# Package ‘xegaPermGene’

May 8, 2026

**Title** Operations on Permutation Genes

**Version** 1.0.0.1

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

**Description** An implementation of
representation-dependent gene level operations for
genetic algorithms with genes representing permutations:
Initialization of genes, mutation, and crossover.
The crossover operation provided is position-based crossover
(Syswerda, G., Chap. 21 in Davis, L. (1991, ISBN:0-442-00173-8).
For mutation, several variants are included: Order-based mutation
(Syswerda, G., Chap. 21 in Davis, L. (1991, ISBN:0-442-00173-8),
randomized Lin-Kernighan heuristics
(Croes, G. A. (1958) <doi:10.1287/opre.6.6.791> and
Lin, S. and Kernighan. B. W. (1973)
<doi:10.1287/opre.21.2.498>),
and randomized greedy operators.
A random mix operator for mutation selects a mutation variant
randomly.

**URL** `https://github.com/ageyerschulz/xegaPermGene`

**Encoding** UTF-8

**RoxygenNote** 7.3.2

**Suggests** testthat (>= 3.0.0)

**Imports** xegaSelectGene

**NeedsCompilation** no

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

**Repository** CRAN

**Date/Publication** 2025-04-16 12:20:02 UTC

# Contents

---

## `Decay` *Exponential decay.*

### Description

Exponential decay.

### Usage

```
Decay(t, lambda = 0.05)
```

### Arguments

| | |
|---|---|
| `t` | Number of objects. |
| `lambda` | Exponential decay constant. |

### Value

Vector with t elements with values of exponential decay.

### See Also

Other Utility: `without()`

## Examples

```
Decay(5, 0.4)
Decay(10, 0.4)
```

---

# `lFxegaPermGene` *Generate local functions and objects.*

## Description

`lFxegaPermGene` is a list of functions which contains a definition of all local objects required for the use of genetic operators with the We refer to this object as local configuration.

## Usage

```
lFxegaPermGene
```

## Format

An object of class `list` of length 21.

## Details

We use the local function list (the local configuration) for

1. replacing all constants with constant functions.

   Rationale: We need one formal argument (the local function list lF) and we can dispatch multiple functions. E.g. `lF$verbose()`
2. for dynamically binding a local function with a definition from a proper function factory. E.g. the selection methods `lf$SelectGene` and `SelectMate`.
3. for gene representation specific special functions: `lf$InitGene`, `lF$DecodeGene`, `lf$EvalGene` `lf$ReplicateGene`, ...

## See Also

Other Configuration: `xegaPermCrossoverFactory()`, `xegaPermMutationFactory()`

---

## `without` *Returns elements of vector* `x` *without elements in* `y`*.*

---

### Description

Returns elements of vector `x` without elements in `y`.

### Usage

```
without(x, y)
```

### Arguments

| | |
|---|---|
| `x` | Vector. |
| `y` | Vector. |

### Value

Vector.

### See Also

Other Utility: `Decay()`

### Examples

```
a<-sample(1:15,15, replace=FALSE)
b<-c(1, 3, 5)
without(a, b)
```

---

## `xegaPermCross2Gene` *Position-based crossover of 2 genes.*

---

### Description

`xegaPermCross2Gene` determines a random subschedule of random length.

It copies the random subschedule into a new gene. The rest of the positions of the new scheme is filled with the elements of the other gene to complete the permutation. This is done for each gene.

### Usage

```
xegaPermCross2Gene(gg1, gg2, lF)
```

### Arguments

| | |
|---|---|
| `gg1` | Permutation. |
| `gg2` | Permutation. |
| `lF` | Local configuration of the genetic algorithm. |

### Value

List of 2 permutations.

