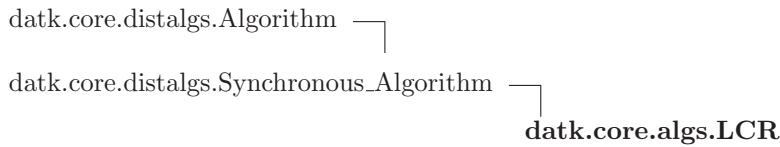
Name	Description
<code>--package--</code>	Value: None

3 Module *datk.core.algs*

3.1 Variables

Name	Description
<code>--package--</code>	Value: <code>'datk.core'</code>

3.2 Class LCR



The LeLann, Chang and Roberts algorithm for Leader Election in a Synchronous Ring Network

Each Process sends its identifier around the ring. When a Process receives an incoming identifier, it compares that identifier to its own. If the incoming identifier is greater than its own, it keeps passing the identifier; if it is less than its own, it discards the incoming identifier; if it is equal to its own, the Process declares itself the leader.

Requires:

- Every process knows `state['n']`, the size of the network

Effects:

- Every process has `state['status']` is 'leader' or 'non-leader'.
- Exactly one process has `state['status']` is 'leader'

3.2.1 Methods

msgs_i(*self*, *p*)

Overrides: `datk.core.distalgs.Algorithm.msgs_i`

trans_i(*self*, *p*, *msgs*)

Overrides: `datk.core.distalgs.Algorithm.trans_i`

cleanup_i(*self*, *p*)

Overrides: `datk.core.distalgs.Algorithm.cleanup_i`

Inherited from `datk.core.distalgs.Synchronous_Algorithm`(Section 4.6)

`execute()`, `msgs()`, `round()`, `run()`, `trans()`

Inherited from `datk.core.distalgs.Algorithm`(Section 4.5)

`__call__()`, `__init__()`, `cleanup()`, `count_msg()`, `delete()`, `get()`, `halt()`, `halt_i()`, `has()`, `increment()`, `set()`

3.3 Class AsyncLCR



The LeLann, Chang and Roberts algorithm for Leader Election in an Asynchronous Ring Network

Each Process sends its identifier around the ring. When a Process receives incoming identifier(s), it compares their largest to its own. If that incoming identifier is greater than its own, it keeps passing that identifier; if it is less than its own, it discards all the incoming identifiers; if it is equal to its own, the Process declares itself the leader. When a Process has declared itself Leader, it sends a Leader Declaration message around the ring, and halts. As it goes around the ring, each other Process outputs 'non-leader', and halts.

Requires:

- Every process knows `state['n']`, the size of the network

Effects:

- Every process has `state['status']` is 'leader' or 'non-leader'.
- Exactly one process has `state['status']` is 'leader'

3.3.1 Methods

`msgs_i(self, p, verbose=False)`

Overrides: `datk.core.distalgs.Algorithm.msgs_i`

`trans_i(self, p, verbose=False)`

Overrides: `datk.core.distalgs.Algorithm.trans_i`

`cleanup_i(self, p)`

Overrides: `datk.core.distalgs.Algorithm.cleanup_i`

Inherited from `datk.core.distalgs.Asynchronous_Algorithm`(Section 4.8)

`run()`, `run_process()`

Inherited from datk.core.distalgs.Algorithm(Section 4.5)

`__call__()`, `__init__()`, `cleanup()`, `count_msg()`, `delete()`, `get()`, `halt()`, `halt_i()`, `has()`, `increment()`, `set()`

3.4 Class FloodMax

UID flooding algorithm for Leader Election in a general network

Every process maintains a record of the maximum UID it has seen so far (initially its own). At each round, each process propagates this maximum on all of its outgoing edges. After `diam` rounds, if the maximum value seen is the process's own UID, the process elects itself the leader; otherwise, it is a non-leader.

Requires:

- Every process, `p`, has `p.state["diam"] >= dist(p, q)`, forall `q`.

3.4.1 Methods

`msgs_i(self, p)`

Overrides: `datk.core.distalgs.Algorithm.msgs_i`

`trans_i(self, p, msgs, verbose=False)`

Overrides: `datk.core.distalgs.Algorithm.trans_i`

`cleanup_i(self, p)`

Overrides: `datk.core.distalgs.Algorithm.cleanup_i`

Inherited from datk.core.distalgs.Synchronous_Algorithm(Section 4.6)

`execute()`, `msgs()`, `round()`, `run()`, `trans()`

Inherited from datk.core.distalgs.Algorithm(Section 4.5)

`__call__()`, `__init__()`, `cleanup()`, `count_msg()`, `delete()`, `get()`, `halt()`, `halt_i()`, `has()`, `increment()`, `set()`

3.5 Class *SynchBFS*



Constructs a BFS tree with the 'leader' Process at its root

At any point during execution, there is some set of processes that is "marked," initially just *i0*. Process *i0* sends out a search message at round 1, to all of its outgoing neighbors. At any round, if an unmarked process receives a search message, it marks itself and chooses one of the processes from which the search has arrived as its parent. At the first round after a process gets marked, it sends a search message to all of its outgoing neighbors.

