Package ‘mcMST’

September 18, 2017

Title A Toolbox for the Multi-Criteria Minimum Spanning Tree Problem

Description Algorithms to approximate the Pareto-front of multi-criteria minimum spanning tree problems. Additionally, a modular toolbox for the generation of multi-objective benchmark graph problems is included.

Version 1.0.1

Encoding UTF-8

Date 2017-09-13

Maintainer Jakob Bossek <j.bossek@gmail.com>

License BSD_2_clause + file LICENSE

URL https://github.com/jakobbossek/mcMST

BugReports https://github.com/jakobbossek/mcMST/issues

Depends BBmisc (>= 1.6), ecr (>= 2.1.0)

Imports checkmate (>= 1.1), parallelMap (>= 1.3), reshape2 (>= 1.4.1), gtools, vegan, ggplot2 (>= 1.0.0), lhs

Suggests testthat (>= 0.9.1), knitr, rmarkdown, gridExtra

ByteCompile yes

LazyData yes

RoxygenNote 6.0.1

VignetteBuilder knitr

NeedsCompilation no

Author Jakob Bossek [aut, cre]

Repository CRAN

Date/Publication 2017-09-18 09:46:57 UTC

R topics documented:

`mcMST-package` *mcMST: A Toolbox for the Multi-Criteria Minimum Spanning Tree Problem.*

Description

The **mcMST** package provides a set of algorithms to approximate the Pareto-optimal front of multi-criteria minimum spanning tree (mcMST) problems. Besides, the package contains a modular toolbox for benchmark problem generation.

Algorithms

Currently, the following algorithms are included:

mcPrim A multi-criteria version of Prim's algorithm for the single-objective MST (see [1]).

ZhouEmoa Evolutionary multi-objective algorithm operating on the Pruefer-encoding as proposed by Zhou and Gen [2].

BGEmoa Evolutionary multi-objective algorithm operating on a direct edge list encoding. This algorithm applies a sub-tree based mutation operator as proposed by Bossek and Grimme [3].

Exhaustive Enumeration A simple method to enumerate all Pareto-optimal solutions of a given combinatorial problem. This method is not limited to mcMST problems.

References

[1] Knowles, J. D., and Corne, D. W. 2001. A Comparison of Encodings and Algorithms for Multi-objective Minimum Spanning Tree Problems. In Proceedings of the 2001 Congress on Evolutionary Computation (Ieee Cat. No.01TH8546), 1:544–51 vol. 1. doi:10.1109/CEC.2001.934439.

[2] Zhou, G., and Gen, M. 1999. Genetic Algorithm Approach on Multi-Criteria Minimum Spanning Tree Problem. European Journal of Operational Research 114 (1): 141–52. doi:https://doi.org/10.1016/S0377-2217(98)00016-2.

[3] Bossek, J., and Grimme, C. 2017. A Pareto-Beneficial Sub-Tree Mutation for the Multi-Criteria Minimum Spanning Tree Problem. In Proceedings of the 2017 IEEE Symposium Series on Computational Intelligence. (accepted)

`addCenters` *Add cluster centers to graph.*

Description

Places `n.centers` cluster centers in the two-dimensional euclidean plane by means of a `generator`, e.g., by Latin-Hypercube-Sampling (LHS).

Usage

```
addCenters(graph, n.centers = NULL, center.coordinates = NULL,
  generator = NULL, ...)
```

Arguments

`graph`	`[mcGP]` Multi-objective graph problem.
`n.centers`	`[integer(1)]` Number of cluster centers.
`center.coordinates`	`[matrix(n, 2)]` Matrix of center coordinates (each row is one point). Default is `NULL`. If this is set, `n.centers` and `generator` are ignored.
`generator`	`[function(n, ...)]` Function used to generate cluster centers. The generator needs to expect the number of points to generate as the first argument `n`. Additional control argument are possible.
`...`	[any] Additional arguments passed down to `generator`.

Value

`mcGP` Multi-objective graph problem.

See Also

Other graph generators: addCoordinates, addWeights, mcGP

---

addCoordinates *Add node coordinates to graph.*

Description

Places node coordinates in the two-dimensional euclidean plane.