### References

Syswerda, G. (1991): Schedule Optimization Using Genetic Algorithms. In: Davis, L. (Ed.): Handbook of Genetic Algorithms, Chapter 21, p. 343. Van Nostrand Reinhold, New York. (ISBN:0-442-00173-8)

### See Also

Other Crossover: `xegaPermCrossGene()`

### Examples

```
gene1<-xegaPermInitGene(lFxegaPermGene)
gene2<-xegaPermInitGene(lFxegaPermGene)
xegaPermDecodeGene(gene1, lFxegaPermGene)
xegaPermDecodeGene(gene2, lFxegaPermGene)
newgenes<-xegaPermCross2Gene(gene1, gene2)
xegaPermDecodeGene(newgenes[[1]], lFxegaPermGene)
xegaPermDecodeGene(newgenes[[2]], lFxegaPermGene)
```

---

## `xegaPermCrossGene` *Position-based crossover of 2 genes.*

### Description

`xegaPermCrossGene` determines a random subschedule of random length.

It copies the random subschedule into a new gene. The rest of the positions of the new scheme is filled with the elements of the other gene to complete the permutation.

### Usage

```
xegaPermCrossGene(gg1, gg2, lF)
```

### Arguments

| | |
|---|---|
| `gg1` | Permutation. |
| `gg2` | Permutation. |
| `lF` | Local configuration of the genetic algorithm. |

### Value

A list of 2 permutations.

### References

Syswerda, G. (1991): Schedule Optimization Using Genetic Algorithms. In: Davis, L. (Ed.): Handbook of Genetic Algorithms, Chapter 21, p. 343. Van Nostrand Reinhold, New York. (ISBN:0-442-00173-8)

### See Also

Other Crossover: `xegaPermCross2Gene()`

### Examples

```
gene1<-xegaPermInitGene(lFxegaPermGene)
gene2<-xegaPermInitGene(lFxegaPermGene)
xegaPermDecodeGene(gene1, lFxegaPermGene)
xegaPermDecodeGene(gene2, lFxegaPermGene)
newgenes<-xegaPermCrossGene(gene1, gene2)
xegaPermDecodeGene(newgenes[[1]], lFxegaPermGene)
```

---

## `xegaPermCrossoverFactory`

*Configure the crossover function of a genetic algorithm.*

---

### Description

`xegaPermCrossoverFactory` implements the selection of one of the crossover functions in this package by specifying a text string. The selection fails ungracefully (produces a runtime error) if the label does not match. The functions are specified locally.

Current support:

1. Crossover functions with two kids:
   (a) "Cross2Gene" returns `xegaPermCross2Gene`.
2. Crossover functions with one kid:
   (a) "CrossGene" returns `xegaPermCrossGene`.

### Usage

```
xegaPermCrossoverFactory(method = "Cross2Gene")
```

### Arguments

| | |
|---|---|
| `method` | A string specifying the crossover function. |

### Value

A crossover function for genes.

### See Also

Other Configuration: `lFxegaPermGene`, `xegaPermMutationFactory()`

### Examples

```
XGene<-xegaPermCrossoverFactory("Cross2Gene")
gene1<-xegaPermInitGene(lFxegaPermGene)
gene2<-xegaPermInitGene(lFxegaPermGene)
XGene(gene1, gene2, lFxegaPermGene)
```

---

## `xegaPermDecodeGene` *Decode a permutation.*

### Description

`xegaPermDecodeGene` decodes a permutation gene.

### Usage

```
xegaPermDecodeGene(gene, lF)
```

### Arguments

| | |
|---|---|
| `gene` | Permutation. |
| `lF` | Local configuration of the genetic algorithm. |

### Details

`xegaPermDecodeGene` is the identy function.

### Value

A permutation gene.

### Examples

```
g<-xegaPermInitGene(lFxegaPermGene)
xegaPermDecodeGene(g)
```

## `xegaPermGene` *Package xegaPermGene.*

### Description

Genetic operations for permutation genes.

### Details

Permutation genes are a representation of a tour of a Traveling Salesman Problem (TSP).

For permutation genes, the `xegaPermGene` package provides

- Gene initiatilization.
- Decoding of parameters.
- Mutation functions as well as a function factory for configuration.
- Crossover functions as well as a function factory for configuration.

