Package 'ompr'

September 9, 2023

Type Package

Title Model and Solve Mixed Integer Linear Programs

Version 1.0.4

Description Model mixed integer linear programs in an algebraic way directly in R.
The model is solver-independent and thus offers the possibility
to solve a model with different solvers. It currently only supports
linear constraints and objective functions. See the 'ompr'
website <https://dirkschumacher.github.io/ompr/> for more information,
documentation and examples.

License MIT + file LICENSE

RoxygenNote 7.2.3

Encoding UTF-8

URL https://github.com/dirkschumacher/ompr

BugReports https://github.com/dirkschumacher/ompr/issues

Depends R (>= 3.5.0)

Imports lazyeval, rlang (>= 0.2.0), listcomp (>= 0.4.0), methods,
data.table, Matrix, fastmap

Suggests covr, magrittr, testthat

ByteCompile Yes

Collate 'abstract-model-impl.R' 'helper.R'
'linear-optimization-model-impl.R'
'linear-optimization-model-linear-constraints.R'
'linear-optimization-model-linear-functions.R' 'model-api.R'
'milp-impl.R' 'milp-linearopt-variables.R' 'ompr-package.R'
'solution-api.R' 'solution-impl.R'

NeedsCompilation no

Author Dirk Schumacher [aut, cre]

Maintainer Dirk Schumacher <mail@dirk-schumacher.net>

Repository CRAN

Date/Publication 2023-09-09 08:40:02 UTC

R topics documented:

`additional_solver_output` *Retrieve additional solver specific output*

Description

Retrieve additional solver specific output

Usage

```
additional_solver_output(solution)
```

Arguments

`solution`	a solution object

Value

A list of named entries. What is in that list is determined by the solver function. For `ompr.roi` this is usually a solver specific message and status information.

add_constraint *Add a constraint*

Description

Add one or more constraints to the model using quantifiers.

Usage

```
add_constraint(.model, .constraint_expr, ..., .show_progress_bar = TRUE)

add_constraint_(
  .model,
  .constraint_expr,
  ...,
  .dots,
  .show_progress_bar = TRUE
)
```

Arguments

.model	the model
.constraint_expr	the constraint. Must be a linear (in)equality with operator "<=", "==" or ">=".
...	quantifiers for the indexed variables. For all combinations of bound variables a new constraint is created. In addition you can add filter expressions
.show_progress_bar	displays a progressbar when adding multiple constraints
.dots	Used to work around non-standard evaluation.

Value

a Model with new constraints added

Examples

```
library(magrittr)
MIPModel() %>%
  add_variable(x[i], i = 1:5) %>%
  # creates 5 constraints
  add_constraint(x[i] >= 1, i = 1:5) %>%
  # you can also use filter expressions
  add_constraint(x[i] >= 1, i = 1:5, i %% 2 == 0) %>%
  # and depent on other indexes
  add_constraint(x[j] >= 1, i = 1:10, j = 1:i, j <= 5)
```

`add_variable` *Add a variable to the model*

Description

A variable can either be a name or an indexed name. See examples.

Usage

```
add_variable(.model, .variable, ..., type = "continuous", lb = -Inf, ub = Inf)

add_variable_(
  .model,
  .variable,
  ...,
  type = "continuous",
  lb = -Inf,
  ub = Inf,
  .dots
)
```

Arguments

`.model`	the model
`.variable`	the variable name/definition
`...`	quantifiers for the indexed variable. Including filters
`type`	must be either continuous, integer or binary
`lb`	the lower bound of the variable
`ub`	the upper bound of the variable
`.dots`	Used to work around non-standard evaluation.

Examples

```
library(magrittr)
MIPModel() %>%
  add_variable(x) %>% # creates 1 variable named x
  add_variable(y[i],
    i = 1:10, i %% 2 == 0,
    type = "binary"
  ) # creates 4 variables
```

as_colwise *As_colwise*

Description

Convert lists or vectors to colwise semantic.

