Package ‘tspmeta’

July 8, 2015

Title  Instance Feature Calculation and Evolutionary Instance Generation for the Traveling Salesman Problem

Description  Instance feature calculation and evolutionary instance generation for the traveling salesman problem. Also contains code to "morph" two TSP instances into each other. And the possibility to conveniently run a couple of solvers on TSP instances.

Author  Bernd Bischl <bernd_bischl@gmx.net>, Jakob Bossek <jakob.bossek@tu-dortmund.de>, Olaf Mersmann <olafm@p-value.net>

Maintainer  Bernd Bischl <bernd_bischl@gmx.net>

URL  https://github.com/berndbischl/tspmeta

BugReports  https://github.com/berndbischl/tspmeta/issues

License  BSD_3_clause + file LICENSE

Depends  ggplot2, TSP, MASS

Imports  BBmisc, checkmate (>= 1.5), fpc, vegan, stringr, splancs

Suggests  testthat

LazyData  yes

ByteCompile  yes

Version  1.2

NeedsCompilation  yes

Repository  CRAN

Date/Publication  2015-07-08 11:38:24

R topics documented:

  as_TSP ................................................................. 2
  autoplot.tsp_instance ........................................... 3
  center_of_mass .................................................... 3
  fast_two_opt ....................................................... 4
  features ............................................................ 4
as_TSP

Convert to TSP instance object of package TSP.

Description

Convert to TSP instance object of package TSP.

Usage

as_TSP(x)

Arguments

x [tsp_instance]
TSP instance.
**Value**

TSP.

---

**Description**

Plot TSP instance.

**Usage**

```r
## S3 method for class 'tsp_instance'
autoplot(object, opt_tour, ...)
```

**Arguments**

- **object** [tsp_instance]
  TSP instance.
- **opt_tour** [TOUR]
  TOUR object from package TSP, containing order of cities, tour length and method name that generated this solution.
- **...** [any]
  Not used.

**Value**

ggplot.

---

**center_of_mass**

*Return the center of all cities of a TSP instance.*

---

**Description**

Return the center of all cities of a TSP instance.

**Usage**

```r
center_of_mass(instance)
```

**Arguments**

- **instance** [tsp_instance]
  TSP instance.

**Value**

numeric(2) Center of all cities of the TSP instance.
### fast_two_opt

**Runs 2-Opt local search on TSP instance.**

**Description**

Runs 2-Opt local search on TSP instance.

**Usage**

```r
fast_two_opt(xL initial_tour)
```

**Arguments**

- **x** [tsp_instance]
  - TSP instance.
- **initial_tour** [numeric]
  - Initial tour.

**Value**

TOUR TOUR object from package TSP, containing order of cities, tour length and method name that generated this solution.

### features

**Calculates list of all TSP features for an instance.**

**Description**

Calculates list of all TSP features for an instance.

**Usage**

```r
features(x, rescale = TRUE)
```

**Arguments**

- **x** [tsp_instance]
  - TSP instance.
- **rescale** [logical(1)]
  - Rescale x to \([0, 1]^2\) before calculation of features? Default is TRUE.

**Value**

list.
feature_angle

See Also

feature_angle, feature_centroid, feature_cluster, feature_bounding_box, feature_chull,
feature_distance, feature_modes, feature_mst, feature_nnds

Examples

```r
x = random_instance(10)
print(features(x))
```

---

**feature_angle**  
*Angle features.*

**Description**

Statistics of the distribution of the angle between a node and its 2 next neighbors.

**Usage**

```r
feature_angle(x)
```

**Arguments**

- `x`  
  *tsp_instance*  
  TSP instance.

**Value**

list.

---

**feature_bounding_box**  
*Bounding box features.*

**Description**

Determines the ratio of cities which lie within a certain distance to the bounding box.

**Usage**

```r
feature_bounding_box(x, distance_fraction = 0.1)
```

**Arguments**

- `x`  
  *tsp_instance*  
  TSP instance.