Requires:

- `testLeaderElection`

Effects:

- every Process has `state['parent']`. Leader has `state['parent'] = None`

3.5.1 Methods

<code>is_i0(self, p)</code>

<code>msgs_i(self, p)</code>

Overrides: <code>datk.core.distalgs.Algorithm.msgs_i</code>

<code>trans_i(self, p, msgs)</code>

Overrides: <code>datk.core.distalgs.Algorithm.trans_i</code>
--

Inherited from `datk.core.distalgs.Synchronous_Algorithm`(Section 4.6)

`execute()`, `msgs()`, `round()`, `run()`, `trans()`

Inherited from `datk.core.distalgs.Algorithm`(Section 4.5)

`_call_()`, `_init_()`, `cleanup()`, `cleanup_i()`, `count_msg()`, `delete()`, `get()`, `halt()`, `halt_i()`, `has()`, `increment()`, `set()`

3.6 Class *SynchBFSAck*



Constructs a BFS tree with children pointers and the 'leader' Process at its root

Algorithm (Informal): At any point during execution, there is some set of processes that is "marked," initially just *i0*. Process *i0* sends out a search message at round 1, to all of its outgoing neighbors. At any round, if an unmarked process receives a search message, it marks itself and chooses one of the processes from which the search arrived as its parent. At the first round after a process gets marked, it sends a search message to all of its outgoing neighbors, and an acknowledgement to its parent, so that nodes will also know their children.

Requires:

- `testLeaderElection`

Effects:

- Every process knows:
 - `state['parent']`. Leader has `state['parent'] = None`
 - `state['children']`. Leaves have `state['children'] = []`

3.6.1 Methods

<code>is_i0(self, p)</code>

<code>msgs_i(self, p)</code>

Overrides: <code>datk.core.distalgs.Algorithm.msgs_i</code>

<code>trans_i(self, p, msgs)</code>

Overrides: <code>datk.core.distalgs.Algorithm.trans_i</code>
--

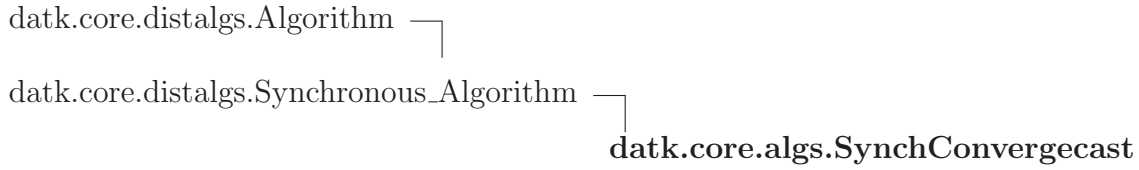
Inherited from `datk.core.distalgs.Synchronous_Algorithm`(Section 4.6)

`execute()`, `msgs()`, `round()`, `run()`, `trans()`

Inherited from `datk.core.distalgs.Algorithm`(Section 4.5)

`_call_()`, `_init_()`, `cleanup()`, `cleanup_i()`, `count_msg()`, `delete()`, `get()`, `halt()`, `halt_i()`, `has()`, `increment()`, `set()`

3.7 Class *SynchConvergecast*



Known Subclasses: *datk.core.algs.SynchConvergeHeight*

The abstract superclass of a class of Synchronous Algorithms that propagate information from the leaves of a BFS tree to its root.

Requires:

- Every Process knows state['parent']

#TODO If Processes also know state['children'] ==> Reduced Communication Complexity.

3.7.1 Methods

is_root(*self*, *p*)

msgs_i(*self*, *p*)

Overrides: *datk.core.distalgs.Algorithm.msgs_i*

trans_i(*self*, *p*, *msgs*)

Overrides: *datk.core.distalgs.Algorithm.trans_i*

cleanup_i(*self*, *p*)

Overrides: *datk.core.distalgs.Algorithm.cleanup_i*

trans_root(*self*, *p*, *msgs*)

output_root(*self*, *p*)

initial_msg_to_parent(*self*, *p*)

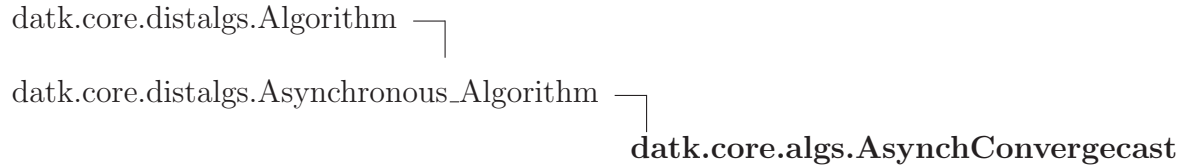
trans_msg_to_parent(*self*, *p*, *msgs*)

