Package ‘localsolver’

February 20, 2015

Type Package

Title R API to LocalSolver

Description The package converts R data onto input and data for LocalSolver, executes optimization and exposes optimization results as R data. LocalSolver (http://www.localsolver.com/) is an optimization engine developed by Innovation24 (http://www.innovation24.fr/). It is designed to solve large-scale mixed-variable non-convex optimization problems. The localsolver package is developed and maintained by WLOG Solutions (http://www.wlogsolutions.com/en/) in collaboration with Decision Support and Analysis Division at Warsaw School of Economics (http://www.sgh.waw.pl/en/).

Version 2.3

Encoding UTF-8

Date 2014-04-07

Author Walerian Sokolowski [aut, cre, cph],
Wit Jakuczun [aut, cph],
Natalia Okinczyc [aut],
Bogumil Kaminski [aut]

License LGPL-2.1

Depends R (>= 3.0.1)

Suggests knitr

SystemRequirements At least trial version of LocalSolver to be downloaded from http://www.localsolver.com/download.html

VignetteBuilder knitr

Maintainer Walerian Sokolowski `<walerian.sokolowski@wlogsolutions.com>`

NeedsCompilation no

Repository CRAN

Date/Publication 2014-06-18 00:29:05

R topics documented:

`add.output.expr` *Add expression to output of ls.problem.*

Description

Added expression will be printed out by the LocalSolver and added to `ls.solve` result.

Usage

```
add.output.expr(lsp, expr.text.lsp, dimensions = 1)
```

Arguments

`lsp`	problem instance created with `ls.problem`.
`expr.text.lsp`	text of expression in LSP language (an objective function, constraint or decision variable name).
`dimensions`	vector of variables expected dimensions. 1 for a number, length of a vector or dimensions of resulting matrix or array (see `array` `dim` parameter). The vector of the dimensions must be of length 1, 2 or 3, as the library does not maintain arrays of more than 3 dimensions.

Details

Each added expression is extracted out of LocalSolver output according to `dimensions` provided. Extracted values are converted into R data structures. If `dimension` is 1 the expression is considered to be a number. Otherwise the expression is converted to R `array` with `dimensions` passed as dim(see `array`).

All output expression values are exposed as `numeric`s.

Currently errors in expression and inconsistency in dimensions passed are not detected properly. Handling such situations is planned to be implemented in next localsolver package version.

Value

Updated `ls.problem` instance.

Examples

```
model.text.lsp <- lsp.model.example('extdata/knapsack.txt')
lsp <- ls.problem(model.text.lsp)
lsp <- add.output.expr(lsp, "knapsackWeight")
# produces table x[i in 1..5][j in 1..10] in LocalSolver output
# and array with dims = c(5,10) under name x in output of ls.solve.
lsp <- add.output.expr(lsp, "x", c(5, 10))
```

`clear.output.exprs` *Remove all output expressions (see: `add.output.expr`).*

Description

Remove all output expressions (see: `add.output.expr`).

Usage

```
clear.output.exprs(lsp)
```

Arguments

`lsp`	problem instance created with `ls.problem`.

Value

updated ls.problem instance.

Examples

```
model.text.lsp <- lsp.model.example('extdata/knapsack.txt')
lsp <- ls.problem(model.text.lsp)
lsp <- add.output.expr(lsp, "knapsackWeight")
lsp <- clear.output.exprs(lsp)
```

`google_example_1_data` *Data set for google_machine_reassignment demo.*

Description

Data set for google_machine_reassignment demo.

`localsolver` *R API to LocalSolver*

Description

The package converts R data onto input and data for LocalSolver, executes optimization and exposes optimization results as R data.

Details

LocalSolver (`http://www.localsolver.com/`) is an optimization engine developed by *Innovation24* (`http://www.innovation24.fr/`). It is designed to solve large-scale mixed-variable non-convex optimization problems.

The localsolver package is result of cooperation of *WLOG Solutions* (`http://www.wlogsolutions.com/en/`) in collaboration Decision Support and Analysis Division at *Warsaw School of Economics* (`http://www.sgh.waw.pl/en/`).

