Package 'mcMST'

March 13, 2023

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.

Version 1.1.1

Encoding UTF-8

Date 2023-03-13

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), grapherator

Imports checkmate (>= 1.1), gtools, ggplot2 (>= 1.0.0), vegan, qgraph, viridis, igraph

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

ByteCompile yes

RoxygenNote 7.2.3

VignetteBuilder knitr

NeedsCompilation no

Author Jakob Bossek [aut, cre]

Repository CRAN

Date/Publication 2023-03-13 19:00:02 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 set/front of multi-criteria minimum spanning tree (mcMST) problems.

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)

`charVecToEdgelist` *Convert characteristic vector to edge list.*

Description

Convert characteristic vector to edge list.

Usage

```
charVecToEdgelist(charvec)
```

Arguments

`charvec`	[`integer`] Characteristic vector.

Value

`matrix` Edge list.

See Also

Other transformation functions: `edgeListToCharVec()`, `nodelistToEdgelist()`, `permutationToCharVec()`, `permutationToEdgelist()`, `prueferToCharVec()`, `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)#'
edgelist = charVecToEdgelist(prueferToCharVec(pcode))
```

computeSimilarityMatrix *Compute similarity matrix.*

Description

Given a list of objects and a function which computes a similarity measure between two objects of the list, `computeSimilarity` returns a similarity matrix.

Usage

```
computeSimilarityMatrix(set, sim.fun, ...)
```

Arguments

`set`	[`list`] List of objects.
`sim.fun`	[`function(x, y, ...)`] Function which expects two objects `x` and `y` as first and second arguments and returns a scalar value.
`...`	[any] Passed down to `sim.fun`.

Value

`matrix(n, n)` (n, n) matrix with n being the length of `set`.

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: charVecToEdgelist(), nodelistToEdgelist(), 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 two uniformly randomly distribted edge weights.*

Description

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

grapherator Graph.

Note

This is a simple wrapper around the much more flexible graph generation system in package **grapherator**.

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 = grapherator::plot(g)

## End(Not run)
```

genRandomSpanningTree *Generate a random spanning tree.*

Description

Generate a random spanning tree of a graph given the number of nodes of the problem instance.

Usage

```
genRandomSpanningTree(n, type = "pruefer")
```

Arguments

n	[integer] Number of nodes of the problem.
type	[character(1)] String representing the desired format of the generated spanning tree. Possible values are "pruefer" (Pruefer-code), "edgelist" and "charvec" (characteristic vector). Default is "pruefer".

Value

integer | matrix(2, n) Return type depends on type.

Examples

```
genRandomSpanningTree(10)
genRandomSpanningTree(10, type = "edgelist")
```

genRandomSpanningTrees

Generate a set of random spanning trees.

Description

Generate a set of random spanning trees of a graph given the number of nodes of the problem instance.

Usage

```
genRandomSpanningTrees(m, n, type = "pruefer", simplify = TRUE)
```

Arguments

m	[integer(1)] Number of random spanning trees to be generated.
n	[integer] Number of nodes of the problem.
type	[character(1)] String representing the desired format of the generated spanning tree. Possible values are "pruefer" (Pruefer-code), "edgelist" and "charvec" (characteristic vector). Default is "pruefer".
simplify	[logical(1)] Should the result be simplified to a matrix if appropriate? Only relevant if type is either "pruefer" or "charvec". Default is TRUE.

Value

list | matrix Result type depends on simplify and type.

Examples

```
genRandomSpanningTrees(3, 10)
genRandomSpanningTrees(3, 10, simplify = FALSE)

genRandomSpanningTrees(3, 10, type = "edgelist")
```

---

getCommonSubtrees *Get common subtrees of two trees.*

Description

Given two spanning trees, the function returns the subtrees of the intersection of these.

Usage

```
getCommonSubtrees(x, y, n = NULL)
```

Arguments

x	[matrix] Edge list of first tree.
y	[matrix] Edge list of second tree.
n	[integer(1) | NULL] Number of nodes. Default to ncol(x) + 1.