- `distance_fraction`  
  *numeric(1)*  
  Distance ratio to bounding box.
<table>
<thead>
<tr>
<th>Feature</th>
<th>Description</th>
<th>Usage</th>
<th>Arguments</th>
</tr>
</thead>
<tbody>
<tr>
<td>feature_chull</td>
<td>Convex hull features.</td>
<td>feature_chull(x)</td>
<td>x: [tsp_instance] TSP instance.</td>
</tr>
<tr>
<td>feature_centroid</td>
<td>Centroid features.</td>
<td>feature_centroid(x)</td>
<td>x: [tsp_instance] TSP instance.</td>
</tr>
</tbody>
</table>

**Description**

- **feature_chull**: Determines the area of the convex hull and the ratio of the cities which lie on the convex hull in the euclidean space.
- **feature_centroid**: Includes the coordinates of the mean coordinates of the the point cloud and the statistics of the distances of all cities from it.
**feature_cluster**  
*Cluster features.*

**Description**
Determines the number of clusters and the mean distances from all cities in a cluster to its centroid.

**Usage**
```
feature_cluster(x, epsilon)
```

**Arguments**
- `x`: [tsp_instance]  
  TSP instance.
- `epsilon`: [numeric(1)]  
  Probability in [0,1]. Used to compute the reachability distance for the underlying dbscan clustering algorithm.

**Value**
list.

**feature_distance**  
*Distance features.*

**Description**
Computes different statistics describing the distribution of pairwise distances between cities.

**Usage**
```
feature_distance(x)
```

**Arguments**
- `x`: [tsp_instance]  
  TSP instance.

**Value**
list List of statistics describing the distribution of distances.
**feature_modes**  
*Modes of edge cost distribution feature.*

**Description**
Includes the number of modes of the edge cost distribution.

**Usage**
```
feature_modes(x)
```

**Arguments**
- `x`  
  [tsp_instance]
  TSP instance.

**Value**
- `list`  
  List containing (estimated) number of modes.

---

**feature_mst**  
*MST features.*

**Description**
Construct minimum spanning tree, then calculate the statistics of a) the distances in the MST, b) the depths of all nodes in the MST.

**Usage**
```
feature_mst(x)
```

**Arguments**
- `x`  
  [tsp_instance]
  TSP instance.

**Value**
- `list`  
  List
feature_nnds  Nearest neighbor features.

Description
Statistics describing the distribution of distances of each city to its nearest neighbor.

Usage
feature_nnds(x)

Arguments
x  [tsp_instance]
TSP instance.

Value
list.

get_solvers  Returns integrated solver names.

Description
Returns integrated solver names.

Usage
get_solvers()
### Greedy Point Matching

**Description**

Pairs of cities are matched in a greedy fashion for morphing, first the closest pair w.r.t. euclidean distance, then the closest pair of the remaining cities, and so on.

**Usage**

```r
greedy_point_matching(x, y)
```

**Arguments**

- `x` [tsp_instance] First TSP instance.

**Value**

Matrix: Numeric matrix of point indices with shortest distance.

### Instance Dimension

**Description**

Get instance dimensionality (space where coords live).

**Usage**

```r
instance_dim(x)
```

**Arguments**

- `x` [tsp_instance] TSP instance.

**Value**

Integer(1).
**morph_instances**

*Morphing (convex-combination) of two instances with parameter alpha.*

**Description**

Pairs of cities are matched in a greedy fashion, see `greedy_point_matching`.

**Usage**

`morph_instances(x, y, alpha)`

**Arguments**

- **x**
  - `[tsp_instance]`
- **y**
  - `[tsp_instance]`
- **alpha**
  - `[numeric(1)]`
  
  Coefficient alpha for convex combination.

**Value**

`tsp_instance` Morphed TSP instance.

**Examples**

```r
x = random_instance(10)
y = random_instance(10)
z = morph_instances(x, y, 0.5)
autoplot(x)
autoplot(y)
autoplot(z)
```

---

**normalization_angle**

*Calculate rotation angle such that the main axis through the cities is aligned with the X axis.*

**Description**

Calculate rotation angle such that the main axis through the cities is aligned with the X axis.

**Usage**

`normalization_angle(instance)`
Arguments

instance [tsp_instance]
TSP instance.

Value

numeric(1)

normalize_rotation Normalize an instance w.r.t. its rotation.

Description

Normalization is performed by aligning the main axis of the cities with the X axis.

Usage

normalize_rotation(instance)

Arguments

instance [tsp_instance]

Value

A rotated tsp_instance.

See Also

normalization_angle

number_of_cities Get number of cities in tsp instance.

Description

Get number of cities in tsp instance.

Usage

number_of_cities(x)

Arguments

x [tsp_instance]
TSP instance.
**Value**
numeric(1).

---

**numvec_feature_statistics**

*Computes statistics from a vector of values.*

**Description**

E.g. computes features from distribution of distances. Computed statistics: min, median, mean, max, sd, span, coeff_of_var.

