Package 'xegaMigration'

August 8, 2026

Title 'Xega' Island Models

Version 0.5.0.4

Description Implements asynchronous message-passing communication protocols
for island models of extended and evolutionary algorithms
(see Tomassini, Marco (2005, ISBN:978-3-540-24193-5)) for the
R-package 'xega' <https://CRAN.R-project.org/package=xega>.
Basic asynchronous as well as synchronized communication primitives
are supplied based on file I/O operations ('rds') on a shared file system
or by 'openMPI' (MPI) messages. The gene selection and replacement
strategies, the migration policy as well as the communication topology
between islands are configurable.
Homogeneous and heterogeneous island algorithms are supported.
For examples (R and shell-scripts), see
<https://github.com/ageyerschulz/xega/tree/main/examples/IslandModels>.

License MIT + file LICENSE

URL https://github.com/ageyerschulz/xegaMigration

Encoding UTF-8

RoxygenNote 7.3.3

Depends R (>= 3.5.0)

Imports xegaSelectGene (>= 1.0.0.4), xegaPopulation

Suggests testthat (>= 3.0.0)

NeedsCompilation no

Author Andreas Geyer-Schulz [aut, cre] (ORCID:
<https://orcid.org/0009-0000-5237-3579>),
Seyedmostafa Zamanishandiz [aut] (ORCID:
<https://orcid.org/0009-0000-3324-4739>)

Maintainer Andreas Geyer-Schulz <Andreas.Geyer-Schulz@kit.edu>

Repository CRAN

Date/Publication 2026-08-08 11:30:02 UTC

Contents

xegaMigration-package *The migration support for island models for the* xega *package.*

Description

Implements asynchronous message-passing communication protocols for island models of extended and evolutionary algorithms (see Tomassini, Marco (2005, ISBN:978-3-540-24193-5)) for the R-package 'xega' https://CRAN.R-project.org/package=xega. Basic asynchronous as well as synchronized communication primitives are supplied based on file I/O operations ('rds') on a shared file system or by 'openMPI' (MPI) messages. The gene selection and replacement strategies, the migration policy as well as the communication topology between islands are configurable. Homogeneous and heterogeneous island algorithms are supported. For examples (R and shell-scripts), see https://github.com/ageyerschulz/xega/tree/main/examples/IslandModels.

An Adaptive Completely Decentral Migration Strategy

The goal of the migration strategy of xega is to exploit the computing power of a set of loosely coupled processors as well as possible:

- Processes should not wait.
- All processes should terminate almost at the same time, so that no processor is idle for an extended period of time.
- Migration policies (e.g. on improvement) should be configurable.
- The number of messages exchanged between islands should be small (and configurable).
- The communication topology should be configurable.
- The algorithm should terminate, when either
  - one process fulfills a local termination predicate (reaches the optimal solution),
  - the slowest processor reaches its generation limit. It is not known which of the processes is the slowest.
  - The hardware/software requirements should be low and the software should be scalable.
    The first requirement is addressed by providing process communication via file I/O on a shared file system (rds) and can be used with just base R and xega for Linux and macOS operating systems.
    Scalability is supported by process communication via MPI.

Island models require the following communication protocols between island algorithms:

1. An (asynchronous) message exchange protocol.
2. A synchronization mechanism (optional).
3. An (asynchronous) Termination protocol.
4. A result collection protocol (optional).

The (Basic) Migration Algorithm

1. Probe (non-blocking) for a distributed termination predicate DTP. from other islands (asynchronous termination protocol). If a distributed termination predicate is set, return DTP to main.
2. If local termination predicate LTP, broadcast termination signal to other islands. (asynchronous termination protocol).
3. Select emigrants from population and send emigrants to neighboring islands (asynchronous message exchange protocol). The neighborhood of an island is defined by the communication topology graph. (With the `xegaRun()` option `migrate="OnImprovement"` the emigration step is skipped, if no fitness improvement has occured in the last generation.
4. Receive immigrants and replace genes in population by immigrants (asynchronous message exchange protocol or barrier synchronization followed by asynchronous message exchange protocol).
5. Update the generation limit and identify the slowest pid.

The Asynchronous Message Exchange Protocol

Each island process performs the essentially the following two communication steps:

1. It **sends** a list with genes, the slowest time, and the pid of the slowest process to a list of receiving processes.
2. It **receives** none, one, or several list(s) with genes, the slowest time, and the pid of the slowest process.

	Send	**Receive**
rds	rdsSend()	rdsReceiveGenes()
MPI	mpiSend()	mpiReceiveGenes() with message tag 9

Synchronization of Island Processes

Synchronization of island processes is achieved by implementing a blocking message receive function which uses barrier synchronization (a barrier function which blocks until all island processes reach the barrier) before the asynchronous message receive function is called.

	Receive	**Barrier Used**
rds	rdsReceiveGenesBlocking()	rdsBarrier()
MPI	mpiReceiveGenesBlocking()	Rmpi::mpiBarrier()

The (Asynchronous) Termination Protocol

The **terminator** (an island process who detects a local termination condition either by reaching an optimization goal or by resource exhaustion) broadcasts a termination message to the **terminated** processes (all other island processes in the process ensemble) and terminates. All other island processes receive the termination message and terminate.