Value

list List of matrizes. Each matrix contains the edges of one connected subtree.

Examples

```
# assume we have a graph with n = 10 nodes
n.nodes = 10
# we define two trees (matrices with colwise edges)
stree1 = matrix(c(1, 2, 1, 3, 2, 4, 5, 6, 6, 7), byrow = FALSE, nrow = 2)
stree2 = matrix(c(1, 3, 1, 2, 2, 4, 5, 8, 6, 7), byrow = FALSE, nrow = 2)
# ... and compute all common subtrees
subtrees = getCommonSubtrees(stree1, stree2, n = 10)
```

---

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, simplify = TRUE)
```

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.
`simplify`	[`logical(1)`] Should pareto set be simplified to matrix? This will only be done if all elements are of the same length. Otherwise the parameter will be ignored. Default is `TRUE`.

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)
```

---

getExtremeSolutions *Compute extreme spanning trees of bi-criteria graph problem.*

Description

Internally mcMSTPrim is called with weights set accordingly.

Usage

```
getExtremeSolutions(graph)
```

Arguments

graph	[grapherator] Graph.

Value

matrix(2, 2) The i-th column contains the objective vector of the extreme i-th extreme solution

getNumberOfSpanningTrees *Compute number of spanning trees of a graph*

Description

Makes use of Kirchhoff's matrix tree theorem to compute the number of spanning trees of a given graph in polynomial time.

Usage

```
getNumberOfSpanningTrees(graph)
```

Arguments

graph	[grapherator] Graph.

Value

integer(1)

Examples

```
# generate complete graph
g = genRandomMCGP(10)

# this is equal to 10^8 (Cayley's theorem)
getNumberOfSpanningTrees(g)
```

getRandomSpanningTree *Generate random spanning tree.*

Description

Given a grapherator object this function returns a random spanning tree. The tree generation process is a simple heuristic: A random weight from a $U(0, 1)$-distribution is assigned to each edge of the graph. Next, a spanning tree is computed by spantree.

Usage

```
getRandomSpanningTree(graph)
```

Arguments

graph	[grapherator] Graph.

Value

`matrix` Edge list of spanning tree edges.

Note

Most likely this heuristic does not produce each spanning tree with equal probability.

Examples

```
g = genRandomMCGP(10L)
stree = getRandomSpanningTree(g)
```

---

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, obj.types = NULL)
```

Arguments

graph	[grapherator] Graph.
edgelist	[matrix(2, k)] Matrix of edges (each column is one edge).
obj.types	[character] How to aggregate edge weights? Possible values are "sum" for sum objective and "bottleneck" for bottleneck/min-max objectives. Default is "sum" for each objective.

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))
getWeight(g, prueferToEdgeList(pcode), obj.types = "bottleneck")
```

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	[grapherator] Multi-objective graph.
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.

... [any]
Further parameters passed to mutator.

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	[grapherator] Multi-objective graph.
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 Prufer 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 $\lambda_i w_1 + (1 - \lambda_i) w_2$ for different weights $\lambda_i \in [0, 1]$.

Usage

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

Arguments

`instance`	[`grapherator`] Graph.
`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), instance = NULL)
```

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)`.
`instance`	[`grapherator`] Multi-objective graph.

Value

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

See Also

Evolutionary multi-objective algorithm `mcMSTEmoaBG`

Other mcMST EMOA mutators: `mutKEdgeExchange()`, `mutSubforestMST()`, `mutSubgraphMST()`, `mutUniformPruefer()`

mutKEdgeExchange *k-edge-exchange mutator for edge list representation of spanning trees.*

Description

Let m be the number of spanning tree edges. Then, the operator selects $1 \leq k \leq m$ edges randomly and replaces each of the k edges with another feasible edge.

Usage