The localsolver package allows for solving optimization problems in R by using the LocalSolver program. It combines efficiency and easiness of model formulation, which characterize the LocalSolver program, with the elasticity of data in R. The model formulation in LSP language is provided by a user as a string or text file, and the input data - as a list. The solution is also a list, with chosen decision variables, constraints and objective function values. Thus the problem can be solved many times (for example, by iterating with different parameter settings) and the result is provided in a form which makes further processing or visualization in R fast and easy.

It is highly recommended that the LocalSolver software is installed before the R package.
The test version (limited to 100 decisions and 1000 variables) can be downloaded from the website: `http://www.localsolver.com/download.html`
The access to the trial license for full version requires contacting the *Innovation24* company. For further information please visit the website: `http://www.localsolver.com/`

Bugs and feature requests can be reported by e-mail: *Walerian Sokolowski <walerian.sokolowski@wlogsolutions.com>*.
We are also looking forward for any future package development suggestions.

`ls.problem` *Create problem instance from model formulated in LSP language.*

Description

Creates problem instance from model LSP code passed. Detects functions in model LSP code.

Usage

```
ls.problem(model.text.lsp, ls.path = NULL)
```

Arguments

`model.text.lsp`	text of model formulated in LSP language.
`ls.path`	path to the LocalSolver executable. Optional argument (NULL by default). If not provided, the LocalSolver will be searched by the system in `PATH` environment variable.

Details

Functions detected in model LSP code must contain `model` function as it main 'workhorse' function of LocalSolver model. They can contain `param`, `display` and any other custom functions.

Because localsolver package defines `input` function based on data passed to `ls.solve` and `output` function base on output expressions added to problem instance with `add.output.expr` these two(`input` and `output`) functions cannot occur in model LSP code passed. Passing them will cause appropriate error.

For mode details see LocalSolver LSP language reference manual: `http://www.localsolver.com/lspreferencemanual.html`.

Value

created ls.problem instance.

Examples

```
model.text.lsp <- lsp.model.example('extdata/knapsack.txt')
lsp <- ls.problem(model.text.lsp)
```

`ls.solve` *Solves a LocalSolver problem on data passed.*

Description

Prepares input and data for LocalSolver, runs the application and parses its output to get resulting values.

Usage

```
ls.solve(lsp, data)
```

Arguments

`lsp`	problem instance created with `ls.problem`.
`data`	named list of data items. Each element of the list should be indexed with the parameter name and should be a number, vector, matrix or array (of dimension 2 or 3) of numbers. The class of the numbers should be either integer (they will be then handled as integer by the LocalSolver) or numeric (LocalSolver will then treat them as elements of class double).

Details

Result of this function is named list of output expressions added to the problem (for description of R data structures form see `add.output.expr`). Parameters set with `set.params` are passed to LocalSolver by means of generation(or modification) of LocalSolver `param` function (see LocalSolver LSP language reference manual `http://www.localsolver.com/lspreferencemanual.html` for more details).

Make sure you pass integers in `data` if you want them to be `ints` in LocalSolver. Otherwise they will be considered `doubles`.

Errors occurred in model LSP code (passed while creating problem with `ls.problem`) are handled: They cause error containing original error message and error occurrence context to make it easier to detect potential errors in model LSP code. All other LocalSolver errors (e.g. in output expressions) and interaction errors (between localsolver package and LocalSolver process) are passed to caller without processing.

Value

named list of all output expression values on best solution found.

Examples

```
model.text.lsp <- lsp.model.example('extdata/knapsack.txt')
lsp <- ls.problem(model.text.lsp)
lsp <- set.params(lsp, lsTimeLimit=60)
lsp <- add.output.expr(lsp, "x", 4)
data <- list(nbItems=4L, itemWeights=c(1L,2L,3L,4L), itemValues=c(5,6,7,8), knapsackBound=40L)
result <- ls.solve(lsp, data)
```

`lsp.model.example` *Load LSP model example code.*

Description

Load LSP model example code.