### Permutation Gene Representation

A permutation gene is a named list with at least the following elements:

- `$gene1`: The gene must be a permutation vector.
- `$fit`: The fitness value of the gene (for EvalGeneDet and EvalGeneU) or the mean fitness (for stochastic functions evaluated with EvalGeneStoch).
- `$evaluated`: Boolean. Has the gene been evaluated?
- `$evalFail`: Boolean. Has the evaluation of the gene failed?

### Abstract Interface of a Problem Environment for the TSP

A problem environment `penv` for the TSP must provide:

- `$name()`: Returns the name of the problem environment.
- `$genelength()`: The number of integers of a permutation. Used in `InitGene`.
- `$dist()`: The distance matrix of the TSP.
- `$cities()`: A list of city names or `1:numberOfCities`.
- `$f(permutation, gene, lF)`: Returns the fitness of the permutation (the length of a tour).
- `$solution()`: The minimal tour length (if known).
- `$path()`: An optimal TSP tour.
- `$show(permutation)`: Prints the tour with the distances and the cumulative distances between the cities.
- TSP Heuristics:
  - `$greedy(startposition, k)`: Computes a greedy tour of length k.
  - `$kBestgreedy(k)`: Computes the best greedy tour of length k.

  - `$rnd2Opt(permutation, maxTries)`: Generate a new permutation by a random 2-change. `maxTries` is the maximal number of trials to find a better permutation. `$rnd2Opt` either returns a better permutation or, if no better permutation can be found in `maxTries` attempts, the original permutation.
  - `$LinKernighan(permutation, maxTries)`: Returns a permutation generated by a random sequence of 2-changes with improving performance. The optimality criterion of the k Lin-Kernighan heuristics is replaced by the necessity of finding a sequence of random 2-changes with strictly increasing performance.

### Abstract Interface of Mutation Functions

Each mutation function has the following function signature:

`newGene<-Mutate(gene, lF)`

All local parameters of the mutation function configured are expected in the local configuration `lF`.

### Local Constants of Mutation Functions

The local constants of a mutation function determine the behavior of the function.

| **Constant** | **Default** | **Used in** |
|---:|:---:|:---|
| `lF$BitMutationRate1()` | 0.005 | xegaPermMutateGeneOrderBased |
| `lF$Lambda()` | 0.05 | xegaPermMutateGenekInversion<br>xegaPermMutateGenekGreedy<br>xegaPermMutateGeneBestGreedy |
| `lF$max2Opt()` | 100 | xegaPermMutateGene2Opt<br>xegaPermMutateGenekOptLK |

### Abstract Interface of Crossover Functions

The signatures of the abstract interface to the 2 families of crossover functions are:

`ListOfTwoGenes<-Crossover2(gene1, gene2, lF)`

`newGene<-Crossover(gene1, gene2, lF)`

### The Architecture of the xegaX-Packages

The xegaX-packages are a family of R-packages which implement eXtended Evolutionary and Genetic Algorithms (xega). The architecture has 3 layers, namely the user interface layer, the population layer, and the gene layer:

- The user interface layer (package `xega`) provides a function call interface and configuration support for several algorithms: genetic algorithms (sga), permutation-based genetic algorithms (sgPerm), derivation-free algorithms as e.g. differential evolution (sgde), grammar-based genetic programming (sgp) and grammatical evolution (sge).
- The population layer (package `xegaPopulation`) contains population-related functionality as well as support for population statistics dependent adaptive mechanisms and parallelization.
- The gene layer is split into a representation-independent and a representation-dependent part:

1. The representation-indendent part (package `xegaSelectGene`) is responsible for variants of selection operators, evaluation strategies for genes, as well as profiling and timing capabilities.
2. The representation-dependent part consists of the following packages:
   - `xegaGaGene` for binary coded genetic algorithms.
   - `xegaPermGene` for permutation-based genetic algorithms.
   - `xegaDfGene` for derivation-free algorithms as e.g. differential evolution.
   - `xegaGpGene` for grammar-based genetic algorithms.
   - `xegaGeGene` for grammatical evolution algorithms.