Inherited from datk.core.distalgs.Synchronous_Algorithm(Section 4.6)

execute(), *msgs()*, *round()*, *run()*, *trans()*

Inherited from datk.core.distalgs.Algorithm(Section 4.5)

`_call_()`, `_init_()`, `cleanup()`, `count_msg()`, `delete()`, `get()`, `halt()`, `halt_i()`, `has()`,
`increment()`, `set()`

3.8 Class *AsynchConvergecast*

Known Subclasses: `datk.core.algs.AsynchConvergeHeight`

The abstract superclass of a class of Synchronous Algorithms that propagate information from the leaves of a BFS tree to its root.

Requires:

- Every Process knows `state['parent']` and `state['children']`

3.8.1 Methods

<code>is_root(self, p)</code>

<code>msgs_i(self, p)</code>

Overrides: <code>datk.core.distalgs.Algorithm.msgs_i</code>

<code>trans_i(self, p, msgs)</code>
--

Overrides: <code>datk.core.distalgs.Algorithm.trans_i</code>
--

<code>cleanup_i(self, p)</code>
--

Overrides: <code>datk.core.distalgs.Algorithm.cleanup_i</code>
--

<code>trans_root(self, p, msgs)</code>

<code>outpout_root(self, p)</code>

<code>initial_msg_to_parent(self, p)</code>
--

trans_msg_to_parent (<i>self</i> , <i>p</i> , <i>msgs</i>)

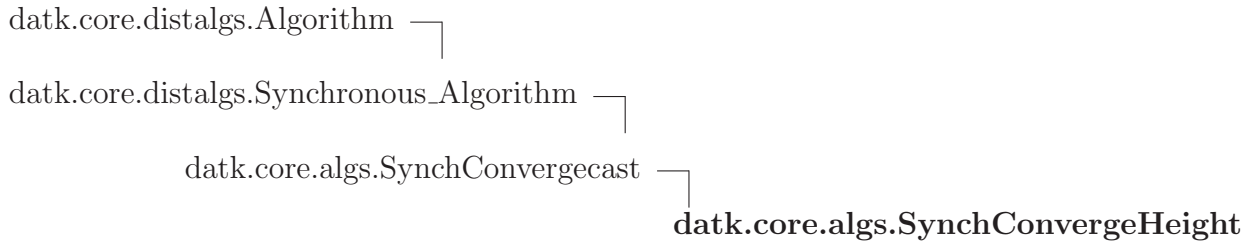
Inherited from datk.core.distalgs.Asynchronous_Algorithm(Section 4.8)

run(), run_process()

Inherited from datk.core.distalgs.Algorithm(Section 4.5)

__call__(), __init__(), cleanup(), count_msg(), delete(), get(), halt(), halt_i(), has(), increment(), set()

3.9 Class *SynchConvergeHeight*



Requires:

- BFS Tree

Effects:

- Root Process knows height of tree in state["height"]

3.9.1 Methods

cleanup_i (<i>self</i> , <i>p</i>)

Overrides: datk.core.distalgs.Algorithm.cleanup_i

initial_msg_to_parent (<i>self</i> , <i>p</i>)

Overrides: datk.core.algs.SynchConvergecast.initial_msg_to_parent

output_root (<i>self</i> , <i>p</i>)

trans_msg_to_parent (<i>self</i> , <i>p</i> , <i>msgs</i>)

Overrides: datk.core.algs.SynchConvergecast.trans_msg_to_parent

trans_root (<i>self</i> , <i>p</i> , <i>msgs</i>)
--

Overrides: datk.core.algs.SynchConvergecast.trans_root
--

Inherited from datk.core.algs.SynchConvergecast(Section 3.7)

is_root(), msgs_i(), output_root(), trans_i()

Inherited from datk.core.distalgs.Synchronous_Algorithm(Section 4.6)

execute(), msgs(), round(), run(), trans()

Inherited from datk.core.distalgs.Algorithm(Section 4.5)

__call__(), __init__(), cleanup(), count_msg(), delete(), get(), halt(), halt_i(), has(), increment(), set()

3.10 Class AsyncConvergeHeight

datk.core.distalgs.Algorithm └

datk.core.distalgs.Asynchronous_Algorithm └

datk.core.algs.AsyncConvergecast └

datk.core.algs.AsyncConvergeHeight

Requires:

- BFS Tree

Effects:

- Root Process knows height of tree in state["height"]

3.10.1 Methods

cleanup_i (<i>self</i> , <i>p</i>)

Overrides: datk.core.distalgs.Algorithm.cleanup_i

initial_msg_to_parent (<i>self</i> , <i>p</i>)