Usage

```
as_colwise(x)
```

Arguments

x	a list of numeric vectors or a numeric vector

colwise *Format variables colwise*

Description

This function should be used if you to expand a variable across columns and not rows. When passing a vector of indexes to MILPModel variable, it creates a new row for each vector element. With colwise you can create columns instead. Please see the examples below.

Usage

```
colwise(...)
```

Arguments

...	create a colwise vector

Details

'colwise' is probably the concept that is likely to change in the future.

Examples

```
## Not run:
# vectors create matrix rows
# x[1, 1]
# x[2, 1]
# x[3, 1]
x[1:3, 1]

# colwise() creates columns per row
# 1 * x[1, 1] + 2 * x[1, 2] + 3 * x[1, 3]
```

```
colwise(1, 2, 3) * x[1, colwise(1, 2, 3)]

# or you have multiple rows and columns and different coefficients
# 1 * x[1, 1] + 2 * x[1, 2] + 3 * x[1, 3]
# 4 * x[2, 1] + 5 * x[2, 2] + 6 * x[1, 3]
colwise(1:6) * x[1:2, colwise(1:3)]
# in the example above, the colwise vector multiplied with the variable
# has an element per row and column
# in general, it can be a multiple of number of columns

# you can also combine the two
# x[1, 1]
# x[2, 1] + x[2, 2]
# x[3, 1] + x[3, 2] + x[3, 2]
x[1:3, colwise(1, 1:2, 1:3)]

## End(Not run)
```

`extract_constraints` *Extract the constraint matrix, the right hand side and the sense from a model*

Description

Extract the constraint matrix, the right hand side and the sense from a model

Usage

```
extract_constraints(model)
```

Arguments

`model`	the model

Value

a list with three named elements. 'matrix' the (sparse) constraint matrix from the Matrix package. 'rhs' is the right hand side vector in the order of the matrix. 'sense' is a vector of the constraint senses

Examples

```
library(magrittr)
model <- MIPModel() %>%
  add_variable(x[i], i = 1:3) %>%
  add_variable(y[i], i = 1:3) %>%
  add_constraint(x[i] + y[i] <= 1, i = 1:3)
extract_constraints(model)
```

get_column_duals *Gets the column duals of a solution*

Description

Gets the column duals of a solution

Usage

```
get_column_duals(solution)
```

Arguments

solution	a solution

Value

Either a numeric vector with one element per column or 'NA_real_'.

Examples

```
## Not run:
result <- MIPModel() %>%
  add_variable(x[i], i = 1:5) %>%
  add_variable(y[i, j], i = 1:5, j = 1:5) %>%
  add_constraint(x[i] >= 1, i = 1:5) %>%
  set_bounds(x[i], lb = 3, i = 1:3) %>%
  set_objective(sum_over(i * x[i], i = 1:5)) %>%
  solve_model(with_ROI("glpk"))

get_column_duals(result)

## End(Not run)
```

get_row_duals *Gets the row duals of a solution*

Description

Gets the row duals of a solution

Usage

```
get_row_duals(solution)
```

Arguments

solution	a solution

Value

Either a numeric vector with one element per row or 'NA_real_'.

Examples

```
## Not run:
result <- MIPModel() %>%
  add_variable(x[i], i = 1:5) %>%
  add_variable(y[i, j], i = 1:5, j = 1:5) %>%
  add_constraint(x[i] >= 1, i = 1:5) %>%
  set_bounds(x[i], lb = 3, i = 1:3) %>%
  set_objective(sum_expr(i * x[i], i = 1:5)) %>%
  solve_model(with_ROI("glpk"))

get_row_duals(result)

## End(Not run)
```

---

get_solution *Get variable values from a solution*

Description

Get variable values from a solution

Usage

```
get_solution(solution, expr, type = "primal")

get_solution_(solution, expr, type = "primal")
```

Arguments

solution	the solution object
expr	a variable expression. You can partially bind indexes.
type	optional, either "primal" or "dual". The default value is "primal". If "primal" it returns the primal solution, otherwise the column duals. Especially the dual values depend on the solver. If no duals are calculated, the function stops with an error message.