```
mutKEdgeExchange(ind, k = 1L, instance = NULL)
```

Arguments

ind	[matrix(2, m)] Matrix of edges (each column is one edge).
k	[integer(1)] Number of edges to swap.
instance	[grapherator] Multi-objective graph.

See Also

Evolutionary multi-objective algorithm mcMSTEmoaBG

Other mcMST EMOA mutators: mutEdgeExchange(), mutSubforestMST(), mutSubgraphMST(), mutUniformPruefer()

mutSubforestMST *Forest-mutator for edge list representation.*

Description

mutForestMST drops k edges randomly. In consequence the tree is decomposed into k+1 subtrees (forest). Now the operator reconnects the subtrees by constructing a minimum spanning tree between the components.

Usage

```
mutSubforestMST(ind, sigma = ncol(ind), scalarize = FALSE, instance = NULL)
```

Arguments

ind	[matrix(2, m)] Matrix of edges (each column is one edge).
sigma	[integer()] Upper bound for number of dropped edges.
scalarize	[logical(1)] Should a scalarized version of the the subproblem be solved? If TRUE, a random weight $\lambda \in [0, 1]$ is sampled from a U[0, 1]-distribution. Next, a weighted sum scalarization $\lambda \cdot c_1 + (1 - \lambda) \cdot c_2$ of the subproblem is solved. Default is FALSE, i.e., the single-objective subproblem is solved. The objective to optimize for is sampled with equal probability.
instance	[grapherator] Multi-objective graph.

Value

matrix(2, m) Mutated edge list.

See Also

Evolutionary multi-objective algorithm mcMSTEmoaBG

Other mcMST EMOA mutators: mutEdgeExchange(), mutKEdgeExchange(), 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),
  scalarize = FALSE,
  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.
scalarize	[logical(1)] Should a scalarized version of the the subproblem be solved? If TRUE, a random weight $\lambda \in [0, 1]$ is sampled from a U[0, 1]-distribution. Next, a weighted sum scalarization $\lambda \cdot c_1 + (1 - \lambda) \cdot c_2$ of the subproblem is solved. Default is FALSE, i.e., the single-objective subproblem is solved. The objective to optimize for is sampled with equal probability.
instance	[grapherator] Multi-objective graph.

Value

matrix(2, m) Mutated edge list.

See Also

Evolutionary multi-objective algorithm mcMSTEmoaBG

Other mcMST EMOA mutators: mutEdgeExchange(), mutKEdgeExchange(), mutSubforestMST(), 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()`, `mutKEdgeExchange()`, `mutSubforestMST()`, `mutSubgraphMST()`

`nodelistToEdgelist` *Convert sequence of nodes to edge list.*

Description

Convert sequence of nodes to edge list.

Usage

```
nodelistToEdgelist(nodelist)
```

Arguments

`nodelist`	`[integer]` Sequence of nodes.

Value

`matrix` Edge list.

See Also

Other transformation functions: `charVecToEdgelist()`, `edgeListToCharVec()`, `permutationToCharVec()`, `permutationToEdgelist()`, `prueferToCharVec()`, `prueferToEdgeList()`

Examples

```
# first generate a random permutation, e.g., representing
# a roundtrip tour in a graph
nodelist = sample(1:8)
# now convert into an edge list
nodelistToEdgelist(nodelist)
```

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: charVecToEdgelist(), edgeListToCharVec(), nodelistToEdgelist(), 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: charVecToEdgelist(), edgeListToCharVec(), nodelistToEdgelist(), 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)
```

---

plotEdgeFrequency *Visualization of edge frequency among solution set.*

Description

Given a list of graphs and a list of solutions (encoded as edge lists) for each graph the function generates each one plot. This is a 2D-scatterplot of edge weights of the graph. Size and colour of each point indicate the number of solutions the edge is part of.

