Complete_Algorithm

Benhao Gu

2026-01-08

```
library(gurobi)
```

```
## Warning: package 'gurobi' was built under R version 4.5.0
```

```
## Loading required package: slam
```

```
library(Matrix)
```

```
# Input format: DD/MM/2025, output format: an integer number 1 to 365
# Note: the year MUST be 2025
# Converts DD/MM/YYYY to day of year (1-365) relative to 2025
date_to_int <- function(date_str) {
  dt <- as.Date(date_str, format = "%d/%m/%Y")
  base <- as.Date("2025-01-01")
  return(as.numeric(dt - base) + 1)
}
```

DATA INPUT

```
# Source Data Frame
sources <- data.frame(
  ID = c("FS001", "FS002", "FS003", "FS004", "FS005", "FS006", "FS007", "FS008", "FS009", "FS010"),
  # Here we have use I(list(...)) is because we want to store multiple categories (with different numbe
  Categories = I(list(
    c("Salary"), c("Equipment"), c("Travel"), c("Salary", "Travel"),
    c("Equipment", "Travel"), c("Salary"), c("Equipment"), c("Travel"),
    c("Salary", "Equipment"), c("Salary", "Equipment", "Travel")
  )),
  ValidFrom = c("01/02/2025", "01/02/2025", "01/03/2025", "01/02/2025", "01/04/2025",
                "01/06/2025", "01/01/2025", "01/05/2025", "01/07/2025", "01/02/2025"),
  ValidTo = c("30/06/2025", "31/08/2025", "30/09/2025", "31/12/2025", "31/10/2025",
              "31/12/2025", "31/12/2025", "30/11/2025", "31/12/2025", "31/12/2025"),
  Amount = c(15000, 12000, 8000, 20000, 10000, 18000, 36000, 5000, 14000, 10000)
)

sources$Categories
```

```
## [[1]]
## [1] "Salary"
```

```
##
## [[2]]
## [1] "Equipment"
##
## [[3]]
## [1] "Travel"
##
## [[4]]
## [1] "Salary" "Travel"
##
## [[5]]
## [1] "Equipment" "Travel"
##
## [[6]]
## [1] "Salary"
##
## [[7]]
## [1] "Equipment"
##
## [[8]]
## [1] "Travel"
##
## [[9]]
## [1] "Salary"    "Equipment"
##
## [[10]]
## [1] "Salary"    "Equipment" "Travel"
```

```r
# Expense Data Frame
expenses <- data.frame(
  ID = c("E009", "E014", "E015", "E013", "E001", "E002", "E003", "E004", "E005",
         "E006", "E007", "E008", "E010", "E011", "E012"),
  Category = c("Travel", "Equipment", "Travel", "Salary", "Salary", "Equipment",
               "Travel", "Salary", "Equipment", "Travel", "Salary", "Equipment",
               "Salary", "Equipment", "Travel"),
  Amount = c(6000, 20000, 15000, 10000, 5000, 8000, 3000, 12000, 15000, 4000,
             8000, 10000, 15000, 12000, 5000),
  Date = c("10/08/2025", "20/12/2025", "25/12/2025", "01/01/2025", "15/02/2025",
           "20/02/2025", "10/03/2025", "15/04/2025", "20/05/2025", "10/06/2025",
           "15/07/2025", "20/07/2025", "15/09/2025", "20/10/2025", "10/11/2025")
)

# getting total number of funding source and expenses (NOT total amount)
n_sources <- nrow(sources)
n_expenses <- nrow(expenses)
```

PRE-PROCESSING

```r
# This is a matrix with size n_sources x n_expenses (row is each funding sources, and column is each ex
# We build a Compatibility Matrix (Valid = 1, Invalid = 0)
compatibility <- matrix(0, nrow = n_sources, ncol = n_expenses)

for (i in 1:n_sources) {
```

```
  for (j in 1:n_expenses) {
    # 1. Category Check
    # Expense category must be in the Source's allowed list
    cat_match <- expenses$Category[j] %in% sources$Categories[[i]]

    # 2. Time Validity Check
    # Convert dates to integers (Day of Year)
    s_from <- date_to_int(sources$ValidFrom[i])
    s_to   <- date_to_int(sources$ValidTo[i])
    e_date <- date_to_int(expenses$Date[j]) # Assumes this is the actual payment date

    # LOGIC: The payment date must be INSIDE the funding window (Inclusive)
    # ValidFrom <= PaymentDate <= ValidTo
    time_match <- (e_date >= s_from & e_date <= s_to)

    # Combine Checks
    if (cat_match && time_match) {
      compatibility[i, j] <- 1
    } else {
      compatibility[i, j] <- 0
    }
  }
}
```