Value

a data.frame. One row for each variable instance and a column for each index. Unless it is a single variable, then it returns a single number. Please note that in case of a `data.frame` there is no guarantee about the ordering of the rows. This could change in future `ompr` versions. Please always use the indexes to retrieve the correct values.

Examples

```
## Not run:
library(magrittr)
result <- MIPModel() %>%
  add_variable(x[i], i = 1:5) %>%
  add_variable(y[i, j], i = 1:5, j = 1:5) %>%
  add_constraint(x[i] >= 1, i = 1:5) %>%
  set_bounds(x[i], lb = 3, i = 1:3) %>%
  set_objective(0) %>%
  solve_model(with_ROI("glpk"))
solution <- get_solution(result, x[i])
solution2 <- get_solution(result, y[i, 1])
solution3 <- get_solution(result, y[i, j])
duals <- get_solution(result, x[i], type = "dual")

## End(Not run)
```

MILPModel *Experimental: Create a new MILP Model*

Description

Create an an empty mixed-integer linear programming model that is about 1000 times faster than 'MIPModel'.

Usage

```
MILPModel()
```

Details

Please only use it if you can deal with potential API changes in the future. When you use 'MILPModel' make sure to always model your problem with 'MIPModel' as well, just to make sure you get the same results.

It is also always a good idea to test your model with very small input sizes and examine the coefficients and rows of the constraint matrix.

MIPModel *Create a new MIP Model*

Description

Create a new MIP Model

Usage

```
MIPModel()
```

nconstraints *Number of variables (rows) of the model*

Description

Number of variables (rows) of the model

Usage

```
nconstraints(model)
```

Arguments

model	the model

Value

An integer equal to the number of variables. A variable is here a column in the resulting constraint matrix.

Examples

```
library(magrittr)
model <- MIPModel() %>%
  add_variable(x) %>%
  add_variable(y[i], i = 1:10)
nconstraints(model) # 11
```

new_solution *Create a new solution*

Description

This function/class should only be used if you develop your own solver.

Usage

```
new_solution(
  model,
  objective_value,
  status,
  solution,
  solution_column_duals = function() NA_real_,
  solution_row_duals = function() NA_real_,
  additional_solver_output = list()
)
```

Arguments

`model`	the optimization model that was solved
`objective_value`	a numeric objective value
`status`	the status of the solution
`solution`	a named numeric vector containing the primal solution values
`solution_column_duals`	A function without arguments that returns a numeric vector containing the column dual solution values. 'NA_real_', if no column duals are available/defined.
`solution_row_duals`	A function without arguments that returns a numeric vector containing the column dual solution values. 'NA_real_', if no column duals are available/defined.
`additional_solver_output`	A named list of additional solver information

nvars *Number of variables of a model*

Description

Number of variables of a model

Usage

```
nvars(model)
```

Arguments

model	the model

Value

a list with three named elements. 'binary' => number of binary variables, 'integer' => number of integer variables, 'continuous' => number of continuous variables.

Examples

```
library(magrittr)
model <- MIPModel() %>%
  add_variable(x[i], i = 1:10, type = "binary") %>%
  add_variable(y[i], i = 1:5, type = "continuous") %>%
  add_variable(z[i], i = 1:2, type = "integer")
nvars(model)
```

---

objective_function *Extract the objective function from a model*

Description

Extract the objective function from a model

Usage

```
objective_function(model)
```

Arguments

model	the model

Value

a list with two named elements, 'solution' and 'constant'. 'solution' is a sparse vector from the Matrix package. 'constant' is a constant that needs to be added to get the final obj. value.

Examples

```
library(magrittr)
model <- MIPModel() %>%
  add_variable(x[i], i = 1:5) %>%
  set_objective(sum_over(i * x[i], i = 1:5) + 10)
objective_function(model)
```

objective_value *Extract the numerical objective value from a solution*

Description

Extract the numerical objective value from a solution

Usage

```
objective_value(solution)
```

Arguments

solution	a solution

Value

numeric single item vector

set_bounds *Set the bounds of a variable*

Description

Change the lower and upper bounds of a named variable, indexed variable or a group of variables.