Overrides: datk.core.algs.AsyncConvergecast.initial_msg_to_parent

output_root (<i>self</i> , <i>p</i>)

trans_msg_to_parent (<i>self</i> , <i>p</i> , <i>msgs</i>)

Overrides: <i>datk.core.algs.AsyncConvergecast.trans_msg_to_parent</i>
--

trans_root (<i>self</i> , <i>p</i> , <i>msgs</i>)
--

Overrides: <i>datk.core.algs.AsyncConvergecast.trans_root</i>

Inherited from datk.core.algs.AsyncConvergecast(Section 3.8)

is_root(), *msgs_i()*, *outpout_root()*, *trans_i()*

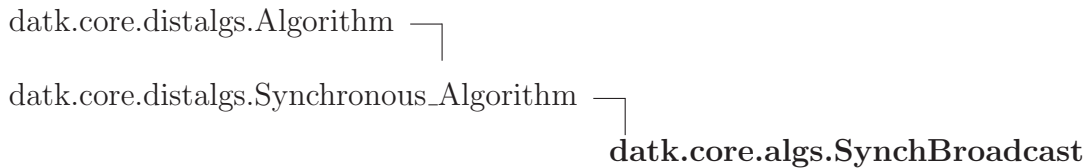
Inherited from datk.core.distalgs.Asynchronous_Algorithm(Section 4.8)

run(), *run_process()*

Inherited from datk.core.distalgs.Algorithm(Section 4.5)

call(), *_init_()*, *cleanup()*, *count_msg()*, *delete()*, *get()*, *halt()*, *halt_i()*, *has()*, *increment()*, *set()*

3.11 Class *SynchBroadcast*



Broadcasts a value stored in Process, *p*, to the BFS tree rooted at *p*

Requires:

- The attribute to be broadcasted must be specified in *self.params['attr']*
- BFS Tree with children pointers, where root node has *state[self.params['attr']]*

Effects:

- All Processes have *state[self.params['attr']] := the original value of in state[self.params['attr']]* of the root Process.

3.11.1 Methods

msgs_i (<i>self</i> , <i>p</i>)
--

Overrides: <i>datk.core.distalgs.Algorithm.msgs_i</i>

```
trans_i(self, p, msgs)
```

Overrides: `datk.core.distalgs.Algorithm.trans_i`

```
cleanup_i(self, p)
```

Overrides: `datk.core.distalgs.Algorithm.cleanup_i`

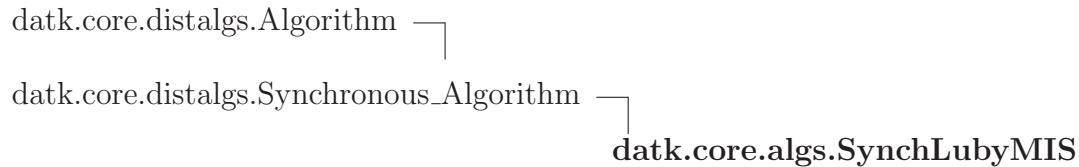
Inherited from `datk.core.distalgs.Synchronous_Algorithm`(Section 4.6)

`execute()`, `msgs()`, `round()`, `run()`, `trans()`

Inherited from `datk.core.distalgs.Algorithm`(Section 4.5)

`__call__()`, `__init__()`, `cleanup()`, `count_msg()`, `delete()`, `get()`, `halt()`, `halt_i()`, `has()`, `increment()`, `set()`

3.12 Class *SynchLubyMIS*



A randomized algorithm that constructs a Maximal Independent Set

The algorithm works in stages, each consisting of three rounds.

- Round 1: In the first round of a stage, the processes choose their respective vals and send them to their neighbors. By the end of round 1, when all the val messages have been received, the winners—that is, the processes in F —know who they are.
- Round 2: In the second round, the winners notify their neighbors. By the end of round 2, the losers—that is, the processes having neighbors in F —know who they are.
- Round 3: In the third round, each loser notifies its neighbors. Then all the involved processes—the winners, the losers, and the losers’ neighbors—remove the appropriate nodes and edges from the graph. More precisely, this means the winners and losers discontinue participation after this stage, and the losers’ neighbors remove all the edges that are incident on the newly removed nodes.

Requires:

- Every process knows `state['n']`, the size of the network

Effect:

- Every process knows `state['MIS']`. A boolean representing whether it is a member of the Maximal Independent Set found by Luby’s algorithm.