MODEL CONSTRUCTION

```
# Maximising logic: maximise the
model <- list()
model$modelsense <- "max"

# --- A. Objective Function ---
# Maximise Sum(Weight_j * y_j)
# Coefficients for x[i,j] are 0. Coefficients for y[j] are the weights.
# It works because $$2^k > \sum_{i=0}^{k-1} 2^i$$, so it will always incentivise to use the funding to

weights <- 2^(n_expenses - 1 - (0:(n_expenses-1))) # Powers of 2 descending
obj_x <- rep(0, n_sources * n_expenses)             # Zeros for x vars
model$obj <- c(obj_x, weights)                      # Combine

# --- B. Variable Types & Bounds ---
# x is Continuous ('C'), y is Binary ('B')
model$vtype <- c(rep('C', n_sources * n_expenses), rep('B', n_expenses))

# Variable Bounds (lb is 0 by default, ub is Inf)
# We use UB to enforce the "Compatibility Constraint"
# If compatibility[i,j] == 0, then x[i,j] must be 0.
ub_x <- rep(Inf, n_sources * n_expenses)
idx <- 0
for (i in 1:n_sources) {
  for (j in 1:n_expenses) {
    idx <- idx + 1
    if (compatibility[i, j] == 0) {
      ub_x[idx] <- 0
```

```
    }
  }
}
model$ub <- c(ub_x, rep(1, n_expenses)) # y vars are binary (0-1)

# --- C. Constraint Matrix (A) ---
# We will build rows for the matrix A
rows <- list()
rhs <- c()
sense <- c()

# Constraint 1: Supply (For each Source i: Sum(x_ij) <= S_i)
for (i in 1:n_sources) {
  # Create a row of zeros
  row_vec <- rep(0, n_sources * n_expenses + n_expenses)

  # Identify indices for x[i, *]
  # Start index for this source block
  start_idx <- (i - 1) * n_expenses + 1
  end_idx <- i * n_expenses

  # Set coefficients to 1 for this source's allocations
  row_vec[start_idx:end_idx] <- 1

  rows[[length(rows) + 1]] <- row_vec
  rhs <- c(rhs, sources$Amount[i])
  sense <- c(sense, "<=")
}

# Constraint 2: Demand Linking (For each Expense j: Sum(x_ij) - D_j * y_j = 0)
# Rearranged: Sum(x_ij) + (-D_j) * y_j = 0
for (j in 1:n_expenses) {
  row_vec <- rep(0, n_sources * n_expenses + n_expenses)

  # Identify indices for x[*, j] (This is strided/every Nth element)
  # Also identify index for y[j]

  # 1. Fill x coefficients (1 for every source pointing to this expense)
  for (i in 1:n_sources) {
    idx_x <- (i - 1) * n_expenses + j
    row_vec[idx_x] <- 1
  }

  # 2. Fill y coefficient (-Demand)
  idx_y <- (n_sources * n_expenses) + j
  row_vec[idx_y] <- -expenses$Amount[j]

  rows[[length(rows) + 1]] <- row_vec
  rhs <- c(rhs, 0)
  sense <- c(sense, "=")
}

# Combine rows into a Sparse Matrix
```

```r
model$A <- do.call(rbind, rows)
model$A <- Matrix(model$A, sparse = TRUE) # Convert to sparse for Gurobi
model$rhs <- rhs
model$sense <- sense
```

OPTIMIZE

```r
result <- gurobi(model)
```

```
## Set parameter Username
## Set parameter LicenseID to value 2752712
## Academic license - for non-commercial use only - expires 2026-12-11
## Gurobi Optimizer version 13.0.0 build v13.0.0rc1 (win64 - Windows 10.0 (19045.2))
##
## CPU model: AMD Ryzen 7 PRO 4750U with Radeon Graphics, instruction set [SSE2|AVX|AVX2]
## Thread count: 8 physical cores, 16 logical processors, using up to 16 threads
##
## Optimize a model with 25 rows, 165 columns and 315 nonzeros (Max)
## Model fingerprint: 0x4c69c375
## Model has 15 linear objective coefficients
## Variable types: 150 continuous, 15 integer (15 binary)
## Coefficient statistics:
##   Matrix range     [1e+00, 2e+04]
##   Objective range  [1e+00, 2e+04]
##   Bounds range     [1e+00, 1e+00]
##   RHS range        [5e+03, 4e+04]
## Found heuristic solution: objective -0.0000000
## Presolve removed 1 rows and 100 columns
## Presolve time: 0.00s
## Presolved: 24 rows, 65 columns, 119 nonzeros
## Variable types: 51 continuous, 14 integer (14 binary)
##
## Root relaxation: objective 3.071900e+04, 16 iterations, 0.00 seconds (0.00 work units)
##
##     Nodes    |    Current Node    |     Objective Bounds      |     Work
##  Expl Unexpl |  Obj  Depth IntInf | Incumbent    BestBd   Gap | It/Node Time
##
## *    0     0               0    30719.000000 30719.0000  0.00%     -    0s
##
## Explored 1 nodes (16 simplex iterations) in 0.00 seconds (0.00 work units)
## Thread count was 16 (of 16 available processors)
##
## Solution count 2: 30719 -0
##
## Optimal solution found (tolerance 1.00e-04)
## Best objective 3.071900000000e+04, best bound 3.071900000000e+04, gap 0.0000%
```