Usage

```
set_bounds(.model, .variable, ..., lb = NULL, ub = NULL)

set_bounds_(.model, .variable, ..., lb = NULL, ub = NULL, .dots)
```

Arguments

.model	the model
.variable	the variable name/definition or a linear constraint
...	quantifiers for the indexed variable
lb	the lower bound of the variable.
ub	the upper bound of the variable For MIPModel you can also pass (in)equalities to define bounds. Please look at the examples.
.dots	Used to work around non-standard evaluation.

Examples

```
library(magrittr)
MIPModel() %>%
  add_variable(x[i], i = 1:5) %>%
  add_constraint(x[i] >= 1, i = 1:5) %>% # creates 5 constraints
  set_bounds(x[i], lb = 3, i = 1:3) %>%
  variable_bounds()

MIPModel() %>%
  add_variable(x[i], i = 1:5) %>%
  set_bounds(x[i] <= i, i = 1:5) %>% # upper bound
  set_bounds(x[i] >= 0, i = 1:5) %>% # lower bound
  set_bounds(x[5] == 45) %>%
  variable_bounds()
```

---

set_objective *Set the model objective*

Description

Set the model objective

Usage

```
set_objective(model, expression, sense = c("max", "min"))

set_objective_(model, expression, sense = c("max", "min"))
```

Arguments

model	the model
expression	the linear objective as a sum of variables and constants
sense	the model sense. Must be either "max" or "min".

Value

a Model with a new objective function definition

Examples

```
library(magrittr)
MIPModel() %>%
  add_variable(x, lb = 2) %>%
  add_variable(y, lb = 40) %>%
  set_objective(x + y, sense = "min")
```

solver_status *Get the solver status from a solution*

Description

Get the solver status from a solution

Usage

```
solver_status(solution)
```

Arguments

solution	a solution

Value

character vector being either "infeasible", "optimal", "unbounded", "userlimit" or "error

solve_model *Solve a model*

Description

Solve a model

Usage

```
solve_model(model, solver)
```

Arguments

model	the model
solver	a function mapping a model to a solution

Value

solver(model)

sum_over *Sum over indexes*

Description

This functions helps to create summations over indexes.

Usage

```
sum_over(.expr, ...)

sum_expr(.expr, ...)
```

Arguments

.expr	an expression that can be expanded to a sum
...	bind variables in expr using dots. See examples.

Value

the sum over all the indexes

See Also

add_constraint

set_objective

Please note that sum_expr is deprecated when used together with MIPModel.

Examples

```
if (FALSE) {
  # create a sum from x_1 to x_10
  sum_over(x[i], i = 1:10)
  # create a sum from x_2 to x_10 with even indexes
  sum_over(x[i], i = 1:10, i %% 2 == 0)
  sum_over(x[i, j], i = 1:10, j = 1:i)
}
```

variable_bounds *Variable lower and upper bounds of a model*

Description

Variable lower and upper bounds of a model

Usage

```
variable_bounds(model)
```

Arguments

model	the model

Value

a list with two components 'lower' and 'upper' each having a numeric vector of bounds. One for each variable.

Examples

```
library(magrittr)
model <- MIPModel() %>%
  add_variable(x, type = "binary") %>%
  add_variable(y, type = "continuous", lb = 2) %>%
  add_variable(z, type = "integer", ub = 3)
variable_bounds(model)
```

variable_keys *Get all unique names of the model variables*

Description

Get all unique names of the model variables

Usage

```
variable_keys(model)
```

Arguments

model	the model

Value

a character vector ordered in the same way as the constraint matrix columns and objective vector

Examples

```
library(magrittr)
model <- MIPModel() %>%
  add_variable(x[i], i = 1:3)
variable_keys(model)
```

`variable_types` *Variable types of a model*

Description

One component for each variable in the correct order

Usage

```
variable_types(model)
```

Arguments

`model`	the model

Value

a factor with levels binary, continuous, integer

Examples

```
library(magrittr)
model <- MIPModel() %>%
  add_variable(x, type = "binary") %>%
  add_variable(y, type = "continuous") %>%
  add_variable(z, type = "integer")
variable_types(model)
```

Index