   The packages `xegaDerivationTrees` and `xegaBNF` support the last two packages: `xegaBNF` essentially provides a grammar compiler and `xegaDerivationTrees` is an abstract data type for derivation trees.

## Copyright

## License

## URL

<https://github.com/ageyerschulz/xegaPermGene>

## Installation

From CRAN by `install.packages('xegaPermGene')`

## Author(s)

Andreas Geyer-Schulz

## See Also

Useful links:

- `https://github.com/ageyerschulz/xegaPermGene`

---

## `xegaPermInitGene` *Initialize a gene with a permutation of integers*

---

### Description

`xegaPermInitGene` generates a random permutation with a given length n.

### Usage

```
xegaPermInitGene(lF)
```

### Arguments

| | |
|---|---|
| `lF` | Local configuration of the genetic algorithm. |

### Details

In the permutation representation of package `xegaPerm`, a *gene* is a list with

1. `$evaluated`: Boolean: TRUE if the fitness is known.
2. `$fit`: The fitness of the genotype of `$gene1`.
3. `$gene1`: The permutation (the genotype).

### Value

A permutation gene.

### Examples

```
xegaPermInitGene(lFxegaPermGene)
```

---

## `xegaPermMutateGene2Opt` *Mutate a gene (by a random 2-Opt move).*

---

### Description

`xegaPermMutateGene2Opt` mutates a permutation.

### Usage

```
xegaPermMutateGene2Opt(gene, lF)
```

### Arguments

| | |
|---|---|
| `gene` | A Permutation. |
| `lF` | Local configuration of the genetic algorithm. |

### Details

This operator is an implementation of the 2-Opt move due to Croes (1958).

Two edges are exchanged, if the exchange improves the result.

### Value

A Permutation.

### References

Croes, G. A. (1958): A Method for Solving Traveling-Salesman Problems. Operations Research, 6(6), pp. 791-812. <doi:10.1287/opre.6.6.791>

### See Also

Other Mutation: `xegaPermMutateGeneBestGreedy()`, `xegaPermMutateGeneGreedy()`, `xegaPermMutateGeneOrderBased` `xegaPermMutateGenekInversion()`, `xegaPermMutateGenekOptLK()`, `xegaPermMutateMix()`

### Examples

```
gene1<-xegaPermInitGene(lFxegaPermGene)
xegaPermDecodeGene(gene1, lFxegaPermGene)
gene<-xegaPermMutateGene2Opt(gene1, lFxegaPermGene)
xegaPermDecodeGene(gene, lFxegaPermGene)
```

---

## `xegaPermMutateGeneBestGreedy`

*Mutate a gene (by inserting the best greedy path at a random start position with a random length of k).*

### Description

`xegaPermMutateGeneBestGreedy` mutates a permutation by inserting the best greedy path of length `k` at a random position `start`.

### Usage

```
xegaPermMutateGeneBestGreedy(gene, lF)
```

### Arguments

| | |
|---|---|
| `gene` | A Permutation. |
| `lF` | Local configuration of the genetic algorithm. |

## Details

The path length `k` is exponentially decaying with exponential decay constant `lF$lambda()`.

## Value

A Permutation

## See Also

Other Mutation: `xegaPermMutateGene2Opt()`, `xegaPermMutateGeneGreedy()`, `xegaPermMutateGeneOrderBased()`, `xegaPermMutateGenekInversion()`, `xegaPermMutateGenekOptLK()`, `xegaPermMutateMix()`

## Examples

```
gene1<-xegaPermInitGene(lFxegaPermGene)
xegaPermDecodeGene(gene1, lFxegaPermGene)
gene<-xegaPermMutateGeneGreedy(gene1, lFxegaPermGene)
xegaPermDecodeGene(gene, lFxegaPermGene)
```

---

# `xegaPermMutateGeneGreedy` *Mutate a gene (by inserting a greedy path at a random start position with a random length of k).*

## Description

`xegaPermMutateGeneGreedy` mutates a permutation by inserting a greedy path of length `k` at a random position `start`.