Usage

```
addCoordinates(graph, n, generator, by.centers = FALSE, par.fun = NULL, ...)
```

Arguments

graph	[mcGP] Multi-objective graph problem.
n	[integer] Number of coordinates to place. If `by.centers` is `FALSE` a single integer value is expected. Otherwise, a vector v may be passed. In this case v[i] coordinates are generated for each cluster. However, if a single value is passed and `by.center == TRUE`, each cluster is assigned the same number of coordinates.
generator	[function(n, ...)] Function used to generate coordinates. The generator needs to expect the number of points to generate as the first argument `n`. Additional control argument are possible.
by.centers	[logical(1)] Should coordinates be placed for each cluster center seperately? This enables geneation of clustered coordinates. Default is `FALSE`.
par.fun	[function(cc) | NULL] Optional function which is applied to each cluster center `cc` before the generation of coordinates in case `by.centers` is `TRUE`. This enables to specifically determine additional parameters for the `generator` for each cluster.
...	[any] Furhter arguments passed down to `generator`.

Value

mcGP Multi-objective graph problem.

See Also

Other graph generators: addCenters, addWeights, mcGP addWeights *Add weights to a multi-objective graph.*

Description

addWeights allows to generate edge weights for a multi-objective graph instance. The weights can be generated on basis of the node coordinates (in this case dist is applied with the cooresponding method). Alternatively, all kinds of random weights can be generated.

Usage

```
addWeights(graph, method = "euclidean", weights = NULL, weight.fun = NULL,
  n = NULL, symmetric = TRUE, ...)
```

Arguments

graph	[mcGP] Multi-objective graph problem.
method	[character(1)] Method used to generate weights. Possible values are "euclidean", "maximum", "manhatten", "canberra", "binary", minkowski or random. The latter generates (random) weights utilizing weight.fun. The remaining options are passed down to dist, i.e., weights are generated as distances between the node coordinates.
weights	[matrix] Square matrix of weights. If some weights are already assigned, pay attention to the correct dimensions. If this is passed all other arguments are ignored. Default is NULL.
weight.fun	[function(m, ...) | NULL] Function used to generate weights. The first arument needs to be number of weights to generate.
n	[integer(1)] Number of nodes. This is required only if there are no coordinates or no weights until now. Default is NULL, i.e., the number of nodes is extracted from graph.
symmetric	[logical(1)] Should the weights be symmetric, i.e., w(i, j) = w(j, i) for each pair i, j of nodes? Default is TRUE.
...	[any] Additional arguments passed down to weight.fun or dist. See documentation of argument method for details.

Value

mcGP Multi-objective graph problem.

See Also

Other graph generators: addCenters, addCoordinates, mcGP

---

coordGenerators *Coordinate generators.*

Description

Functions for the placement of node coordinates in the euclidean plane. Function `coordLHS` generates a space-filling latin hypercube sample, `coordUniform` samples points from a bivariate uniform distribution and `coordGrid` generates a regular grid of points.

Usage

```
coordLHS(n, lower = 0, upper = 1, method = NULL)

coordUniform(n, lower, upper)

coordGrid(n, lower, upper)
```

Arguments

n	[integer(1)] Number of points to generate.
lower	[numeric(2)] Minimal values for the first and second coordinates respectively. Default is 0.
upper	[numeric(2)] Maximal values for the first and second coordinates respectively. Default is 1.
method	[function] Function from package **lhs**. Default is maximinLHS.

Value

`matrix(n, 2)` Matrix of node coordinates.

edgeListToCharVec — *Convert edge list to characteristic vector.*

Description

Convert edge list to characteristic vector.

Usage

```
edgeListToCharVec(edgelist, n = NULL)
```

Arguments

`edgelist`	`[matrix(2, k)]` Matrix of edges (each column is one edge).
`n`	`[integer]` Number of nodes of the problem.

Value

`integer` Characteristic vector cv with cv[i] = 1 if the i-th edge is in the tree.

See Also

Other transformation functions: `permutationToCharVec`, `permutationToEdgelist`, `prueferToCharVec`, `prueferToEdgeList`

Examples

```
# first we generate a small edge list by hand
# (assume the given graph has n = 4 nodes)
edgelist = matrix(c(1, 2, 2, 4, 3, 4), ncol = 3)
print(edgelist)
# next we transform the edge into
# a characteristic vector
cvec = edgeListToCharVec(edgelist, n = 4)
print(cvec)
```

enumerateTSP — *Enumerate all solution candidates.*

Description

These functions enumerate all candidate solutions for a certain combinatorial optimization problem, e.g., all permutations for a TSP or all Pruefer-codes for a MST problem. Note that the output grows exponentially with the instance size `n`.

