Package 'TSP'

July 16, 2014

Type Package

Title Traveling Salesperson Problem (TSP)

Version 1.0-9

Date 2013-07-16

Description Basic infrastructure and some algorithms for the traveling
salesperson problem (also traveling salesman problem; TSP).
The package provides some simple algorithms and
an interface to Concorde, the currently fastest TSP solver. Concorde
itself is not included in the package and has to be obtained separately.

Classification/ACM G.1.6, G.2.1, G.4

URL <http://R-Forge.R-project.org/projects/tsp/>,<http://lyle.smu.edu/IDA/seriation/>

Depends R (>= 2.14.0)

Suggests maps, sp, maptools

License GPL-3

Copyright All code is Copyright (C) Michael Hahsler and Kurt Hornik.

Author Michael Hahsler [aut, cre, cph],Kurt Hornik [aut, cph]

Maintainer Michael Hahsler <mhahsler@lyle.smu.edu>

NeedsCompilation yes

Repository CRAN

Date/Publication 2014-07-16 15:56:16

1

R **topics documented:**

ATSP   *Class ATSP – Asymmetric traveling salesperson problem*

Description

Constructor to create an instance of the asymmetric traveling salesperson problem (ATSP) and some auxiliary methods.

Usage

```
## constructor
ATSP(x, labels = NULL)

## coercion
as.ATSP(object)

## methods
## S3 method for class 'ATSP'
n_of_cities(x)
## S3 method for class 'ATSP'
image(x, order, col = gray.colors(64), ...)
## S3 method for class 'ATSP'
labels(object, ...)
## S3 method for class 'ATSP'
print(x, ...)
```

Arguments

x, object	an object (a square matrix) to be converted into an ATSP or, for the methods, an object of class ATSP.
labels	optional city labels. If not given, labels are taken from x.
col	color scheme for image.

order	order of cities as an integer vector or an object of class TOUR.
...	further arguments are passed on.

Details

Objects of class ATSP are internally represented by a matrix (use as.matrix() to get just the matrix).

Value

ATSP() returns x as an object of class ATSP.

n_of_cities() returns the number of cities in x.

labels() returns a vector with the names of the cities in x.

See Also

[TOUR](#), [insert_dummy](#), [tour_length](#), [solve_TSP](#).

Examples

```
data <- matrix(runif(10^2), ncol = 10, dimnames = list(1:10, 1:10))

atsp <- ATSP(data)
atsp

## use some methods
n_of_cities(atsp)
labels(atsp)

## calculate a tour
tour <- solve_TSP(atsp)

tour_length(atsp)
tour_length(atsp, tour)

image(atsp, tour)
```

Concorde	*Using the Concorde TSP Solver*

Description

The Concorde TSP Solver package contains several solvers. Currently, interfaces to the Concorde solver (Applegate et al. 2001), one of the most advanced and fastest TSP solvers using branch-and-cut, and the Chained Lin-Kernighan (Applegate et al. 2003) implementation are provided in **TSP**.

The Concorde TSP Solver package is freely available for academic research and has to be obtained separately from the Concorde web site (see details).

Usage

```
## set path for executables
concorde_path(path)

## obtain a list of command line options for the solvers
concorde_help(exe = NULL)
linkern_help(exe = NULL)
```

Arguments

exe   an optional character string containing the whole path to the executable (includ-
                ing the executable's name).

                Other options are (1) to use concorde_path() to set the path to the directory
                containing the executables for concorde and linkern, or (2) to put the executables
                somewhere in the search path.

path   path to the directory where the executables are stored.

Details

The code of the Concorde TSP package is not included in this package and has to be obtained
separately from the Concorde web site (see references). Either download the precompiled ex-
ecutables and place them in a suitable directory (either in the search path, or you have to use
concorde_path()) and make them executable, or you can get the source code and compile it on
your own.

To get a list of all available command line options which can be used via the clo option for
solve_TSP use concorde_help() and linkern_help(). Several options ('-x', '-o', '-N', '-Q')
are not available via solve_TSP since they are used by the interface.

References

Concorde home page, <http://www.tsp.gatech.edu/concorde/>

Concorde download page, <http://www.tsp.gatech.edu/concorde/downloads/downloads.htm>

David Appletgate, Robert Bixby, Vasek Chvatal, William Cook (2001): TSP cuts which do not
conform to the template paradigm, Computational Combinatorial Optimization, M. Junger and D.
Naddef (editors), Springer.