## Usage

```
xegaPermMutateGeneGreedy(gene, lF)
```

## Arguments

| | |
|---|---|
| `gene` | A Permutation. |
| `lF` | Local configuration of the genetic algorithm. |

## Details

The path length `k` is exponentially decaying with exponential decay constant `lambda`.

## Value

A Permutation.

### See Also

Other Mutation: `xegaPermMutateGene2Opt()`, `xegaPermMutateGeneBestGreedy()`, `xegaPermMutateGeneOrderBased()` `xegaPermMutateGenekInversion()`, `xegaPermMutateGenekOptLK()`, `xegaPermMutateMix()`

### Examples

```
gene1<-xegaPermInitGene(lFxegaPermGene)
xegaPermDecodeGene(gene1, lFxegaPermGene)
gene<-xegaPermMutateGeneGreedy(gene1, lFxegaPermGene)
xegaPermDecodeGene(gene, lFxegaPermGene)
```

---

## `xegaPermMutateGenekInversion` *Mutate a gene (k random inversions).*

### Description

`xegaPermMutateGenekInversion` performs k random inversions. The number of inversions is exponentially decaying with exponential decay constant `lambda`.

### Usage

```
xegaPermMutateGenekInversion(gene, lF)
```

### Arguments

| | |
|---|---|
| `gene` | A Permutation. |
| `lF` | Local configuration of the genetic algorithm. |

### Details

The only difference to the order-based mutation operator (Syswerda, 1991) is the exponential decay in the number of inversions.

1. The indices of a random subschedule are extracted.
2. The subschedule is extracted, permuted, and reinserted.

### Value

A Permutation.

### References

Syswerda, G. (1991): Schedule Optimization Using Genetic Algorithms. In: Davis, L. (Ed.): Handbook of Genetic Algorithms, Chapter 21, pp. 332-349. Van Nostrand Reinhold, New York.

## See Also

Other Mutation: `xegaPermMutateGene2Opt()`, `xegaPermMutateGeneBestGreedy()`, `xegaPermMutateGeneGreedy()`, `xegaPermMutateGeneOrderBased()`, `xegaPermMutateGenekOptLK()`, `xegaPermMutateMix()`

## Examples

```
gene1<-xegaPermInitGene(lFxegaPermGene)
xegaPermDecodeGene(gene1, lFxegaPermGene)
gene<-xegaPermMutateGenekInversion(gene1, lFxegaPermGene)
xegaPermDecodeGene(gene, lFxegaPermGene)
```

---

# `xegaPermMutateGenekOptLK`

*Mutate a gene (by a random Lin-Kernighan k-OPT move).*

## Description

`xegaPermMutateGenekOptLK` mutates a permutation.

## Usage

```
xegaPermMutateGenekOptLK(gene, lF)
```

## Arguments

| | |
|---|---|
| `gene` | A Permutation. |
| `lF` | Local configuration of the genetic algorithm. |

## Details

This operator implements a random k-Opt move version of the Lin-Kernighan heuristic.

A sequence of random 2-Opt moves, all of which improve the result is executed.

## Value

A Permutation.

## References

Lin, S. and Kernighan. B. W. (1973): An Effective Heuristic Algorithm for the Traveling-Salesman Problem. Operations Research, 21(2), pp. 791-812. <doi:10.1287/opre.21.2.498>

## See Also

Other Mutation: `xegaPermMutateGene2Opt()`, `xegaPermMutateGeneBestGreedy()`, `xegaPermMutateGeneGreedy()`, `xegaPermMutateGeneOrderBased()`, `xegaPermMutateGenekInversion()`, `xegaPermMutateMix()`

## Examples

```
gene1<-xegaPermInitGene(lFxegaPermGene)
xegaPermDecodeGene(gene1, lFxegaPermGene)
gene<-xegaPermMutateGenekOptLK(gene1, lFxegaPermGene)
xegaPermDecodeGene(gene, lFxegaPermGene)
```

---

# `xegaPermMutateGeneOrderBased`

*Mutate a gene (generalized order based mutation).*

---

## Description

`xegaPermMutateGene` mutates a permutation. The per-position mutation rate is given by `lF$BitMutationRate1()`.