Usage

```
enumerateTSP(n)

enumerateMST(n)
```

Arguments

n	[integer(1)] Instance size.

Value

matrix Each row contains a candidate solution.

Examples

```
sols = enumerateTSP(4L)
sols = enumerateMST(4L)
```

---

genRandomMCGP *Generate a bi-criteria graph with uniformly randomly distribted edge weights.*

Description

No topology is defined. The instance is composed of two symmetric weight matrices. The first weight is drawn independently at random from a $\mathcal{R}[10, 100]$ distribution, the second one from a $\mathcal{R}[10, 50]$ distribution (see references).

Usage

```
genRandomMCGP(n)
```

Arguments

n	[integer(1)] Instance size, i.e., number of nodes.

Value

mcGP

Note

This is a simple wrapper around the much more flexible graph generation system (see, e.g., mcGP).

References

Zhou, G. and Gen, M. Genetic Algorithm Approach on Multi-Criteria Minimum Spanning Tree Problem. In: European Journal of Operational Research (1999).

Knowles, JD & Corne, DW 2001, A comparison of encodings and algorithms for multiobjective minimum spanning tree problems. in Proceedings of the IEEE Conference on Evolutionary Computation, ICEC|Proc IEEE Conf Evol Comput Proc ICEC. vol. 1, Institute of Electrical and Electronics Engineers , pp. 544-551, Congress on Evolutionary Computation 2001, Soul, 1 July.

Examples

```
g = genRandomMCGP(10L)
## Not run:
pl = plot(g)

## End(Not run)
```

`getExactFront` *Enumerate all Pareto-optimal solutions.*

Description

Function which expects a problem instance of a combinatorial optimization problem (e.g., MST), a multi-objective function and a solution enumerator, i.e., a function which enumerates all possible solutions (e.g., all Pruefer codes in case of a MST problem) and determines both the Pareto front and Pareto set by exhaustive enumeration.

Usage

```
getExactFront(instance, obj.fun, enumerator.fun, n.objectives)
```

Arguments

`instance`	[any] Problem instance.
`obj.fun`	[function(solution, instance)] Objective function which expects a numeric vector `solution` encoding a solution candidate and a problem instance `instance`. The function should return a numeric vector of length `n.objectives`.
`enumerator.fun`	[function(n)] Function to exhaustively generate all possible candidate solutions. Expects a single integer value n, i.e., the instance size, e.g., the number of nodes for a graph problem.
`n.objectives`	[integer(1)] Number of objectives of problem.

Value

`list` List with elements `pareto.set` (matrix of Pareto-optimal solutions) and `pareto.front` (matrix of corresponding weight vectors).

Note

This method exhaustively enumerates all possible solutions of a given multi-objective combinatorial optimization problem. Thus, it is limited to small input size due to combinatorial explosion.

Examples

```
# here we enumerate all Pareto-optimal solutions of a bi-objective mcMST problem
# we use the Pruefer-code enumerator. Thus, we need to define an objective
# function, which is able to handle this type of endcoding
objfunMCMST = function(pcode, instance) {
  getWeight(instance, prueferToEdgeList(pcode))
}

# next we generate a random bi-objective graph
g = genRandomMCGP(5L)

# ... and finally compute the exact front of g
res = getExactFront(g, obj.fun = objfunMCMST, enumerator.fun = enumerateMST, n.objectives = 2L)
## Not run:
plot(res$pareto.front)

## End(Not run)
```

`getWeight` *Get the overall costs/weight of a subgraph given its edgelist.*

Description

Get the overall costs/weight of a subgraph given its edgelist.

Usage

```
getWeight(graph, edgelist)
```

Arguments

`graph`	[mcGP] Multi-objective graph problem.
`edgelist`	[`matrix(2, k)`] Matrix of edges (each column is one edge).

Value

`numeric(2)` Weight vector.

Examples

```
# generate a random bi-objective graph
g = genRandomMCGP(5)

# generate a random Pruefer code, i.e., a random spanning tree of g
pcode = sample(1:5, 3, replace = TRUE)

getWeight(g, prueferToEdgeList(pcode))
```

`mcGP` *Generate a bare multi-objective graph.*

Description

This function generates a bare multi-objective weights. The generated object does not contain nodes, edges or edge weights. It serves as a starting point for the step-by-step construction of multi-objective graph problem.