3.12.1 Methods

msgs_i (<i>self</i> , <i>p</i>)
--

Overrides: <i>datk.core.distalgs.Algorithm.msgs_i</i>

trans_i (<i>self</i> , <i>p</i> , <i>msgs</i>)

Overrides: <i>datk.core.distalgs.Algorithm.trans_i</i>
--

Inherited from *datk.core.distalgs.Synchronous_Algorithm*(Section 4.6)

execute(), *msgs()*, *round()*, *run()*, *trans()*

Inherited from *datk.core.distalgs.Algorithm*(Section 4.5)

__call__(), *__init__()*, *cleanup()*, *cleanup_i()*, *count_msg()*, *delete()*, *get()*, *halt()*,
halt_i(), *has()*, *increment()*, *set()*

4 Module `datk.core.distalgs`

4.1 Variables

Name	Description
<code>__package__</code>	Value: <code>'datk.core'</code>

4.2 Class Message

Known Subclasses: `datk.core.algs.AsyncLCR.Leader_Declaration`, `datk.core.algs.SynchBFS.Search`, `datk.core.algs.SynchBFSAck.AckParent`, `datk.core.algs.SynchBFSAck.Search`

4.2.1 Methods

```
__init__(self, algorithm, content=None)
```

```
__str__(self)
```

4.3 Class Process

A computing element located at a node of a directed network graph. Processes are identical except for their UID

4.3.1 Methods

```
__init__(self, UID, state=None, in_nbrs=[], out_nbrs=[])
```

```
link_to(self, new_out_nbr)
```

```
bi_link(self, nbr)
```

```
output(self, key, val, silent=False)
```

```
send_to_all_neighbors(self, msg)
```

```
send_msg(self, msg, out_nbrs=None)
```

```
get_msgs(self, algorithm, in_nbrs=None)
```

```
add(self, algorithm)
```

```
terminate(self, algorithm)
```

```
__str__(self)
```

```
__repr__(self)
```

4.4 Class Network

Known Subclasses: `datk.core.networks.Bidirectional_Line`, `datk.core.networks.Bidirectional_Ring`, `datk.core.networks.Complete_Graph`, `datk.core.networks.Random_Line_Network`, `datk.core.networks.Unidirectional_Line`, `datk.core.networks.Unidirectional_Ring`

A collection of Processes that know `n`, the # of processes. Known subclasses in `networkd.py`

4.4.1 Methods

```
__getitem__(self, i)
```

```
__init__(self, n, index_to_UID=None)
```

Creates a network of `n` disconnected Processes, with random distinct UUIDs, or as specified by the `index_to_UID` function

```
__iter__(self)
```

```
__len__(self)
```

```
__repr__(self)
```

```
add(self, algorithm)
```

```
clone(self)
```

```
draw(self)
```

```
index(self, p)
```

```
run(self, algorithm)
```

```
state(self)
```

4.5 Class Algorithm

Known Subclasses: `datk.core.distalgs.Asynchronous_Algorithm`, `datk.core.distalgs.Chain`, `datk.core.distalgs.Synchronous_Algorithm`

Abstract superclass for a distributed algorithm.

4.5.1 Methods

```
__init__(self, network=None, params={'draw': False, 'silent': True},
name=None)
```

Parameters

`params`: Optional `run()` parameters.

```
msgs_i(self, p)
```

```
trans_i(self, p, msgs)
```

```
halt_i(self, p)
```

```
cleanup_i(self, p)
```

```
cleanup(self)
```

```
__call__(self, network, params={})
```

```
run(self, network, params={})
```

```
halt(self)
```

```
count_msg(self, message_count)
```

```
set(self, process, state, value)
```

```
increment(self, process, state, inc=1)
```

```
has(self, process, state)
```

```
get(self, process, state)
```

```
delete(self, process, state)
```

4.6 Class *Synchronous_Algorithm*

```
datk.core.distalgs.Algorithm └─ datk.core.distalgs.Synchronous_Algorithm
```

Known Subclasses: *datk.core.distalgs.Compose*, *datk.core.distalgs.Do_Nothing*, *datk.core.algs.FloodMax*, *datk.core.algs.LCR*, *datk.core.algs.SynchBFS*, *datk.core.algs.SynchBFSAck*, *datk.core.algs.SynchBroadcast*, *datk.core.algs.SynchConvergecast*, *datk.core.algs.SynchLubyMIS*

We assume that Processes take steps simultaneously, that is, that execution proceeds in synchronous rounds.