- `rds` communication. All island processes have access to a shared file system. The **terminator** writes a termination file with the file name TermFrom<spid>ToAllRND<pad>.rds and terminates. Each **terminated** process tests for the existence of this file and if the file exists, it terminates. The termination file remains in the shared directory and is not deleted.
- `MPI` communication. All island processes are members of the MPI communication area 1 and share a MPI message bus. The **terminator** sends a termination message with tag 7 to the **terminated** processes. Each **terminated** process tests for the existence of a termination message with tag 7, and if such messages exist, it consumes them and terminates. The termination message which the **terminator** has sent to itself is not consumed and remains on the MPI message bus.

	Terminator	**terminated**
rds	rdsBroadcastTerm()	rdsProbeTerm()
mpi	mpiBroadcastTerm()	mpiProbeTerm() with message tag 7

Collection of Results

Island models may use thousands of loosely-coupled concurrent genetic algorithms each of which produces its own xegaRun result object. For convenience, the user may configure a result collection step which produces a single result object which consists of the list of all xegaRun objects and a return code which indicates if the results of all island processes have been collected successfully.

By convention, the process with pid `0` is considered as master process which receives the results. All other processes send their processes to pid `0`.

	Function	**uses:**
rds	rdsCollect()	readRDS() with error handling
MPI	mpiCollect()	If `pid==0`: mpiReceiveResult() with message tag 8 If `not(pid==0)`: mpiSendResult() with message tag 8

MPI and rds Communication Primitives

MPI and rds use communication primitives with the same semantics. However, at the moment all collective MPI primitives (except `Rmpi::mpi.barrier()`) are avoided.

send gene(s) from x to y (gene)	mpiSendGenes()	rdsSendGenes()
uses	Rmpi::mpi.send.Robj()	
receive gene(s) from any pid	mpiReceiveGenes	rdsReceiveGenes()
uses	Rmpi::mpi.iprobe() Rmpi::mpi.any.source() Rmpi::mpi.recv.Robj()	
Loose synchronization	Rmpi::mpi.barrier()	rdsBarrier()

rds Communication Primitives

It is assumed that all island algorithms of a single island model have exclusive read/write access to a directory on the shared file system. The atomicity of file operations is guaranteed by the file naming conventions defined below. *Probing* is implemented by file existence tests. However, (rare) file read errors are caught by error handling and delayed retry. File read errors may occur if an attempt is made to read a file which has not yet been completely written.

rds communication primitives rely on file name conventions:

- **Message Exchange Protocol.** File names produced by **rdsSendGenes()** and used by **rdsReceiveGenes()**:
  From<pid>To<pid>RND<random digits>.rds
- **Termination Protocol.** File names produced by **rdsBroadcastTerm()** and used **rdsProbeTerm()**:
  TermFrom<pid>ToAllRND<random digits>.rds
- **Barrier synchronization.** File names produced and used by **rdsBarrier()**:
  Barrier<pid>pad<random digits>.rds
- **Result Collection.** File names produced by **xegaRun()** of island processes and used by **xegaRun()** of process `0`:
  xegaResult<exclusive pattern>.rds
  File name produced by **xegaRun()** of island process `0`:
  xegaIResult<exclusive pattern>.rds

Author(s)

Maintainer: Andreas Geyer-Schulz <Andreas.Geyer-Schulz@kit.edu> (ORCID)

Authors:

- Seyedmostafa Zamanishandiz <mostafa-shandiz@partner.kit.edu> (ORCID)

See Also

Useful links:

- https://github.com/ageyerschulz/xegaMigration

---

adaptId *The generation limit is not changed.*

Description

The generation limit is not changed.

Usage

```
adaptId(lF)
```

Arguments

lF	Local function configuration.

Value

The number of generations (integer).

See Also

Other Migration: NewLFxegaMigrate(), adaptSlowest(), xegaMigrate()

Examples

```
lF<-list()
lF$slowestTime<-function(){5}
lF$Generations<-function() {100}
lF$avgTime<-function() {2.5}
adaptId(lF)
```

---

adaptSlowest *The generation limit is adapted to the slowest process.*

Description

The function computes an update of the generation limit for faster island processes. The goal is to match the run-time of the faster island processes to the run-time of the slowest island process in order to keep all processors busy as long as the slowest process runs.

Usage

```
adaptSlowest(lF)
```

Arguments

lF	Local function configuration.

Value

The (new) number of generations (integer).

See Also

Other Migration: NewLFxegaMigrate(), adaptId(), xegaMigrate()

Examples

```
lF<-list()
lF$slowestTime<-function(){5}
lF$Generations<-function() {100}
lF$avgTime<-function() {2.5}
adaptSlowest(lF)
```

`gPetersenTop` *Select neighbours in a Generalized Petersen graph GP(n,k).*

Description

The Generalized Petersen graph GP(n, k) is a 3-regular graph on `2n` vertices. Outer ring vertices `0:(n-1)` form a cycle; inner ring vertices `n:(2n-1)` form a `k`-skip cycle; each outer vertex `i` is connected to inner vertex `n + i` by a spoke. The most famous instance GP(5, 2) is the Petersen graph itself: vertex- and edge-transitive, distance-transitive, strongly regular, adjacent vertices without common neigbour, and non-connected vertices share exactly one common neighbor.

Usage

```
gPetersenTop(lF)
```

Arguments

`lF`	Local function configuration. Required elements are • `lF$npid()`: Total number of processes; must equal `2 * lF$gp_n()`. • `lF$pid()`: Process number of message sender. • `lF$GP_n()`: Integer n of the GP(n, k) family. Must be ≥ 3. • `lF$GP_k()`: Integer k of the GP(n, k) family. Must satisfy $1 \leq k \leq n - 1$.