Usage

```
mcGP(lower, upper)
```

Arguments

`lower`	`[integer(1)]` Lower bounds for coordinates.
`upper`	`[integer(1)]` Upper bounds for coordinates.

Value

`mcGP` Multi-objective graph problem.

See Also

Other graph generators: `addCenters`, `addCoordinates`, `addWeights` mcMSTEmoaBG *Subgraph EMOA for the multi-criteria MST problem.*

Description

Evolutionary multi-objective algorithm to solve the multi-objective minimum spanning tree problem. The algorithm relies to mutation only to generate offspring. The package contains the subgraph mutator (see mutSubgraphMST) or a simple one-edge exchange mutator (see mutEdgeExchange). Of course, the user may use any custom mutator which operators on edge lists as well (see makeMutator).

Usage

```
mcMSTEmoaBG(instance, mu, lambda = mu, mut = NULL, selMating = NULL,
  selSurvival = ecr::selNondom, ref.point = NULL, max.iter = 100L)
```

Arguments

instance	[mcGP] Multi-objective graph problem.
mu	[integer(1)] Population size.
lambda	[integer(1)] Number of offspring generated in each generation. Default is mu.
mut	[ecr_mutator] Mutation operator. Default is mutSubgraphMST.
selMating	[ecr_selector] Mating selector. Default is selSimple.
selSurvival	[ecr_selector] Survival selector. Default is link[ecr]{selNondom}.
ref.point	[numeric(n.objectives) | NULL] Reference point for hypervolume computation used for logging. If NULL the sum of the n largest edges in each objective is used where n is the number of nodes of instance. This is an upper bound for the size of each spanning tree with $(n-1)$ edges.
max.iter	[integer(1)] Maximal number of iterations. Default is 100.

Value

ecr_result List of type ecr_result with the following components:

task The ecr_optimization_task.

log Logger object.

pareto.idx Indizes of the non-dominated solutions in the last population.

pareto.front (n x d) matrix of the approximated non-dominated front where n is the number of non-dominated points and d is the number of objectives.

pareto.set Matrix of decision space values resulting with objective values given in pareto.front.

last.population Last population.

message Character string describing the reason of termination.

References

Bossek, J., and Grimme, C. A Pareto-Beneficial Sub-Tree Mutation for the Multi-Criteria Minimum Spanning Tree Problem. In Proceedings of the 2017 IEEE Symposium Series on Computational Intelligence (2017). (accepted)

See Also

Mutators `mutSubgraphMST` and `mutEdgeExchange`

Other mcMST EMOAs: `mcMSTEmoaZhou`

Other mcMST algorithms: `mcMSTEmoaZhou`, `mcMSTPrim`

Examples

```
inst = genRandomMCGP(10)
res = mcMSTEmoaBG(inst, mu = 20L, max.iter = 100L)
print(res$pareto.front)
print(tail(getStatistics(res$log)))
```

`mcMSTEmoaZhou` *Pruefer-EMOA for the multi-objective MST problem.*

Description

Evolutionary multi-objective algorithm to solve the multi-objective minimum spanning tree problem. The algorithm adopts the so-called Pruefer-number as the encoding for spanning trees. A Pruefer-number for a graph with nodes $V = \{1, \ldots, n\}$ is a sequence of $n - 2$ numbers from V. Cayleys theorem states, that a complete graph width n nodes has exactly n^{n-2} spanning trees. The algorithm uses mutation only: each component of an individual is replaced uniformly at random with another node number from the node set.

Usage

```
mcMSTEmoaZhou(instance, mu, lambda = mu, mut = mutUniformPruefer,
  selMating = ecr::selSimple, selSurvival = ecr::selNondom,
  ref.point = NULL, max.iter = 100L)
```

Arguments

`instance`	[mcGP] Multi-objective graph problem.
`mu`	[integer(1)] Population size.
`lambda`	[integer(1)] Number of offspring generated in each generation. Default is `mu`.
`mut`	[ecr_mutator] Mutation operator. Defaults to `mutUniformPruefer`, i.e., each digit of the Pruefer encoding is replaced with some probability with a random number from $V = \{1, \ldots, n\}$.
`selMating`	[ecr_selector] Mating selector. Default is `selSimple`.
`selSurvival`	[ecr_selector] Survival selector. Default is `link[ecr]{selNondom}`.
`ref.point`	[numeric(n.objectives) | NULL] Reference point for hypervolume computation used for logging. If `NULL` the sum of the n largest edges in each objective is used where n is the number of nodes of `instance`. This is an upper bound for the size of each spanning tree with $(n - 1)$ edges.
`max.iter`	[integer(1)] Maximal number of iterations. Default is `100`.