David Applegate and William Cook and Andre Rohe (2003): Chained Lin-Kernighan for Large
Traveling Salesman Problems, *INFORMS Journal on Computing*, **15**, 82–92.

See Also

solve_TSP

---

cut_tour	*Cut a tour to form a path*

Description

Cuts a tour at a specified city to form a path.

Usage

```
cut_tour(x, cut, exclude_cut = TRUE)
```

Arguments

x	an object of class TOUR.
cut	the index or label of the city to cut the tour.
exclude_cut	exclude the city where we cut? If FALSE, the city at the cut is included in the path as the first city.

See Also

[TOUR](#).

Examples

```
data("USCA50")

tsp <- insert_dummy(USCA50, label = "cut")
tour <- solve_TSP(tsp)

## cut tour into path at the dummy city
path <- cut_tour(tour, "cut")

labels(path)
```

---

insert_dummy	*Insert dummy cities into a distance matrix*

Description

Inserts dummy cities into objects of class TSP or ATSP. A dummy city has the same, constant distance (0) to all other cities and is infinitely far from other dummy cities. A dummy city can be used to transform a shortest Hamiltonian path problem (i.e., finding an optimal linear order) into a shortest Hamiltonian cycle problem which can be solved by a TSP solvers (Garfinkel 1985).

Several dummy cities can be used together with a TSP solvers to perform rearrangement clustering (Climer and Zhang 2006).

Usage

```
insert_dummy(x, n = 1, const = 0, inf = Inf, label =  "dummy")
```

Arguments

x	an object of class TSP or ATSP.
n	number of dummy cities.
const	distance of the dummy cities to all other cities.
inf	distance between dummy cities.
label	labels for the dummy cities. If only one label is given, it is reused for all dummy cities.

Details

The dummy cities are inserted after the other cities in x.

A const of 0 is guaranteed to work if the TSP finds the optimal solution. For heuristics returning suboptimal solutions, a higher const (e.g., 2 * max{x}) might provide better results.

References

Sharlee Climer, Weixiong Zhang (2006): Rearrangement Clustering: Pitfalls, Remedies, and Applications, *Journal of Machine Learning Research* **7**(Jun), pp. 919–943.

R.S. Garfinkel (1985): Motivation and modelling (chapter 2). In: E. L. Lawler, J. K. Lenstra, A.H.G. Rinnooy Kan, D. B. Shmoys (eds.) The traveling salesman problem - A guided tour of combinatorial optimization, Wiley \& Sons.

See Also

TSP, ATSP

Examples

```
## make runs comparable
set.seed(4444)

data("iris")
tsp <- TSP(dist(iris[-5]))

## insert 2 dummy cities
tsp_dummy <- insert_dummy(tsp, n = 2, label = "boundary")

## get a solution for the TSP
tour <- solve_TSP(tsp_dummy)

## plot the distance matrix
image(tsp_dummy, tour)

## draw lines where the dummy cities are located
abline(h = which(labels(tour)=="boundary"), col = "red")
```

```
abline(v = which(labels(tour)=="boundary"), col = "red")

## print the results (NAs are the dummy cities)
iris[tour, "Species"]
```

---

reformulate_ATSP_as_TSP

Reformulate a ATSP as a symmetric TSP

Description

A ATSP can be formulated as a symmetric TSP by doubling the number of cities (Jonker and Volgenant 1983). The solution of the TSP also represents the solution of the original ATSP.

Usage

```
reformulate_ATSP_as_TSP(x, infeasible = Inf, cheap = -Inf)
```

Arguments

x	an ATSP.
infeasible	value for infeasible connections.
cheap	value for distance between a city and its corresponding dummy city.

Details

To reformulate the ATSP as a TSP, for each city a dummy city (e.g, for 'New York' a dummy city 'New York*') is added. Between each city and its corresponding dummy city a negative or very small distance with value cheap is used. This makes sure that each cities always occurs in the solution together with its dummy city. The original distances are used between the cities and the dummy cities, where each city is responsible for the distance going to the city and the dummy city is responsible for the distance coming from the city. The distances between all cities and the distances between all dummy cities are set to infeasible, a very large value which makes the infeasible.

Value

a TSP object.

References

Jonker, R. and Volgenant, T. (1983): Transforming asymmetric into symmetric traveling salesman problems, *Operations Research Letters, 2, 161–163.*

See Also

ATSP, TSP.