**Usage**

```r
numvec_feature_statistics(x, name, na.rm = TRUE)
```

**Arguments**

- **x** [numeric]
  Numeric vector.
- **name** [numeric]
  Prefix name for elements in result list.
- **na.rm** [logical(1)]
  Should NAs in x be removed? Default is TRUE.

**Value**

list Elements are named `<name_statistic>`.

---

**print.tsp_instance**

*Print TSP instance*

**Description**

Print TSP instance

**Usage**

```r
## S3 method for class 'tsp_instance'
print(x, ...)
```

**Arguments**

- **x** [tsp_instance]
  TSP instance.
- **...** [any]
  Not used.
random_instance       Generates a random TSP instance by scattering random points in a hypercube.

Description
Generates a random TSP instance by scattering random points in a hypercube.

Usage
random_instance(size, d = 2, lower = 0, upper = 1)

Arguments
- size [integer(1)]
  Number of cities.
- d [integer(1)]
  Space dimensionality, e.g. 2D. Default is 2D.
- lower [numeric(1)]
  Lower box constraint for hypercube. Default is 0.
- upper [numeric(1)]
  Upper box constraint for hypercube. Default is 1.

Value
tsp_instance.

read_tsplib_instance       Read in a TSPLIB style Traveling Salesman Problem from a file.

Description
The current state of the parser does not understand all variants of the TSPLIB format. Much effort has been spent making the parser as robust as possible. It will stop as soon as it sees input it cannot handle.

Usage
read_tsplib_instance(path)

Arguments
- path [character(1)]
  Character string containing path to file in TSPLIB format.
`read_tsplib_instances`  

**Description**  
Read in multiple TSPLIB style Traveling Salesman Problems from a directory.

**Usage**  
```r  
read_tsplib_instances(path, pattern = "*.tsp", max_size = 1000,  
use_names = TRUE, on_no_coords = "stop")  
```

**Arguments**  
- **path**  
  Character string containing path to file in TSPLIB format.
- **pattern**  
  Pattern of files under `path` that are considered as instances.
- **max_size**  
  Upper bound for instance size (i.e. number of cities). Only applicable, if instance size is contained in file name. Default value is 1000.
- **use_names**  
  Use base names of files as names of instances in returned list.
- **on_no_coords**  
  How to handle instances which do not have any coordinates. Possible values are, “stop” and “warn” which either stop or raise a warning respectively.

**Value**  
A list of `tsp_instance` objects.
read_tsplib_tour  
*Read in a TSPLIB style Traveling Salesman Problem tour from a file*

**Description**
Read in a TSPLIB style Traveling Salesman Problem tour from a file

**Usage**

```r
read_tsplib_tour(path)
```

**Arguments**

- `path` [character(1)]
  Filename of file containing a TSP tour.

**Value**

**TOUR** TOUR object from package TSP, containing order of cities, tour length and method name that generated this solution.

---

remove_zero_distances  
*Remove any duplicate cities in a tsp instance.*

**Description**
Remove any duplicate cities in a tsp instance.

**Usage**

```r
remove_zero_distances(instance)
```

**Arguments**

- `instance` [tsp_instance]
  TSP instance object.

**Value**

New TSP instance in which all duplicate cities have been removed.
rescale_instance

Rescale coords of TSP instance to $[0, 1]^2$.

Description
Rescale coords of TSP instance to $[0, 1]^2$.

Usage

rescale_instance(x)
rescale_coords(coords)

Arguments

x [tsp_instance] TSP instance.
coords [matrix] Numeric matrix of city coordinates, rows denote cities.

Value
matrix for rescale_coords and tsp_instance for rescale_instance. Numeric matrix of scaled city coordinates.

rotate_coordinates

Rotate a matrix of 2D coordinates

Description
Rotate a matrix of 2D coordinates

Usage

rotate_coordinates(coords, angle, center)

Arguments

coords [matrix] Numeric matrix of 2D coordinates to rotate
angle [numeric(1)] Angle by which to rotate the coordinates. In radians.
center [matrix] Center around which to rotate the coordinates.
Value

A matrix of rotated coordinates.

---

**rotate_instance**

*Rotate the cities of a TSP instance around a point.*

---

**Description**

Rotate the cities of a TSP instance around a point.

**Usage**

```
rotate_instance(instance, angle, center)
```

**Arguments**

- **instance**
  - [tsp_instance]
  - TSP instance.
- **angle**
  - [numeric(1)]
  - Angle by which to rotate the coordinates. In radians.
- **center**
  - [numeric]
  - Point around which to rotate the cities. If missing, defaults to the center of mass of the cities.