Details

Each processing unit in GP(n, k) has exactly three neighbours (the graph is cubic). `gpTop()` returns all three by default.

Value

Integer vector of process numbers of message receivers. Length 3 (the cubic neighbour set).

References

Watkins, M. E. (1969). A theorem on Tait colorings with an application to the generalized Petersen graphs. *J. Combin. Theory* 6:152-164. <doi:10.1016/S0021-9800(69)80116-X>

Holton, D. A. & Sheehan, J. (1993). *The Petersen Graph*. Cambridge University Press. (ISBN:978-0521435949)

See Also

Other Communication Topology: `ring2Top()`, `ringTop()`, `rndTop()`, `torus2DTop()`, `torus3DTop()`

Examples

```
lF<-list()
lF$npid<-function() {10}
lF$GPn<-function() {5}
lF$GPk<-function() {2}
lF$pid<-function()  {0}
gPetersenTop(lF)    # outer vertex 0: prev=4, next=1, spoke=5
lF$pid<-function()  {7}
gPetersenTop(lF)    # inner vertex 7 (= n+2): inner_prev, inner_next, spoke=2
```

`mpiBroadcastTerm` *Broadcast termination message (mpi).*

Description

Sends termination messages (`tag=7`) to all island processes.

Usage

```
mpiBroadcastTerm(lF)
```

Arguments

`lF`	Local function configuration.

Details

Expects `lF$RmpiFNS` elements bound to `Rmpi` functions `mpi.iprobe()`, `mpi.any.source()`, and `mpi.recv.Robj()`.

The termination messages (tag=7) are sent to all processes with pids ranging from `0` to `lF$npid()`. The message that the broadcaster sends to himself is not received and remains in the mpi queue.

Value

0 (invisible).

mpiCollect *Collects results of all island processes (mpi).*

Description

Upon termination all island processes (except process `0`) write their result object to a rds-file with prefix "xegaResult". Process `0` reads these files and aggregates them.

Usage

```
mpiCollect(result = list(), lF = lF)
```

Arguments

`result`	A result of xegaRun. Default: `list()`.
`lF`	Local function configuration. Default: lF.

Value

A named list with

- `$results`: A list of xegaRun result objects.
- `$rc`: The return code. A return code of `0` means that the results of all processes have been collected. A return code of `-1` means that not all results have been collected within a specified time interval.

mpiProbeTerm *Probes for termination message(s) (MPI).*

Description

Probes for termination messages (`tag=7`). If messages exist, returns `TRUE` for setting the `DTP` (the Distributed Termination Predicate) and consumes all termination messages.

Usage

```
mpiProbeTerm(lF)
```

Arguments

`lF`	Local function configuration.

Details

Expects `lF$RmpiFNS` elements bound to `Rmpi` functions `mpi.iprobe()`, `mpi.any.source()`, and `mpi.recv.Robj()`.

Value

Boolean. TRUE indicates that a termination message has been received and that the process should terminate.

See Also

Other MPI communication: mpiReceiveGenes(), mpiReceiveGenesBlocking(), mpiReceiveResult(), mpiSendGenes(), mpiSendResults()

---

mpiReceiveGenes *Receive genes from neighbor processes (non-blocking).*

Description

Multiple MPI messages are received, if they exist (non-blocking). Test for existence of message: Rmpi::mpi.iprobe() and message receive: Rmpi::mpi.recv.Robj().

Usage

```
mpiReceiveGenes(lF)
```

Arguments

lF	Local function configuration.

Value

A gene list.

See Also

Other MPI communication: mpiProbeTerm(), mpiReceiveGenesBlocking(), mpiReceiveResult(), mpiSendGenes(), mpiSendResults()

---

mpiReceiveGenesBlocking *Receive genes from neighbor processes (blocking).*

Description

Blocking means barrier synchronization. Multiple MPI messages are received, if they exist. Test for existence of message (blocks): mpi.probe() and message receive: mpi.recv.Robj().

Usage

```
mpiReceiveGenesBlocking(lF)
```

Arguments

lF	Local function configuration.

Value

A gene list.

See Also

Other MPI communication: mpiProbeTerm(), mpiReceiveGenes(), mpiReceiveResult(), mpiSendGenes(), mpiSendResults()

---

mpiReceiveResult *Receive a result (non-blocking).*

Description

A result object (tag=8) is received, if one exists.

Usage

```
mpiReceiveResult(lF)
```

Arguments

lF	Local function configuration.

Value

A result object or an empty list.

See Also

Other MPI communication: mpiProbeTerm(), mpiReceiveGenes(), mpiReceiveGenesBlocking(), mpiSendGenes(), mpiSendResults()

`mpiSendGenes` *Send genes to neighbor process(es).*

Description

A list of genes (the emigrants) is sent to each destination. The list of destinations is determined by the communication topology used. If there is more than one destination, the same list of emigrants is sent to each destination.

Usage

```
mpiSendGenes(genes, lF)
```

Arguments

`genes`	A gene list.
`lF`	Local function configuration.

Details

Expects `Rmpi::mpi.send.Robj` bound to `lF$RmpiFNS$mpi.send.Robj`. The MPI message must be tagged with 9.