Value

`ecr_result` List of type `ecr_result` with the following components:

task The `ecr_optimization_task`.

log Logger object.

pareto.idx Indizes of the non-dominated solutions in the last population.

pareto.front (n x d) matrix of the approximated non-dominated front where n is the number of non-dominated points and d is the number of objectives.

pareto.set Matrix of decision space values resulting with objective values given in pareto.front.

last.population Last population.

message Character string describing the reason of termination.

References

Zhou, G. and Gen, M. Genetic Algorithm Approach on Multi-Criteria Minimum Spanning Tree Problem. In: European Journal of Operational Research (1999).

See Also

Mutator `mutUniformPruefer`

Other mcMST EMOAs: `mcMSTEmoaBG`

Other mcMST algorithms: `mcMSTEmoaBG`, `mcMSTPrim`

`mcMSTPrim` *Multi-Objective Prim algorithm.*

Description

Approximates the Pareto-optimal mcMST front of a multi-objective graph problem by iteratively applying Prim's algorithm for the single-objective MST problem to a scalarized version of the problem. I.e., the weight vector (w_1, w_2) of an edge (i, j) is substituted with a weighted sum $\lambda_i w_1 + (1 - \lambda_i) w_2$ with weight $\lambda_i \in [0, 1]$ for different weights.

Usage

```
mcMSTPrim(instance, n.lambdas = NULL, lambdas = NULL)
```

Arguments

`instance`	`[mcGP]` Multi-objective graph problem.
`n.lambdas`	`[integer(1) | NULL]` Number of weights to generate. The weights are generated equdistantly in the interval $[0, 1]$.
`lambdas`	`[numerci]` Vector of weights. This is an alternative to `n.lambdas`.

Value

`list` List with component `pareto.front`.

Note

Note that this procedure can only find socalled supported efficient solutions, i.e., solutions on the convex hull of the Pareto-optimal front.

References

J. D. Knowles and D. W. Corne, "A comparison of encodings and algorithms for multiobjective minimum spanning tree problems," in Proceedings of the 2001 Congress on Evolutionary Computation (IEEE Cat. No.01TH8546), vol. 1, 2001, pp. 544–551 vol. 1.

See Also

Other mcMST algorithms: `mcMSTEmoaBG`, `mcMSTEmoaZhou`

Examples

```
g = genRandomMCGP(30)
res = mcMSTPrim(g, n.lambdas = 50)
print(res$pareto.front)
```

`mutEdgeExchange` *One-edge-exchange mutator for edge list representation of spanning trees.*

Description

Each edge is replaced with another feasible edge with probability p. By default p = 1/m where m is the number of edges, i.e., in expectation one edge is replaced. The operators maintains the spanning tree property, i.e., the resulting edge list is indeed the edge list of a spanning tree.

Usage

```
mutEdgeExchange(ind, p = 1/ncol(ind))
```

Arguments

`ind`	`[matrix(2, m)]` Matrix of edges (each column is one edge).
`p`	`[numeric(1)]` Probability of edge exchange. Default is `1 / ncol(ind)`.

Value

`matrix(2, m)` Mutated edge list.

See Also

Evolutionary multi-objective algorithm `mcMSTEmoaBG`

Other mcMST EMOA mutators: `mutSubgraphMST`, `mutUniformPruefer`

`mutSubgraphMST` *Subgraph-mutator for edge list representation.*

Description

`mutSubgraphMST` selects a random edge e = (u, v) and traverses the tree starting form u and v respectively until a connected subtree of at most `sigma` edges is selected. Then the subtree is replaced with the optimal spanning subtree regarding one of the objectives with equal probability.

Usage

```
mutSubgraphMST(ind, sigma = floor(ncol(ind)/2), instance = NULL)
```

Arguments

`ind`	`[matrix(2, m)]` Matrix of edges (each column is one edge).
`sigma`	`[integer()]` Upper bound for the size of the selected subtree.
`instance`	`[mcGP]` Multi-objective graph problem.

Value

`matrix(2, m)` Mutated edge list.