## Usage

```
xegaPermMutateGeneOrderBased(gene, lF)
```

## Arguments

| | |
|---|---|
| `gene` | A Permutation. |
| `lF` | Local configuration of the genetic algorithm. |

## Details

This operator implements a generalized order based mutation operator (Syswerda, 1991).

1. The indices of a random subschedule are extracted.
2. The subschedule is extracted, permuted, and reinserted.

## Value

A Permutation.

## References

Syswerda, G. (1991): Schedule Optimization Using Genetic Algorithms. In: Davis, L. (Ed.): Handbook of Genetic Algorithms, Chapter 21, pp. 332-349. Van Nostrand Reinhold, New York. (ISBN:0-442-00173-8)

## See Also

Other Mutation: `xegaPermMutateGene2Opt()`, `xegaPermMutateGeneBestGreedy()`, `xegaPermMutateGeneGreedy()`, `xegaPermMutateGenekInversion()`, `xegaPermMutateGenekOptLK()`, `xegaPermMutateMix()`

## Examples

```
gene1<-xegaPermInitGene(lFxegaPermGene)
xegaPermDecodeGene(gene1, lFxegaPermGene)
gene<-xegaPermMutateGeneOrderBased(gene1, lFxegaPermGene)
xegaPermDecodeGene(gene, lFxegaPermGene)
```

---

# `xegaPermMutateMix` *Mutation by a random mutation function.*

## Description

A mutation function is randomly selected from the following list: xegaPermMutateGeneOrderBased, xegaPermMutateGenekInversion, xegaPermMutateGene2Opt, xegaPermMutateGenekOptLK, xegaPermMutateGeneGreedy, xegaPermMutateGeneBestGreedy.

## Usage

```
xegaPermMutateMix(gene, lF)
```

## Arguments

| | |
|---|---|
| `gene` | A permutation. |
| `lF` | Local configuration. |

## Value

A permutation.

## See Also

Other Mutation: `xegaPermMutateGene2Opt()`, `xegaPermMutateGeneBestGreedy()`, `xegaPermMutateGeneGreedy()`, `xegaPermMutateGeneOrderBased()`, `xegaPermMutateGenekInversion()`, `xegaPermMutateGenekOptLK()`

## Examples

```
gene1<-xegaPermInitGene(lFxegaPermGene)
xegaPermDecodeGene(gene1, lFxegaPermGene)
gene<-xegaPermMutateMix(gene1, lFxegaPermGene)
xegaPermDecodeGene(gene, lFxegaPermGene)
```

---

## `xegaPermMutationFactory` *Configure the mutation function of a genetic algorithm.*

---

### Description

`xegaPermMutationFactory` implements the selection of one of the gene mutation functions in this package by specifying a text string. The selection fails ungracefully (produces a runtime error) if the label does not match. The functions are specified locally.

Current Support:

1. "MutateGene" returns `xegaPermMutateGeneOrderBased`.
2. "MutateGeneOrderBased" returns `xegaPermMutateGeneOrderBased`.
3. "MutateGenekInversion" returns `xegaPermMutateGenekInversion`.
4. "MutateGene2Opt" returns `xegaPermMutateGene2Opt`.
5. "MutateGenekOptLK" returns `xegaPermMutateGenekOptLK`.
6. "MutateGeneGreedy" returns `xegaPermMutateGeneGreedy`.
7. "MutateGeneBestGreedy" returns `xegaPermMutateGeneBestGreedy`.
8. "MutateGeneMix" returns `xegaPermMutateMix`.

### Usage

```
xegaPermMutationFactory(method = "MutateGene")
```

### Arguments

| | |
|---|---|
| `method` | The name of the mutation method. |

### Value

A permutation based mutation function.

### See Also

Other Configuration: `lFxegaPermGene`, `xegaPermCrossoverFactory()`

### Examples

```
xegaPermMutationFactory(method="MutateGene")
```

# Index