Value

0 (invisible)

See Also

Other MPI communication: `mpiProbeTerm()`, `mpiReceiveGenes()`, `mpiReceiveGenesBlocking()`, `mpiReceiveResult()`, `mpiSendResults()`

`mpiSendResults` *Send a a xega result to pid* `0` *with tag* `8`.

Description

Sends the result object of the island algorithm to the master process.

Usage

```
mpiSendResults(result, lF)
```

Arguments

`result`	The result of the island algorithm.
`lF`	Local function configuration.

Details

Expects `Rmpi::mpi.send.Robj` bound to `lF$RmpiFNS$mpi.send.Robj`. The MPI message must be tagged with 8.

Value

0 (invisible)

See Also

Other MPI communication: `mpiProbeTerm()`, `mpiReceiveGenes()`, `mpiReceiveGenesBlocking()`, `mpiReceiveResult()`, `mpiSendGenes()`

NewLFxegaMigrate *Generate a local function list to test migration.*

Description

To allow local testing of functions of `xegaMigration`, the function factory `NewLFxegaMigrate` returns a local function list with some of the local functions used in configuring `xega::xegaRun()`.

Usage

```
NewLFxegaMigrate()
```

Value

An object of class list of length 51. All list elements are local functions or lists of local functions for configuring `xega::xegaRun`.

1. For some elements of the local function list for configuring migration:
   (a) `$path()`. Returns path of directories for results and rds-file communication.
   (b) `$pid()`. Returns process id (pid) of island.
   (c) `$npid()`. Returns number of islands.
   (d) `$Nmigrants()`. Returns number of emigrants.
   (e) `$nrecv()`. Number of receiving island processes. For random communication topology.
   (f) `$GPn()`. Number of island process in inner and outer rings of a communication topology in the form of a generalized Peterson graph.
   (g) `$GPk()` Spoke shift between inner and outer ring nodes in a generalized Peterson graph.
   (h) `$torusX()` Number of processes on X-coordinate of a 2-D or 3-D torus of processes.
   (i) `$torusY()` Number of processes on Y-coordinate of a 2-D or 3-D torus of processes.
   (j) `$torusZ()` Number of processes on Z-coordinate of a 3-D torus of processes.
   (k) `$TopK()`. Returns number of genes.
   (l) `$SelMigrant()`. Returns selection method for emigrants.
   (m) `$SelReplace()`. Returns Replacement method for genes by immigrants.

(n) `$CommunicationTopology()`. Returns function for computing all neigboring pids as defined by communication graph.
(o) `$Send()`. Returns send method.
(p) `$Receive()`. Returns receive method.
(q) `$ProbeTerm()`. Returns probing function for termination message.
(r) `$BroadcastTerm()`. Returns broadcast function for sending termination message to all island processes.
(s) `$LTP()`. Local Termination Predicate.
(t) `$avgTime()`. Average execution time of island.
(u) `$slowestTime()`. Slowest execution time known at island.
(v) `$slowestPid()`. pid of slowest process in ensemble of island processes.
(w) `$migrationStrategy()`. Returns a boolean function which triggers termination protocol.

2. For the elements of the local function list of `xega::xegaRun()`,
   - `$penv()`, `$replay()`, `$verbose()`, `$CutoffFit()`, `$CBestFitness()`, `$CWorstFitness()`, `$MutationRate1()`, `$MutationRate2()`, `$BitMutationRate1()`, `$BitMutationRate2()`, `$MutationRate()`, `$MutateGene()`, `$CrossRate()`, `$UCrossSwap()`, `$CrossGene()`, `$Max()`, `$Offset()`, `$Eps()`, `$Elitist()`, `$TournamentSize()`, `$GeneMap()`, `$SelectGene()`, `$SelectMate()`, `$Accept()`, `$ReportEvalErrors()`, `$Pipeline()`, `$InitGene()`, `$DecodeGene()`, `$EvalGene()`, `$SelectionContinuation()`, `$Verbose()`, and `$lapply()`.

See Also

Other Migration: `adaptId()`, `adaptSlowest()`, `xegaMigrate()`

`rdsBarrier` *Stop until all processes have reached the barrier.*

Description

The island process reaching the barrier stops until all other island processes also reach the barrier. Then all island processes continue. `rdsBarrier()` implements a group lock mechanism based on file I/O operations with the same synchronization behavior as `Rmpi::mpiBarrier()`.

Usage

```
rdsBarrier(lF)
```

Arguments

`lF`	Local function configuration.

Details

Each island process writes its barrier file and tests if the number of barrier files in the directory matches the number of island processes. The island process which detects that all island processes have arrived at the barrier, removes all barrier files (and thus releases all other island processes) before it continues.

Value

0 (invisible)

See Also

Other rds communication: rdsBarrierFileName(), rdsBroadcastTerm(), rdsFileName(), rdsProbeTerm(), rdsReceiveGenes(), rdsReceiveGenesBlocking(), rdsSendGenes(), rdsTermFileName()

Examples

```
cat("No examples available!\n")
```

---

rdsBarrierFileName *Construct a unique barrier file name with sender embedded in name.*

Description

Construct a unique barrier file name with sender embedded in name.

Usage

```
rdsBarrierFileName(from, path = ".")
```

Arguments

from	Integer (pid of message sender).
path	File path. Default: ".".