See Also

Evolutionary multi-objective algorithm `mcMSTEmoaBG`

Other mcMST EMOA mutators: `mutEdgeExchange`, `mutUniformPruefer`

`mutUniformPruefer` *Uniform mutation for Pruefer code representation.*

Description

`mutUniformPruefer` replaces each component of a Pruefer code of length n - 2 with probability `p` with a random node number between 1 and n.

Usage

```
mutUniformPruefer(ind, p = 1/length(ind))
```

Arguments

`ind`	`[integer]` Pruefer code.
`p`	`[numeric(1)]` Probability of mutation of each component of `ind`. Default is `1 / length(ind)`.

Value

`integer` Mutated Pruefer code.

See Also

Evolutionary multi-objective algorithm `mcMSTEmoaZhou`

Other mcMST EMOA mutators: `mutEdgeExchange`, `mutSubgraphMST`

---

permutationToCharVec *Convert permutation to characteristic vector.*

Description

Convert permutation to characteristic vector.

Usage

```
permutationToCharVec(perm, n)
```

Arguments

perm	[integer] Permutation of nodes, e.g., solution of a TSP.
n	[integer] Number of nodes of the problem.

Value

integer Characteristic vector cv with cv[i] = 1 if the i-th edge is in the tree.

See Also

Other transformation functions: edgeListToCharVec, permutationToEdgelist, prueferToCharVec, prueferToEdgeList

Examples

```
# first generate a random permutation, e.g., representing
# a roundtrip tour in a graph
perm = sample(1:10)
print(perm)
# now convert into an edge list
permutationToCharVec(perm, n = 10)
```

---

permutationToEdgelist *Convert permutation to edge list.*

Description

Convert permutation to edge list.

Usage

```
permutationToEdgelist(perm)
```

Arguments

perm	[integer] Permutation of nodes, e.g., solution of a TSP.

Value

matrix(2, length(perm)) Edge list.

See Also

Other transformation functions: edgeListToCharVec, permutationToCharVec, prueferToCharVec, prueferToEdgeList

Examples

```
# first generate a random permutation, e.g., representing
# a roundtrip tour in a graph
perm = sample(1:10)
print(perm)
# now convert into an edge list
permutationToEdgelist(perm)
```

---

plot.mcGP — *Visualize bi-objective graph.*

Description

Only applicable for bi-objective problems of class mcGP. plot.mcGP generates a scatterplot of edge weights. If the nodes do have coordinates, additionally a scatterplot of the nodes in the euclidean plane is generated.

Usage

```
## S3 method for class 'mcGP'
plot(x, y = NULL, show.cluster.centers = TRUE, ...)
```

Arguments

x	[mcGP] Multi-objective graph problem.
y	Not used at the moment.
show.cluster.centers	[logical(1)] Display cluster centers? Default is TRUE. This option is ignored silently if the instance is not clustered.
...	[any] Not used at the moment.

Value

list A list of ggplot objects with components pl.weights (scatterplot of edge weights) and eventually pl.coords (scatterplot of nodes). The latter is NULL, if graph has no associated coordinates.

---

prueferToCharVec *Convert Pruefer code to characteristic vector.*

Description

Convert Pruefer code to characteristic vector.

Usage

```
prueferToCharVec(pcode)
```

Arguments

pcode [integer] Pruefer code encoding a minimum spanning tree.

Value

integer Characteristic vector cv with cv[i] = 1 if the i-th edge is in the tree.

See Also

Other transformation functions: edgeListToCharVec, permutationToCharVec, permutationToEdgelist, prueferToEdgeList

Examples

```
# here we generate a random Pruefer-code representing
# a random spanning tree of a graph with n = 10 nodes
pcode = sample(1:10, 8, replace = TRUE)
print(pcode)
print(prueferToCharVec(pcode))
```

prueferToEdgeList   *Convert Pruefer code to edge list.*

Description

Convert Pruefer code to edge list.

Usage

```
prueferToEdgeList(pcode)
```

Arguments

pcode	[integer] Pruefer code encoding a minimum spanning tree.

Value

`matrix(2, length(pcode) + 1)` Edge list.

See Also

Other transformation functions: edgeListToCharVec, permutationToCharVec, permutationToEdgelist, prueferToCharVec

Examples

```
# here we generate a random Pruefer-code representing
# a random spanning tree of a graph with n = 10 nodes
pcode = sample(1:10, 8, replace = TRUE)
print(pcode)
edgelist = prueferToEdgeList(pcode)
print(edgelist)
```

Index