Examples

```
data("USCA50")

## set the distances towards Austin to zero which makes it a ATSP
austin <- which(labels(USCA50) == "Austin, TX")
atsp <- as.ATSP(USCA50)
atsp[, austin] <- 0

## reformulate as a TSP
tsp <- reformulate_ATSP_as_TSP(atsp)
labels(tsp)

## create tour (now you could use Concorde or LK)
tour_atsp <- solve_TSP(tsp, method="nn")
head(labels(tour_atsp), n = 10)

## filter out the dummy cities
tour <- TOUR(tour_atsp[tour_atsp <= n_of_cities(atsp)])
tour_length(atsp, tour)
```

---

solve_TSP   *TSP solver interface*

Description

Common interface to all TSP solvers in this package.

Usage

```
solve_TSP(x, method, control)
```

Arguments

x	the TSP given as an object of class TSP or ATSP.
method	method to solve the TSP (default: nearest insertion algorithm; see details).
control	a list of arguments passed on to the TSP solver selected by method.

Details

Currently the following methods are available:

"nearest_insertion", "farthest_insertion", "cheapest_insertion", "arbitrary_insertion"
Nearest, farthest, cheapest and arbitrary insertion algorithms for a symmetric and asymmetric
TSP (Rosenkrantz et al. 1977).

The distances between cities are stored in a distance matrix D with elements $d(i, j)$. All
insertion algorithms start with a tour consisting of an arbitrary city and choose in each step a

city k not yet on the tour. This city is inserted into the existing tour between two consecutive cities i and j, such that

$$d(i, k) + d(k, j) - d(i, j)$$

is minimized. The algorithms stops when all cities are on the tour.

The nearest insertion algorithm chooses city k in each step as the city which is *nearest* to a city on the tour.

For farthest insertion, the city k is chosen in each step as the city which is *farthest* to any city on the tour.

Cheapest insertion chooses the city k such that the cost of inserting the new city (i.e., the increase in the tour's length) is minimal.

Arbitrary insertion chooses the city k randomly from all cities not yet on the tour.

Nearest and cheapest insertion tries to build the tour using cities which fit well into the partial tour constructed so far. The idea behind behind farthest insertion is to link cities far away into the tour fist to establish an outline of the whole tour early.

Additional control options:

start index of the first city (default: random city).

"nn", "repetitive_nn" Nearest neighbor and repetitive nearest neighbor algorithms for symmetric and asymmetric TSPs (Rosenkrantz et al. 1977).

The algorithm starts with a tour containing a random city. Then the algorithm always adds to the last city on the tour the nearest not yet visited city. The algorithm stops when all cities are on the tour.

Repetitive nearest neighbor constructs a nearest neighbor tour for each city as the starting point and returns the shortest tour found.

Additional control options:

start index of the first city (default: random city).

"2-opt" Two edge exchange improvement procedure (Croes 1958).

This procedure systematically exchanges two edges in the graph represented by the distance matrix till no improvements are possible. Exchanging two edges is equal to reversing part of the tour. The resulting tour is called *2-optimal.*

By default, improvement starts with a random tour.

Additional control options:

tour an existing tour which should be improved. If no tour is given, a random tour is used.

rep number of times to try 2-opt with a different initial random tour (default: 1).

"concorde" Concorde algorithm (Applegate et al. 2001).

Concorde is an advanced exact TSP solver for *only symmetric* TSPs based on branch-and-cut. The program is not included in this package and has to be obtained and installed separately (see Concorde).

Additional control options:

exe a character string containing the path to the executable (see Concorde).

clo a character string containing command line options for Concorde, e.g., control = list(clo = "-B -v"). See concorde_help on how to obtain a complete list of available command line options.

precision an integer which controls the number of decimal places used for the internal representation of distances in Concorde. The values given in x are multiplied by $10^{precision}$ before being passed on to Concorde. Note that therefore the results produced by Concorde (especially lower and upper bounds) need to be divided by $10^{precision}$ (i.e., the decimal point has to be shifted precision placed to the left). Note also, that Concorde cannot handle Inf which is therefore replaced by 2 times the maximum value in x (ignoring the infinity entries). The interface to Concorde uses write_TSPLIB (see there for more information).

"linkern" Concorde's Chained Lin-Kernighan heuristic (Applegate et al. 2003).