Value

A file name of the form Barrier<pid>.rds

See Also

Other rds communication: rdsBarrier(), rdsBroadcastTerm(), rdsFileName(), rdsProbeTerm(), rdsReceiveGenes(), rdsReceiveGenesBlocking(), rdsSendGenes(), rdsTermFileName()

Examples

```
rdsBarrierFileName(3)
```

rdsBroadcastTerm *Broadcast termination message (rds).*

Description

Uses saveRDS for writing termination message to file. The file name is of the form TermFrom<spid>ToAllRND<pad>.rds.

Usage

```
rdsBroadcastTerm(lF)
```

Arguments

lF	Local function configuration.

Details

The termination message is "received" by the function `rdsProbeTerm()` by testing for the existence of a file with the name TermFrom<spid>ToAllRND<pad>.rds in a file system shared by all island processes.

Value

0 (invisible).

See Also

Other rds communication: `rdsBarrier()`, `rdsBarrierFileName()`, `rdsFileName()`, `rdsProbeTerm()`, `rdsReceiveGenes()`, `rdsReceiveGenesBlocking()`, `rdsSendGenes()`, `rdsTermFileName()`

Examples

```
lF<-list()
lF$pid<-function() {4}
path<-tempdir()
lF$path<- function() {path}
rdsBroadcastTerm(lF)
DPT<-rdsProbeTerm(lF)
cat("DPT", DPT, "\n")
```

rdsCollect — *Collects results of all island processes (rds).*

Description

Upon termination all island processes (except process `0`) write their result object to a rds-file with prefix "xegaResult". Process `0` reads these files and aggregates them.

Usage

```
rdsCollect(result = list(), lF = lF)
```

Arguments

`result`	A result of xegaRun. Default: `list()`.
`lF`	Local function configuration. Default: lF.

Value

A named list with

- `$results`: A list of xegaRun result objects.
- `$rc`: The return code. A return code of `0` means that the results of all processes have been collected. A return code of `-1` means that not all results have been collected within a specified time interval.

rdsFileName — *Construct a unique file name with sender and receiver embedded in name.*

Description

Construct a unique file name with sender and receiver embedded in name.

Usage

```
rdsFileName(from, to, path = ".")
```

Arguments

`from`	Integer (pid of message sender).
`to`	Integer (pid of message receiver).
`path`	File path. Default: ".".

Value

A file name of the form From<spid>To<rpid>RND<pad>.rds

See Also

Other rds communication: `rdsBarrier()`, `rdsBarrierFileName()`, `rdsBroadcastTerm()`, `rdsProbeTerm()`, `rdsReceiveGenes()`, `rdsReceiveGenesBlocking()`, `rdsSendGenes()`, `rdsTermFileName()`

Examples

```
rdsFileName(3, 2)
```

`rdsProbeTerm` *Probes for termination message(s) (rds).*

Description

Tests if a rds file with the filename TermFrom<spid>ToAllRND<pad>.rds exists. If such a file exists, returns `TRUE` for setting the `DTP` (the Distributed Termination Predicate).

Usage

```
rdsProbeTerm(lF)
```

Arguments

`lF`	Local function configuration.

Details

The termination message is sent by the function `rdsBroadcastTerm()` in the form of a rds file with the filename TermFrom<spid>ToAllRND<pad>.rds.

Value

Boolean. `TRUE` indicates that a termination message has been received and that the process should terminate.

See Also

Other rds communication: `rdsBarrier()`, `rdsBarrierFileName()`, `rdsBroadcastTerm()`, `rdsFileName()`, `rdsReceiveGenes()`, `rdsReceiveGenesBlocking()`, `rdsSendGenes()`, `rdsTermFileName()`

Examples

```
lF<-list()
path<-tempdir()
lF$path<- function() {path}
DPT<-rdsProbeTerm(lF)
cat("DPT", DPT, "\n")
fn<-rdsTermFileName(5, path=lF$path())
d<-"Terminate!"
saveRDS(object=d, file=fn)
DPT<-rdsProbeTerm(lF)
cat("DPT", DPT, "\n")
```

`rdsReceiveGenes` *Receive genes from neighbor processes (non-blocking).*

Description

Multiple messages are received, if they exist (non-blocking).

Usage

```
rdsReceiveGenes(lF)
```

Arguments

`lF`	Local function configuration.

Value

A gene list.

See Also

Other rds communication: `rdsBarrier()`, `rdsBarrierFileName()`, `rdsBroadcastTerm()`, `rdsFileName()`, `rdsProbeTerm()`, `rdsReceiveGenesBlocking()`, `rdsSendGenes()`, `rdsTermFileName()`

Examples

```
lF<-list()
lF$npid<-function() {10}
lF$pid<-function()  {3}
lF$CommunicationTopology<-xegaCommunicationTopologyFactory(method="ring")
path<-tempdir()
lF$path<- function() {path}
genes<-list(sample(0:1, 10, replace=TRUE))
rdsSendGenes(genes, lF)
fn<-list.files(lF$path())
rdsReceiveGenes(lF)
lF$pid<-function()  {4}
rdsReceiveGenes(lF)
```

rdsReceiveGenesBlocking *Receive genes from neighbor processes (blocking).*

Description

Blocking means barrier synchronization.

Usage

```
rdsReceiveGenesBlocking(lF)
```

Arguments

lF	Local function configuration.

Value

A gene list.