Usage

```
lsp.model.example(example.file)
```

Arguments

`example.file`	example file path relative to packaged root.

Value

text of model formulated in LSP language.

Examples

```
lsp.model.example('extdata/knapsack.txt')
```

`print.ls.problem` *Prints an object of class ls.problem.*

Description

Prints an object of class ls.problem.

Usage

```
## S3 method for class 'ls.problem'
print(x, ...)
```

Arguments

`x`	problem instance created with `ls.problem`.
`...`	further arguments passed to or from other methods.

Examples

```
model.text.lsp <- lsp.model.example('extdata/knapsack.txt')
lsp <- ls.problem(model.text.lsp)
lsp
```

`reset.lsp.params` *Reset all ls.problem instance parameters.*

Description

Problem parameters can be set with `set.params`. This method resets all their values to defaults, which have been described in help for the `set.params` function.

Usage

```
reset.lsp.params(lsp)
```

Arguments

`lsp`	the lsp object whose parameters are to be reset.

Value

lsp object with all parameters reset to their default values.

Examples

```
modelText <- lsp.model.example('extdata/knapsack.txt')
lsp <- ls.problem(modelText)
lsp$params
lsp <- set.params(lsp, lsTimeLimit=60, lsSeed=7)
lsp$params
lsp <- reset.lsp.params(lsp)
lsp$params
```

`set.params` *Set ls.problem instance parameters.*

Description

Updates the chosen parameters of an object of class `ls.problem`.

Usage

```
set.params(lsp, lsTimeLimit = lsp$params$lsTimeLimit,
  lsIterationLimit = lsp$params$lsIterationLimit,
  lsTimeBetweenDisplays = lsp$params$lsTimeBetweenDisplays,
  lsSeed = lsp$params$lsSeed, lsNbThreads = lsp$params$lsNbThreads,
  lsAnnealingLevel = lsp$params$lsAnnealingLevel,
  lsVerbosity = lsp$params$lsVerbosity,
  indexFromZero = lsp$params$indexFromZero)
```

Arguments

`lsp`	problem instance created with `ls.problem`.
`lsTimeLimit`	the number of the seconds which will be spent to optimize the objective function (functions), or a vector of times (in seconds) assigned to each objective function. The length of the vector should correspond to the length of the number of objective functions.
`lsIterationLimit`	the number of iterations made to optimize the objective function (functions), or a vector of iteration numbers assigned to each objective function. The length of the vector should correspond to the length of the number of objective functions.
`lsTimeBetweenDisplays`	the time (in seconds) between successive displays of the information about the search (default: 1)
`lsSeed`	pseudo-random number generator seed (default: 0).
`lsNbThreads`	the number of threads over which the search is paralleled (default: 2).
`lsAnnealingLevel`	simulated annealing level (no annealing: 0, default: 1).
`lsVerbosity`	verbosity (no display: 0, default: 1).
`indexFromZero`	indicates whether the data and decision variables (vectors and matrices) are to be indexed from 0. If `FALSE` (by default), they will be indexed from 1.

Value

updated ls.problem instance.

Examples

```
model.text.lsp <- lsp.model.example('extdata/knapsack.txt')
lsp <- ls.problem(model.text.lsp)
lsp <- set.params(lsp, lsTimeLimit=10, lsIterationLimit= 5)
```

`set.temp.dir` *Sets folder to use for the problem instance solving process temporary data.*

Description

Exposed for technical reasons. Temporary folder is used to store files for communication with LocalSolver application. By default system received temp folder is used. Setting temporary folder is useful in case `ls.solve` is performed in parallel. In that case each call should use own lsp instance with dedicated temporary folder. In case of changing this directory, it is important to choose a path to a folder with write access.

Usage

```
set.temp.dir(lsp, path)
```

Arguments

`lsp`	problem instance created with `ls.problem`.
`path`	the directory, which will be used for temporary data.

Examples

```
model.text.lsp <- lsp.model.example('extdata/knapsack.txt')
lsp <- ls.problem(model.text.lsp)
lsp <- set.temp.dir(lsp, tempdir())
```

Index