The Lin-Kernighan (Lin and Kernighan 1973) heuristic uses variable k edge exchanges to improve an initial tour. The program is not included in this package and has to be obtained and installed separately (see Concorde).

Additional control options: see Concorde above.

Value

An object of class TOUR.

References

Concorde home page, <http://www.tsp.gatech.edu/concorde/>

David Appletgate, Robert Bixby, Vasek Chvatal, William Cook (2001): TSP cuts which do not conform to the template paradigm, Computational Combinatorial Optimization, M. Junger and D. Naddef (editors), Springer.

D. Applegate, W. Cook and A. Rohe (2003): Chained Lin-Kernighan for Large Traveling Salesman Problems. *INFORMS Journal on Computing, 15(1):82–92.*

G.A. Croes (1958): A method for solving traveling-salesman problems. *Operations Research, 6(6):791–812.*

S. Lin and B. Kernighan (1973): An effective heuristic algorithm for the traveling-salesman problem. *Operations Research, 21(2): 498–516.*

L.S. Pitsoulis and M.G.C. Resende (2001): Greedy randomized adaptive search procedures. In P.M. Pardalos and M.G.C. Resende, editors, Handbook of Applied Optimization, pp. 168–181.

D.J. Rosenkrantz, R. E. Stearns, and Philip M. Lewis II (1977): An analysis of several heuristics for the traveling salesman problem. *SIAM Journal on Computing, 6(3):563–581.*

See Also

TOUR, TSP, ATSP, write_TSPLIB, Concorde.

Examples

```
data("USCA50")
USCA50

## methods
methods <- c("nearest_insertion", "cheapest_insertion", "farthest_insertion",
    "arbitrary_insertion", "nn", "repetitive_nn", "2-opt")
```

```
## calculate tours
tours <- lapply(methods, FUN = function(m) solve_TSP(USCA50, method = m))
names(tours) <- methods

## use the external solver which has to be installed separately
## Not run:
tours$concorde  <- solve_TSP(USCA50, method = "concorde")
tours$linkern   <- solve_TSP(USCA50, method = "linkern")

## End(Not run)

## show first tour
tours[[1]]

## compare tour lengths
opt <- 14497 # optained by concorde
tour_lengths <- c(sapply(tours, FUN = attr, "tour_length"), optimal = opt)
dotchart(tour_lengths/opt*100-100, xlab = "percent excess over optimum")
```

TOUR   *Class TOUR – Solution to a traveling salesperson problem*

Description

Class to store the solution of a TSP. Objects of this class are returned by TSP solvers in this package. Essentially, an object of class TOUR is a permutation vector containing the order of cities to visit.

Usage

```
## conctructor
TOUR(x, method=NA, tsp=NULL)

## coercion
as.TOUR(object)

## methods
## S3 method for class 'TOUR'
print(x, ...)
```

Arguments

x	an integer permutation vector or, for the methods an object of class TOUR.
object	data (an integer vector) which can be coerced to TOUR..
method	character string; method used to create the tour.
tsp	TSP object the tour applies to. If available then the tour will include the tour length. Also the labes of the cities will be available in the tour (otherwise the labels of x are used).

...   further arguments are passed on.

Details

Since an object of class TOUR is an integer vector, it can be subsetted as an ordinary vector or coerced to an integer vector using as.integer(). It also contains the names of the objects as labels. Additionally, TOUR has the following attributes: "method", "tour_length".

For most functions, e.g., tour_length or image, the TSP/ATSP object used to find the tour is still needed, since the tour does not contain the distance information.

See Also

TSP, ATSP, tour_length, image.

Examples

```
TOUR(1:10)

## calculate a tour
data("USCA50")
tour <- solve_TSP(USCA50)
tour

## get permutation vector
as.integer(tour)

## show labels
labels(tour)
```

---

tour_length   *Calculate the length of a tour*

Description

Calculate the length of a tour given a TSP and an order.

Usage

```
tour_length(x, order)
```

Arguments

x   an object of class TSP or ATSP.

order   optional order of the visited cities as a integer vector or an object of class TOUR. If no order is given, the cities are visited in the original order in x.

Details

If a distance in the tour is infinite, the result is also infinite. If the tour contains positive and negative infinite distances the method returns NA.

See Also

TSP, ATSP, TOUR.

Examples

```
data("USCA50")

## original order
tour_length(USCA50)

## random tour
tour_length(USCA50, sample(1:n_of_cities(USCA50)))
```

TSP   *Class TSP – Symmetric traveling salesperson problem*

Description

Constructor to create an instance of a symmetric traveling salesperson problem (TSP) and some auxiliary methods.