See Also

Other rds communication: rdsBarrier(), rdsBarrierFileName(), rdsBroadcastTerm(), rdsFileName(), rdsProbeTerm(), rdsReceiveGenes(), rdsSendGenes(), rdsTermFileName()

rdsSendGenes *Send genes to neighbor process.*

Description

Uses saveRDS for writing messages to files. The file name is of the form From<spid>To<rpid>RND<pad>.rds

Usage

```
rdsSendGenes(genes, lF)
```

Arguments

genes	A gene list.
lF	Local function configuration.

Value

0 (invisible)

See Also

Other rds communication: rdsBarrier(), rdsBarrierFileName(), rdsBroadcastTerm(), rdsFileName(), rdsProbeTerm(), rdsReceiveGenes(), rdsReceiveGenesBlocking(), rdsTermFileName()

Examples

```
lF<-list()
lF$npid<-function() {10}
lF$pid<-function()  {3}
lF$nrecv<-function() {1}
lF$CommunicationTopology<-xegaCommunicationTopologyFactory(method="random")
path<-tempdir()
lF$path<- function() {path}
genes<-list(sample(0:1, 10, replace=TRUE))
print(genes)
rdsSendGenes(genes, lF)
fn<-list.files(lF$path(), pattern="*\\.rds")
print(fn)
```

---

rdsTermFileName *Construct a unique termination file name with sender embedded in name.*

Description

Construct a unique termination file name with sender embedded in name.

Usage

```
rdsTermFileName(from, path = ".")
```

Arguments

from	Integer (pid of message sender).
path	File path. Default: ".".

Value

A file name of the form TermFrom<spid>ToAllRND<pad>.rds

See Also

Other rds communication: rdsBarrier(), rdsBarrierFileName(), rdsBroadcastTerm(), rdsFileName(), rdsProbeTerm(), rdsReceiveGenes(), rdsReceiveGenesBlocking(), rdsSendGenes()

Examples

```
rdsTermFileName(3)
```

ring2Top *Select neighbors in bidirectional ring topology.*

Description

Select neighbors in bidirectional ring topology.

Usage

```
ring2Top(lF)
```

Arguments

lF	Local function condiguration. Required element are • lF$npid(): Total number of processes. • lF$pid(): Process number of message sender.

Value

Process numbers of message receivers.

See Also

Other Communication Topology: gPetersenTop(), ringTop(), rndTop(), torus2DTop(), torus3DTop()

Examples

```
lF<-list()
lF$npid<-function() {10}
lF$pid<-function()  {3}
ring2Top(lF)
lF$pid<-function()  {9}
ring2Top(lF)
```

ringTop *Select neighbor in ring topology.*

Description

Select neighbor in ring topology.

Usage

```
ringTop(lF)
```

Arguments

lF	Local function condiguration. Required element are • lF$npid(): Total number of processes. • lF$pid(): Process number of message sender.

Value

Process number of message receiver.

See Also

Other Communication Topology: gPetersenTop(), ring2Top(), rndTop(), torus2DTop(), torus3DTop()

Examples

```
lF<-list()
lF$npid<-function() {10}
lF$pid<-function()  {3}
ringTop(lF)
lF$pid<-function()  {9}
ringTop(lF)
```

---

rndTop *Select one or more random neighbors from all neighbors.*

Description

Select one or more random neighbors from all neighbors.

Usage

```
rndTop(lF)
```

Arguments

lF	Local function condiguration. Required element are • lF$npid(): Total number of processes. • lF$pid(): Process number of message sender. • lF$nrecv(): Number of message receivers.

Value

Process number(s) of message receiver(s).

See Also

Other Communication Topology: gPetersenTop(), ring2Top(), ringTop(), torus2DTop(), torus3DTop()

Examples

```
lF<-list()
lF$npid<-function() {10}
lF$pid<-function()  {3}
lF$nrecv<-function() {1}
rndTop(lF)
lF$nrecv<-function() {2}
rndTop(lF)
```

`torus2DTop` *Select neighbors on a 2D-torus.*

Description

Processors are arranged in a X times Y grid. This implies that `lF$npid()==lF$torusX()*lF$torusY()`.

Usage

```
torus2DTop(lF)
```

Arguments

`lF`	Local function condiguration. Required element are • `lF$torusX()`: Number of elements in X axes. • `lF$torusY()`: Number of elements in Y axes. • `lF$pid()`: Process number of message sender.

Details

The algorithm works in the following way:

1. `lF$pid()` is converted into the grid coordinates (x, y) by the local function `n2xy()`.
2. The four neighbours in a distance of 1 are determined.
3. The coordinates are converted back to processor identifiers by the local function `xy2n()`.
4. The list of identifiers of the neighbor processes is returned.

Value

Process numbers of message receivers.

See Also

Other Communication Topology: `gPetersenTop()`, `ring2Top()`, `ringTop()`, `rndTop()`, `torus3DTop()`

Examples

```
lF<-list()
lF$pid<-function() {5}
lF$torusX<-function() {3}
lF$torusY<-function() {2}
torus2DTop(lF)
```

`torus3DTop` *Select neighbors on a 3D-torus.*

Description

Processors are arranged in a X times Y times Z grid. This implies that `lF$npid()==lF$torusX()*lF$torusY()*lF$torusZ`

Usage

```
torus3DTop(lF)
```

Arguments

`lF`	Local function condiguration. Required element are • `lF$torusX()`: Number of elements in X axes. • `lF$torusY()`: Number of elements in Y axes. • `lF$torusZ()`: Number of elements in Z axes. • `lF$pid()`: Process number of message sender.