Usage

```
plotEdgeFrequency(graphs, approx.sets, facet.args = list(), names = NULL)
```

Arguments

graphs	[list(grapherator)] List of grapherator graphs.
approx.sets	[list(list(matrix))] List of approximations sets.
facet.args	[list] Further arguments passed down to facet_wrap. Only relevant if length(graphs) > 1.
names	[character] Optional names of the graph instances. Used for facetting. Default is "Problem_i" with i ranging from 1 to length(graphs).

Value

ggplot

See Also

Other result visualization: plotEdges()

Examples

```
g = genRandomMCGP(50L)
res = mcMSTEmoaBG(mu = 10L, max.iter = 50, instance = g, scalarize = TRUE)
## Not run:
plotEdgeFrequency(list(g), list(res$pareto.set))

## End(Not run)
```

---

plotEdges *Visualize edges common to several solutions.*

Description

Given a list of characteristic vectors (graphs) the function plots an embedding of the nodes in the Euclidean plane and depicts an edge if and only if it is contained in at least one of the graphs. The edge thickness indicates the number of graphs the edge is part of.

Usage

```
plotEdges(x, n = NULL, normalize = TRUE, ...)
```

Arguments

x	[list] List of characteristic vectors.
n	[integer(1)] Number of nodes of the problem instance. Default is sqrt(length(cv)) where cv is the first component of x.
normalize	[logical(1)] Shall edge frequencies be plotted? Default is code TRUE.
...	[any] Further arguments passed down to qgraph.

Value

Nothing

See Also

Other result visualization: plotEdgeFrequency()

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: charVecToEdgelist(), edgeListToCharVec(), nodelistToEdgelist(), 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: `charVecToEdgelist()`, `edgeListToCharVec()`, `nodelistToEdgelist()`, `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)
```

---

sampleWeights *Sample weights*

Description

Sample random weights $\lambda_1, ...\lambda_n, \sum_{i=1}^{n} \lambda_i = 1$ for weighted-sum scalarization.

Usage

```
sampleWeights(n)
```

Arguments

`n`	`[integer(1)]` Number of weights to sample.

Value

`numeric` Weight vector.

Examples

```
sampleWeights(2)

weights = replicate(10, sampleWeights(3L))
colSums(weights)
```

scalarizeWeights *Scalarize weight matrizes.*

Description

Given a list of weight matrizes `weight.mats` and a vector of numeric weights, the function returns a single weight matrix. Each component of the resulting matrix is the weighted sum of the corresponding components of the weight matrizes passed.

Usage

```
scalarizeWeights(weight.mats, lambdas)
```

Arguments

`weight.mats`	[`list`] List of weight matrizes.
`lambdas`	[`numeric`] Vector of weights.

Value

`matrix`

similarity_metrics *Metrics for spanning tree comparisson.*

Description

Functions which expect two (spanning) trees and return a measure of similiarity between those. Function `getNumberOfCommonEdges` returns the (normalized) number of shared edges and function `getSizeOfLargestCommonSubtree` returns the (normalized) size of the largest connected subtree which is located in both trees.

Usage

```
getNumberOfCommonEdges(x, y, n = NULL, normalize = TRUE)

getSizeOfLargestCommonSubtree(x, y, n = NULL, normalize = TRUE)
```

Arguments

x	[matrix(2, n)] First spanning tree represented as a list of edges.
y	[matrix(2, n)] Second spanning tree represented as a list of edges.
n	[integer(1) | NULL] Number of nodes of the graph. Defaults to length(x).
normalize	[logical(1)] Should measure be normalized to $[0, 1]$ by devision through the number of edges? Default is TRUE.

Value

numeric(1) Measure

Examples

```
# Here we generate two random spanning trees of a complete
# graph with 10 nodes
set.seed(1)
st1 = prueferToEdgeList(sample(1:10, size = 8, replace = TRUE))
st2 = prueferToEdgeList(sample(1:10, size = 8, replace = TRUE))
# Now check the number of common edges
NCE = getNumberOfCommonEdges(st1, st2)
# And the size of the largest common subtree
SLS = getSizeOfLargestCommonSubtree(st1, st2)
```

Index