4.6.1 Methods

```
execute(self)
```

```
msgs(self)
```

```
round(self)
```

```
run(self, network, params={})
```

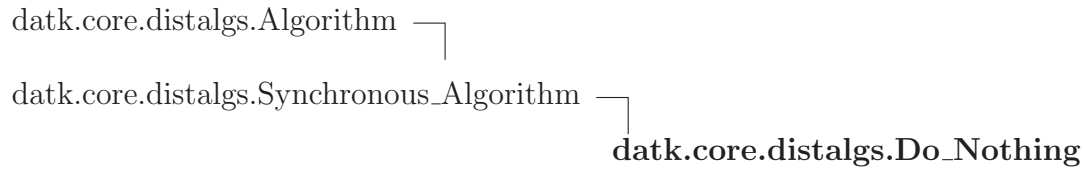
```
Overrides: datk.core.distalgs.Algorithm.run
```

```
trans(self)
```

Inherited from datk.core.distalgs.Algorithm(Section 4.5)

```
_call_(), _init_(), cleanup(), cleanup_i(), count_msg(), delete(), get(), halt(),  
halt_i(), has(), increment(), msgs_i(), set(), trans_i()
```

4.7 Class *Do_Nothing*



4.7.1 Methods

trans_i (<i>self</i> , <i>p</i> , <i>messages</i>) Overrides: <i>datk.core.distalgs.Algorithm.trans_i</i>

Inherited from datk.core.distalgs.Synchronous_Algorithm(Section 4.6)

execute(), msgs(), round(), run(), trans()

Inherited from datk.core.distalgs.Algorithm(Section 4.5)

call(), _init_(), cleanup(), cleanup_i(), count_msg(), delete(), get(), halt(), halt_i(), has(), increment(), msgs_i(), set()

4.8 Class *Asynchronous_Algorithm*



Known Subclasses: *datk.core.algs.AsyncLCR*, *datk.core.algs.AsyncConvergecast*

We assume that the separate Processes take steps in an arbitrary order, at arbitrary relative speeds.

4.8.1 Methods

run (<i>self</i> , <i>network</i> , <i>params</i> ={}) Overrides: <i>datk.core.distalgs.Algorithm.run</i>
--

run_process (<i>self</i> , <i>process</i>)

Inherited from datk.core.distalgs.Algorithm(Section 4.5)

`__call__()`, `__init__()`, `cleanup()`, `cleanup_i()`, `count_msg()`, `delete()`, `get()`, `halt()`,
`halt_i()`, `has()`, `increment()`, `msgs_i()`, `set()`, `trans_i()`

4.9 Class Compose



A `Synchronous_Algorithm` that is the composition of two synchronous algorithms running in parallel.

4.9.1 Methods

`__init__(self, A, B, name=None, params={'draw': False, 'silent': True})`

Parameters

`params`: Optional `run()` parameters.

Overrides: `datk.core.distalgs.Algorithm.__init__`

`msgs_i(self, p)`

Overrides: `datk.core.distalgs.Algorithm.msgs_i`

`trans_i(self, p, msgs)`

Overrides: `datk.core.distalgs.Algorithm.trans_i`

`halt_i(self, p)`

Overrides: `datk.core.distalgs.Algorithm.halt_i`

`cleanup_i(self, p)`

Overrides: `datk.core.distalgs.Algorithm.cleanup_i`

`run(self, network, params={})`

Overrides: `datk.core.distalgs.Algorithm.run`

`__repr__(self)`

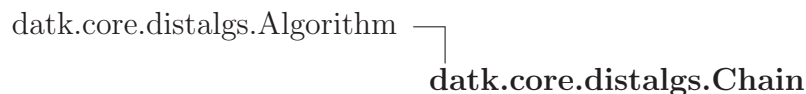
Inherited from `datk.core.distalgs.Synchronous_Algorithm`(Section 4.6)

`execute()`, `msgs()`, `round()`, `trans()`

Inherited from `datk.core.distalgs.Algorithm`(Section 4.5)

`__call__()`, `cleanup()`, `count_msg()`, `delete()`, `get()`, `halt()`, `has()`, `increment()`, `set()`

4.10 Class Chain



An Algorithm that is the result of sequentially running two algorithms

4.10.1 Methods

```
__init__(self, A, B, name=None, params={'draw': False, 'silent': False})
```

Parameters

`params`: Optional `run()` parameters.

Overrides: `datk.core.distalgs.Algorithm.__init__`

```
run(self, network, params={})
```

Overrides: `datk.core.distalgs.Algorithm.run`

```
__repr__(self)
```

Inherited from `datk.core.distalgs.Algorithm`(Section 4.5)

`__call__()`, `cleanup()`, `cleanup_i()`, `count_msg()`, `delete()`, `get()`, `halt()`, `halt_i()`, `has()`, `increment()`, `msgs_i()`, `set()`, `trans_i()`

5 Module `datk.core.imports`

5.1 Variables

Name	Description
<code>TIMEOUT</code>	Value: 5
<code>__package__</code>	Value: <code>'datk.core'</code>
<code>failed_tests</code>	Value: <code>set([])</code>
<code>lock</code>	Value: <code>Lock()</code>
<code>num_tests</code>	Value: 0

6 Module **datk.core.networks**

6.1 Variables

Name	Description
<code>__package__</code>	Value: <code>'datk.core'</code>

6.2 Class **Unidirectional_Ring**

`datk.core.distalgs.Network` └─ **`datk.core.networks.Unidirectional_Ring`**

A Network of *n* Processes arranged in a ring. Each edge is directed from a Process to its clockwise neighbor, that is, messages can only be sent in a clockwise direction.