Details

The algorithm works in the following way:

1. `lF$pid()` is converted into the grid coordinates (x, y, z) by the local function `n2xyz()`.
2. The six neighbours in a distance of 1 are determined.
3. The coordinates are converted back to processor identifiers by the local function `xyz2n()`.
4. The list of identifiers of the neighbor processes is returned.

Value

Process numbers of message receivers.

See Also

Other Communication Topology: `gPetersenTop()`, `ring2Top()`, `ringTop()`, `rndTop()`, `torus2DTop()`

Examples

```
lF<-list()
lF$pid<-function() {5}
lF$torusX<-function() {3}
lF$torusY<-function() {3}
lF$torusZ<-function() {3}
torus3DTop(lF)
```

`xegaAdaptGenerationLimitFactory` *Factory for configuring the adaptation of the generation limit.*

Description

Avalailable methods:

1. "Slowest": Adapt generation limit according to slowest process.
2. "Id": Do not adapt. Fast processes end earlier.

Usage

```
xegaAdaptGenerationLimitFactory(method = "Slowest")
```

Arguments

`method`	Method. Default: "Adapt".

Value

A function for the adapting generation limit.

See Also

Other Configuration: `xegaBroadcastTermFactory()`, `xegaCollectFactory()`, `xegaCommunicationTopologyFactory()`, `xegaProbeTermFactory()`, `xegaReceiveFactory()`, `xegaSendFactory()`

Examples

```
xegaAdaptGenerationLimitFactory(method="Slowest")
```

xegaBroadcastTermFactory *Factory for configuring broadcasting of termination messages.*

Description

Avalailable methods:

1. "rds": Broadcast termination rds-file.
2. "mpi": Broadcast termination message via mpi. Code with comments.

Usage

```
xegaBroadcastTermFactory(method = "rds")
```

Arguments

method	Method. Default: "rds".

Value

A function for broadcasting a termination message.

See Also

Other Configuration: xegaAdaptGenerationLimitFactory(), xegaCollectFactory(), xegaCommunicationTopologyFa
xegaProbeTermFactory(), xegaReceiveFactory(), xegaSendFactory()

Examples

```
xegaBroadcastTermFactory(method="rds")
```

xegaCollectFactory *Factory for configuring the collection of results of island results.*

Description

Avalailable methods:

1. "rds": Collects xegaRun result files. May terminate by a time out, before all result files have been produced.
2. "mpi": Collects xegaRun result files. May terminate by a time out, before all result files have been produced.

Usage

```
xegaCollectFactory(method = "rds")
```

Arguments

method	Method. Default: "rds".

Value

A function for collecting xegaRun result objects.

See Also

Other Configuration: xegaAdaptGenerationLimitFactory(), xegaBroadcastTermFactory(), xegaCommunicationTopologyFactory(), xegaProbeTermFactory(), xegaReceiveFactory(), xegaSendFactory()

Examples

```
xegaCollectFactory(method="rds")
```

---

xegaCommunicationTopologyFactory

Factory for configuring the communication topology.

Description

Avalailable methods:

1. "random": Returns a function which selects (a) random message receiver(s).
2. "ring": Returns a function which selects the ring neighbour mod(i+1, n) of node i as message receiver.
3. "ring2": Returns a function which selects the ring neighbours mod(i+1, n) and mod(i-1, n) of node i as message receiver.
4. "torus2D": Returns a function which selects the neighbours on a 2D-torus.
5. "torus3D": Returns a function which selects the neighbours on a 3D-torus.
6. "gPetersen": Returns a function which selects the neigbors of the process in a generalized Petersen Graph PG(n, k). The number of processes must be 2n, and 1<k<n.

Usage

```
xegaCommunicationTopologyFactory(method = "random")
```

Arguments

method	Method. Default: "random".

Value

A function containing the communication topology for migration.

See Also

Other Configuration: `xegaAdaptGenerationLimitFactory()`, `xegaBroadcastTermFactory()`, `xegaCollectFactory()`, `xegaProbeTermFactory()`, `xegaReceiveFactory()`, `xegaSendFactory()`

Examples

```
xegaCommunicationTopologyFactory(method="random")
```

`xegaMigrate` *Migrate genes.*

Description

The migration algorithm performs the following steps (neglecting termination):

1. Select emigrants.
2. Send emigrants to recipients defined by the communication topology.
3. Receive immigrants.
4. Replace some genes in the population by immigrants.

Usage

```
xegaMigrate(population, fit, lF)
```

Arguments

`population`	A population.
`fit`	A fitness vector.
`lF`	Local function configuration.

Details

The classic non-blocking migration strategy with termination is (simplified)

1. Probe for termination signals of other islands. If such a termination signal exists, return a termination signal.
2. If a local termination signal exists, broadcast termination to other islands and return a termination signal.
3. Select the best gene.
4. Send it to the neigbor (in a ring topology).

5. Receive immigrants from the neighbor (in a ring topology).
6. If there are immigrants, replace the worst genes by the immigrants.
7. Update generation limit and slowest pid.
8. Return population (with immigrants), generation limit, slowest time, and slowest pid.