**Value**

- **tsp_instance** New TSP instance.

---

**run_solver**

*Runs a solver on a TSP instance.*

---

**Description**

Currently the following solvers are supported: nearest_insertion: See `solve_TSP`. farthest_insertion: See `solve_TSP`. cheapest_insertion: See `solve_TSP`. arbitrary_insertion: See `solve_TSP`. nn: See `solve_TSP`. repetitive_nn: See `solve_TSP`. concorde: See `solve_TSP`.

**Usage**

```
run_solver(x, method, ...)
```
Arguments

- **x**  
  [tsp_instance]  
  TSP instance.

- **method**  
  [character(1)]  
  Solver to use on TSP instance. To use concorde and/or linkern it is necessary to specify the path to the concorde/linkern executable with `concorde_path`.

- ...  
  [any]  
  Control parameters for solver.

Value

TOUR TOUR object from package TSP, containing order of cities, tour length and method name that generated this solution.

Examples

```r
x = random_instance(10)
tours = sapply(c("nn", "cheapest_insertion", "arbitrary_insertion"), function(solver) {
  list(solver = run_solver(x, method = solver))
})
## not run:
concorde_path(path = "/absolute/path/to/concorde/executable")
concorde_tour = run_solver(x, method = "concorde")
concorde_tour = run_solver(x, method = "linkern")
## End(not run)
```

tsp_generation_ea  
_TSP generating EA._

description

TSP generating EA.

usage

tsp_generation_ea(fitness_function, pop_size = 30L, inst_size = 50L, generations = 100L, time_limit = 30L, uniform_mutation_rate, normal_mutation_rate, normal_mutation_sd, cells_round = 100L, rnd = TRUE, ...)

Arguments

- **fitness_function**  
  [function(x, ...)]  
  Fitness function used to judge the fitness of a TSP instance. x is a numeric matrix with 2 columns, containing the coordinates of a TSP instance.
pop_size [integer(1)]
Number of TSP instances maintained in each population. Default is 30.

inst_size [integer(1)]
Number of cities of each TSP instance. Default is 50.

generations [integer(1)]
Number of generations. Default is 100L.

time_limit [integer(1)]
Time limit in seconds. Default is 30.

uniform_mutation_rate [numeric(1)]
Mutation probability in uniform mutation (in [0,1]).

normal_mutation_rate [numeric(1)]
Mutation probability in normal mutation (in [0,1])

normal_mutation_sd [numeric(1)]
Standard deviation of normal noise in normal mutation

cells_round [numeric(1)]
Grid resolution for rounding. Default is 100.

rnd [logical(1)]
Round the coordinates before normal mutation. Default is TRUE.

Value

list List containing best individual from the last population, its fitness value, the generational fitness and the last population. Default is 50.

tsp_instance Generates a TSP instance S3 object either from city coordinates.

Description
Generates a TSP instance S3 object either from city coordinates.

Usage

tsp_instance(coords, dists)

Arguments

coords [matrix]
Numeric matrix of city coordinates, rows denote cities.

dists [dist]
Optional distance matrix containing the inter-city distances. If not provided, the (euclidean) distances are computed from the coordinates.
**tsp\_instance**

**Value**

`tsp\_instance`.
Index

as_TSP, 2
autoplot.tsp_instance, 3
center_of_mass, 3
concorde_path, 19
dbscan, 7
fast_two_opt, 4
feature_angle, 5, 5
feature_bounding_box, 5, 5
feature_centroid, 5, 6
feature_chull, 5, 6
feature_cluster, 5, 7
feature_distance, 5, 7
feature_modes, 5, 8
feature_mst, 5, 8
feature_nnds, 5, 9
features, 4
get_solvers, 9
ggplot, 3
greedy_point_matching, 10, 11
instance_dim, 10
morph_instances, 11
normalization_angle, 11, 12
normalize_rotation, 12
number_of_cities, 12
numvec_feature_statistics, 13
print.tsp_instance, 13
random_instance, 14
read_tsplib_instance, 14
read_tsplib_instances, 15
read_tsplib_tour, 16
remove_zero_distances, 16
rescale_coords(rescale_instance), 17
rescale_instance, 17
rotate_coordinates, 17
rotate_instance, 18
run_solver, 18
solve_TSP, 18
TOUR, 3, 4, 16, 19
TSP, 3
tsp_generation_ea, 19
tsp_instance, 2–15, 17–19, 20, 21