6.2.1 Methods

```
__init__(self, n, index_to_UID=None)
```

Creates a network of *n* disconnected Processes, with random distinct UUIDs, or as specified by the `index_to_UID` function

Overrides: `datk.core.distalgs.Network.__init__` extit(inherited documentation)

Inherited from `datk.core.distalgs.Network`(Section 4.4)

```
__getitem__(), __iter__(), __len__(), __repr__(), add(), clone(), draw(), index(), run(), state()
```

6.3 Class **Bidirectional_Ring**

`datk.core.distalgs.Network` └─ **`datk.core.networks.Bidirectional_Ring`**

A Network of *n* Processes arranged in a ring. Each edge between a Process and its neighbor is undirected, that is, messages can be sent in both the clockwise and the counterclockwise directions.

6.3.1 Methods

```
__init__(self, n, index_to_UID=None)
```

Creates a network of *n* disconnected Processes, with random distinct UUIDs, or as specified by the *index_to_UID* function

Overrides: *datk.core.distalgs.Network.__init__* *exitit*(inherited documentation)

*Inherited from **datk.core.distalgs.Network**(Section 4.4)*

```
__getitem__(), __iter__(), __len__(), __repr__(), add(), clone(), draw(), index(), run(), state()
```

6.4 Class *Unidirectional_Line*

```
datk.core.distalgs.Network └─ datk.core.networks.Unidirectional_Line
```

A Network of *n* Processes arranged in a line. Each edge is directed from a Process to its clockwise neighbor, that is, messages can only be sent in a clockwise direction.

6.4.1 Methods

```
__init__(self, n, index_to_UID=None)
```

Creates a network of *n* disconnected Processes, with random distinct UUIDs, or as specified by the *index_to_UID* function

Overrides: *datk.core.distalgs.Network.__init__* *exitit*(inherited documentation)

*Inherited from **datk.core.distalgs.Network**(Section 4.4)*

```
__getitem__(), __iter__(), __len__(), __repr__(), add(), clone(), draw(), index(), run(), state()
```

6.5 Class *Bidirectional_Line*

```
datk.core.distalgs.Network └─ datk.core.networks.Bidirectional_Line
```

A Network of *n* Processes arranged in a line. Each edge between a Process and its neighbor is undirected, that is, messages can be sent in both the clockwise and the counterclockwise

directions.

6.5.1 Methods

```
__init__(self, n, index_to_UID=None)
```

Creates a network of n disconnected Processes, with random distinct UUIDs, or as specified by the `index_to_UID` function

Overrides: `datk.core.distalgs.Network.__init__` `exitit`(inherited documentation)

Inherited from `datk.core.distalgs.Network`(Section 4.4)

```
__getitem__(), __iter__(), __len__(), __repr__(), add(), clone(), draw(), index(), run(), state()
```

6.6 Class `Complete_Graph`



A Network of n Processes arranged at the vertices of a Complete undirected graph of size n.

6.6.1 Methods

```
__init__(self, n, index_to_UID=None)
```

Creates a network of n disconnected Processes, with random distinct UUIDs, or as specified by the `index_to_UID` function

Overrides: `datk.core.distalgs.Network.__init__` `exitit`(inherited documentation)

Inherited from `datk.core.distalgs.Network`(Section 4.4)

```
__getitem__(), __iter__(), __len__(), __repr__(), add(), clone(), draw(), index(), run(), state()
```

6.7 Class `Random_Line_Network`



A Network of n processes arranged randomly at the vertices of a connected undirected line graph of size n . Additional pairs of vertices are connected at random with a probability that is inversely proportional to the difference in their positions on the line.

For example, the Process at index 3 is guaranteed to be connected to the Process at index 4, and is more likely to be connected to the Process at index 5 than to the Process at index 8.

6.7.1 Methods

```
__init__(self, n, sparsity=1)
-----
sparsity = 0 -> a Complete_Graph(n) sparsity = infinity -> a
Bidirectional_Line(n)
Overrides: datk.core.distalgs.Network.__init__
```

Inherited from datk.core.distalgs.Network(Section 4.4)

```
__getitem__(), __iter__(), __len__(), __repr__(), add(), clone(), draw(), index(), run(),
state()
```

7 Module *datk.core.test*er

7.1 Functions

```
test(f=None, timeout=5, main_thread=False, test=True)
```

```
print_with_underline(text)
```

```
summarize()
```

7.2 Variables

Name	Description
TIMEOUT	Value: 5
lock	Value: Lock()
num_tests	Value: 0
failed_tests	Value: set([])
__package__	Value: 'datk.core'