Value

A named list with the following elements:

1. $pop A population.
2. $rucksack Control information (named list)
   (a) $DTP Boolean. Distributed termination predicate.
   (b) $generationLimit Integer. How many generations?
   (c) $slowestTime Integer. The execution time of the slowest process.
   (d) $slowestpid Integer. The pid of the slowest process.

See Also

Other Migration: `NewLFxegaMigrate()`, `adaptId()`, `adaptSlowest()`

Examples

```
lF<-NewLFxegaMigrate()
p<-xegaPopulation::xegaInitPopulation(10, lF)
p1<-xegaPopulation::xegaEvalPopulation(p, lF)
population<-p1$pop
fit<-p1$fit
p2<-xegaMigrate(population, fit, lF)
p2fit<-unlist(lapply(p2$pop, function(x) { x$fit }))
cat("Mean before:", mean(fit), "after migration:", mean(p2fit), "\n")
lF$pid<-xegaSelectGene::parm(6)
p3<-xegaMigrate(population, fit, lF)
p3fit<-unlist(lapply(p3$pop, function(x) { x$fit }))
cat("Mean before:", mean(p2fit), "after migration:", mean(p3fit), "\n")
```

---

xegaMigrationReport *Produce a migration report.*

Description

Produce a migration report.

Usage

```
xegaMigrationReport(msgSent = list(), msgReceived = list(), lF)
```

Arguments

msgSent	Messages sent.
msgReceived	Mesages received.
lF	Local function configuration and algorithm state.

Value

A list of named lists. Each named list has the form:

- $type: "I" indicates an immigrant, "e" an emigrant.
- $pid: pid.
- $tpid: frompid for an immigrant, topid for an emigrant.
- $iteration: Which generation?
- $timems: Time in Milliseconds (see base::Sys.time()).
- $fit: Fitness.
- $gene: The gene.

See Also

Other Reporting: xegaShowMigrationReport()

Examples

```
cat("TODO\n")
```

---

xegaProbeTermFactory *Factory for configuring probing for termination messages.*

Description

Avalailable methods:

1. "rds": Probing for termination rds-file.
2. "mpi": Probing for termination message via MPI.

Usage

```
xegaProbeTermFactory(method = "rds")
```

Arguments

method	Method. Default: "rds".

Value

A function for probing a termination message.

See Also

Other Configuration: xegaAdaptGenerationLimitFactory(), xegaBroadcastTermFactory(), xegaCollectFactory(), xegaCommunicationTopologyFactory(), xegaReceiveFactory(), xegaSendFactory()

Examples

```
xegaProbeTermFactory(method="rds")
```

---

xegaReceiveFactory *Factory for configuring the message receiving*

Description

Avalailable methods:

1. "rds": Message receiving via rds-file I/O. Non-blocking.
2. "rdsb": Message receiving via rds-file I/O. Barrier synchronization.
3. "mpi": Message receiving via mpi. Code with comments. Non-blocking.
4. "mpib": Message receiving via mpi. Code with comments. Barrier synchronization.

Usage

```
xegaReceiveFactory(method = "rds")
```

Arguments

method	Method. Default: "rds".

Value

A function for sending a message.

See Also

Other Configuration: xegaAdaptGenerationLimitFactory(), xegaBroadcastTermFactory(), xegaCollectFactory(), xegaCommunicationTopologyFactory(), xegaProbeTermFactory(), xegaSendFactory()

Examples

```
xegaReceiveFactory(method="rds")
```

xegaSendFactory *Factory for configuring the message sending*

Description

Avalailable methods:

1. "rds": Message sending genes via rds-file I/O.
2. "mpi": Message sending genes via MPI.

Usage

```
xegaSendFactory(method = "rds")
```

Arguments

method	Method. Default: "rds".

Value

A function for sending a message.

See Also

Other Configuration: xegaAdaptGenerationLimitFactory(), xegaBroadcastTermFactory(), xegaCollectFactory(), xegaCommunicationTopologyFactory(), xegaProbeTermFactory(), xegaReceiveFactory()

Examples

```
xegaSendFactory(method="rds")
```

xegaShowMigrationReport *Print migration report.*

Description

A migration report is a list of named lists. A named list has the following elements:

- $timems: Time in milliseconds.
- $iterations: Iteration number.
- $type: "E" for (E)migrant or "I" for (I)mmigrant.
- $pid: pid.

- $tpid: pid.
- $fit: Fitness.

For emigrants, the gene has been sent from $pid to $tpid.

For immigrants, the gene has been received by $pid from $tpid.

Usage

```
xegaShowMigrationReport(report)
```

Arguments

report	A migration report.

Details

Each gene which migrates occurs twice in a complete migration report: first as an emigrant and then as an immigrant.

The time resolution is supposed to help in ordering genes. However, no perfect order of messages sent and received can be established from the time stamp, because the time resolution is too coarse.

Value

Invisible 0.

See Also

Other Reporting: xegaMigrationReport()

Examples

```
msg2<-msg1<-list();
msg1$timems<-as.numeric(Sys.time())
msg1$iteration<-7; msg2$iteration<-6
msg1$type<-"I"; msg2$type<-"E"
msg1$pid<-0; msg2$pid<-0
msg1$tpid<-2; msg2$tpid<-3
msg1$fit<-27.33; msg2$fit<-26.49
msg2$timems<-as.numeric(Sys.time())
r<-list(); r[[1]]<-msg1; r[[2]]<-msg2
xegaShowMigrationReport(r)
```

Index