Usage

```
## constructor
TSP(x, labels = NULL)

## coercion
as.TSP(object)

## methods
## S3 method for class 'TSP'
n_of_cities(x)
## S3 method for class 'TSP'
image(x, order, col = gray.colors(64), ...)
## S3 method for class 'TSP'
labels(object, ...)
## S3 method for class 'TSP'
print(x, ...)
```

Arguments

`x, object`	an object (currently `dist` or a symmetric matrix) to be converted into a TSP or, for the methods, an object of class TSP.
`labels`	optional city labels. If not given, labels are taken from `x`.
`col`	color scheme for image.
`order`	order of cities for the image as an integer vector or an object of class TOUR.
`...`	further arguments are passed on.

Details

Objects of class TSP are internally represented as `dist` objects (use `as.dist()` to get the `dist` object).

Value

`TSP()` returns `x` as an object of class TSP.

`n_of_cities()` returns the number of cities in `x`.

`labels()` returns a vector with the names of the cities in `x`.

See Also

TOUR, insert_dummy, tour_length, solve_TSP.

Examples

```
data("iris")
d <- dist(iris[-5])

## create a TSP
tsp <- TSP(d)
tsp

## use some methods
n_of_cities(tsp)
labels(tsp)
image(tsp)
```

TSPLIB   *Read and write TSPLIB files*

Description

Reads and writes TSPLIB format files. TSPLIB files can be used by most TSP solvers. Sample instances for the TSP in TSPLIB format are available on the TSPLIB homepage (see references).

Usage

```
write_TSPLIB(x, file, precision = 6, inf = NULL, neg_inf = NULL)
read_TSPLIB(file, precision = 0)
```

Arguments

x	an object of class TSP or ATSP.
file	file name or a connection.
precision	controls the number of decimal places used to represent distances (see details). If x already is integer, this argument is ignored and x is used as is.
inf	replacement value for Inf (TSPLIB format cannot handle Inf). If inf is NULL, the default value of 2 times the maximum value in x (ignoring the infinity entries) is used.
neg_inf	replacement value for -Inf. If no value is specified, the default value of 2 times the smallest neg. number is used. If x only contains positive values, -1 is used.

Details

In the TSPLIB format distances are represented by integer values. Therefore, if x contains double values (which is normal in R) the values given in x are multiplied by $10^{precision}$ before coercion to integer. Note that therefore all results produced by programs using the TSPLIB file as input need to be divided by $10^{precision}$ (i.e., the decimal point has to be shifted precision placed to the left).

Currently only EDGE_WEIGHT_TYPE EXPLICIT is implemented!

Value

read_TSPLIB returns an object of class TSP or ATSP.

References

TSPLIB home page, <http://www.iwr.uni-heidelberg.de/groups/comopt/software/TSPLIB95/>

USCA   *USCA312/USCA50 – 312/50 cities in the US and Canada*

Description

The USCA312 dataset contains the distances between 312 cities in the US and Canada as an object of class TSP. USCA50 is a subset of USCA312 containing only the first 50 cities.

The USCA312_map dataset contains spatial data of the 312 cities.

Usage

```
data("USCA312")
data("USCA312_map")
data("USCA50")
```

Format

USCA312 and USCA50 are objects of class TSP. USCA312_map contains in USCA312_coords the spatial coordinates of the 312 cities and in USCA312_basemap a part of the map of North America.

Details

For USCA312_map several packages for geographic data are needed (see Examples section).

We want to thank Roger Bivand for his help with plotting the map.

Source

John Burkardt, CITIES – City Distance Datasets, http://www.csit.fsu.edu/~burkardt/datasets/cities/cities.html

Examples

```
data("USCA312")
data("USCA312_map")

## calculate a tour
tour <- solve_TSP(USCA312)
tour

## load map tools
library("maps")
library("sp")
library("maptools")

## plot map
plot(as(USCA312_coords, "Spatial"), axes=TRUE)
plot(USCA312_basemap, add=TRUE, col = "gray")


## plot tour and add cities
tour_line <- SpatialLines(list(Lines(list(
    Line(USCA312_coords[c(tour, tour[1]),])), ID="1")))

plot(tour_line, add=TRUE, col = "red")
points(USCA312_coords, pch=3, cex=0.4, col="black")
```

Index