8 Package `datk.tests`

8.1 Modules

- **helpers**: Helper functions for tests in `tests.py`
(Section 9, p. 30)
- **networks_tests**: Network Test Suite
(Section 10, p. 31)
- **tests**: Algorithm Test Suite
(Section 11, p. 32)

8.2 Variables

Name	Description
<code>--package--</code>	Value: None

9 Module `datk.tests.helpers`

Helper functions for tests in `tests.py`

9.1 Functions

testLeaderElection (<i>network</i> , <i>isLeader</i> =<function <lambda> at 0x27193f0>, <i>isNonleader</i> =<function <lambda> at 0x2719430>) <hr/> Asserts that exactly one Process is Leader, and all other processes are Non-Leader

testBroadcast (<i>network</i> , <i>attr</i>) <hr/> Asserts that <code>p.state[attr]</code> is identical for all processes <code>p</code>
--

testBFS (<i>network</i>) <hr/> Asserts that every Process, <code>p</code> , knows 'parent', and there exists exactly one Process where 'parent' is None

testBFSWithChildren (<i>network</i>) <hr/> Asserts that every Process, <code>p</code> , knows 'parent' and 'children', and there exists exactly one Process where 'parent' is None
--

testLubyMIS (<i>network</i>) <hr/> Asserts that every process knows a boolean value, 'MIS', and that the Processes where 'MIS' is True form a set that is both independent and maximal.

9.2 Variables

Name	Description
<code>--package--</code>	Value: <code>'datk.tests'</code>

10 Module `datk.tests.networks_tests`

Network Test Suite

Tests Networks defined in `networks.py` by visual inspection

10.1 Variables

Name	Description
<code>DRAW_RANDOM</code>	Value: <code>None</code>
<code>DRAW_HUGE_RANDOM</code>	Value: <code>None</code>
<code>DRAW_UNI_RING</code>	Value: <code>None</code>
<code>DRAW_BI_RING</code>	Value: <code>None</code>
<code>DRAW_COMPLETE_GRAPH</code>	Value: <code>None</code>
<code>DRAW_UNI_LINE</code>	Value: <code>None</code>
<code>DRAW_BI_LINE</code>	Value: <code>None</code>
<code>TIMEOUT</code>	Value: <code>5</code>
<code>__package__</code>	Value: <code>'datk.tests'</code>
<code>failed_tests</code>	Value: <code>set([])</code>
<code>lock</code>	Value: <code>Lock()</code>
<code>num_tests</code>	Value: <code>0</code>

11 Module *datk.tests.tests*

Algorithm Test Suite

Tests algorithms defined in *algs.py*

11.1 Functions

configure_ipython()

Convenient helper function to determine if environment is ipython. Note that drawing is only safe in ipython qtconsole with matplotlib inline. If environment is IPython, returns True and configures IPython. Else returns False.

LCR_UNI_RING()

LCR_BI_RING()

ASYNC_LCR_UNI_RING()

ASYNC_LCR_BI_RING()

FLOODMAX_UNI_RING()

FLOODMAX_BI_RING()

FLOODMAX_BI_LINE()

FLOODMAX_COMPLETE_GRAPH()

FLOODMAX_RANDOM_GRAPH()

SYNCH_BFS()

SYNCH_BFS_ACK()

SYNCH_CONVERGE_HEIGHT()

SYNCH_BROADCAST_HEIGHT()

<code>ASYNCH_BROADCAST_HEIGHT()</code>
--

<code>send_receive_msgs()</code>

<code>SYNCH_DO_NOTHING()</code>

<code>COMPOSE_SYNCH_LCR_AND_DO_NOTHING()</code>

<code>COMPOSE_SYNCH_LCR()</code>

<code>CHAIN_BROADCAST_HEIGHT()</code>

<code>SYNCH_LUBY_MIS_BI_RING()</code>

<code>SYNCH_LUBY_MIS()</code>

11.2 Variables

Name	Description
<code>in_ipython</code>	Value: <code>False</code>
<code>test_params</code>	Value: <code>{'draw': False, 'silent': True}</code>
<code>TIMEOUT</code>	Value: <code>5</code>
<code>--package--</code>	Value: <code>'datk.tests'</code>
<code>failed_tests</code>	Value: <code>set([])</code>
<code>lock</code>	Value: <code>Lock()</code>
<code>num_tests</code>	Value: <code>0</code>

Index

- datk (*package*), 2
 - datk.core (*package*), 3
 - datk.core.algs (*module*), 4–15
 - datk.core.distalgs (*module*), 16–22
 - datk.core.imports (*module*), 23
 - datk.core.networks (*module*), 24–27
 - datk.core.testers (*module*), 28
 - datk.tests (*package*), 29
 - datk.tests.helpers (*module*), 30
 - datk.tests.networks_tests (*module*), 31
 - datk.tests.tests (*module*), 32–33