RESULT

```r
if (result$status == "OPTIMAL") {
  cat("\n============================================================\n")
  cat(sprintf("%-60s\n", "                    FINAL SOLUTION REPORT                    "))
```

```r
cat("============================================================\n")

# Extract Solution Vectors
sol <- result$x
# Split into x (allocations) and y (binary status)
sol_x <- sol[1:(n_sources * n_expenses)]
sol_y <- sol[(n_sources * n_expenses + 1):length(sol)]

# Reshape sol_x back to matrix for easy reading
x_matrix <- matrix(sol_x, nrow = n_sources, ncol = n_expenses, byrow = TRUE)

# --- 1. FILLED EXPENSES ---
cat("\n--- Filled Expenses ---\n")
filled_indices <- which(sol_y > 0.5)
unfilled_indices <- which(sol_y < 0.5)

for (j in filled_indices) {
  cat(sprintf("%s: %s ($%s)\n",
              expenses$ID[j], expenses$Category[j],
              format(expenses$Amount[j], big.mark=",")))
}

# --- 2. UNFILLED EXPENSES ---
cat("\n--- nfilled Expenses ---\n")
for (j in unfilled_indices) {
  cat(sprintf("%s: %s ($%s) - Due: %s\n",
              expenses$ID[j], expenses$Category[j],
              format(expenses$Amount[j], big.mark=","),
              expenses$Date[j]))
}

cat(sprintf("\nSummary: %d Filled / %d Missed\n", length(filled_indices), length(unfilled_indices)))

# --- 3. ALLOCATION DETAILS ---
cat("\n--- Allocation Details ---\n")
cat(sprintf("%-8s %-8s %-15s %s\n", "Source", "Expense", "Exp. Category", "Amount Allocated"))
cat("------------------------------------------------------------\n")

for (i in 1:n_sources) {
  for (j in filled_indices) {
    val <- x_matrix[i, j]
    if (val > 1e-6) {
      cat(sprintf("%-8s -> %-8s %-15s $%s\n",
                  sources$ID[i], expenses$ID[j], expenses$Category[j],
                  format(val, nsmall=2, big.mark=",")))
    }
  }
}

# --- 4. REMAINING BALANCES ---
cat("\n--- Remaining Fund Balances ---\n")
cat(sprintf("%-8s %-10s %-10s %-10s %s\n", "Fund ID", "Initial", "Used", "Remaining", "Allowed Categor
cat("------------------------------------------------------------------\n")
```

```r
  total_unused <- 0
  for (i in 1:n_sources) {
    used <- sum(x_matrix[i, ])
    remaining <- sources$Amount[i] - used
    if (remaining < 1e-6) remaining <- 0
    total_unused <- total_unused + remaining

    cats_str <- paste(unlist(sources$Categories[i]), collapse = ", ")

  cat(sprintf("%-8s %-10s %-10s %-10s %s\n",
              sources$ID[i],
              format(sources$Amount[i], big.mark=",", nsmall=0),
              format(round(used), big.mark=",", nsmall=0),
              format(round(remaining), big.mark=",", nsmall=0),
              cats_str))
  }
  cat("---------------------------------------------------------------------\n")
  cat(sprintf("TOTAL UNUSED FUNDS: $%s\n", format(total_unused, big.mark=",")))

} else {
  cat("No optimal solution found.\n")
}
```

```
##
## ============================================================
##                     FINAL SOLUTION REPORT
## ============================================================
##
## --- Filled Expenses ---
## E009: Travel ($6,000)
## E014: Equipment ($20,000)
## E015: Travel ($15,000)
## E001: Salary ($5,000)
## E002: Equipment ($8,000)
## E003: Travel ($3,000)
## E004: Salary ($12,000)
## E005: Equipment ($15,000)
## E006: Travel ($4,000)
## E007: Salary ($8,000)
## E008: Equipment ($10,000)
## E010: Salary ($15,000)
## E011: Equipment ($12,000)
## E012: Travel ($5,000)
##
## --- nfilled Expenses ---
## E013: Salary ($10,000) - Due: 01/01/2025
##
## Summary: 14 Filled / 1 Missed
##
## --- Allocation Details ---
## Source   Expense  Exp. Category   Amount Allocated
## ------------------------------------------------------------
## FS001    -> E001     Salary          $3,000.00
```

```
## FS001    -> E004     Salary          $12,000.00
## FS002    -> E005     Equipment       $2,000.00
## FS002    -> E008     Equipment       $10,000.00
## FS003    -> E009     Travel          $6,000.00
## FS003    -> E003     Travel          $2,000.00
## FS004    -> E015     Travel          $15,000.00
## FS004    -> E001     Salary          $2,000.00
## FS004    -> E007     Salary          $3,000.00
## FS005    -> E005     Equipment       $10,000.00
## FS006    -> E007     Salary          $5,000.00
## FS006    -> E010     Salary          $13,000.00
## FS007    -> E014     Equipment       $13,000.00
## FS007    -> E002     Equipment       $8,000.00
## FS007    -> E005     Equipment       $3,000.00
## FS007    -> E011     Equipment       $12,000.00
## FS009    -> E014     Equipment       $7,000.00
## FS009    -> E010     Salary          $2,000.00
## FS010    -> E003     Travel          $1,000.00
## FS010    -> E006     Travel          $4,000.00
## FS010    -> E012     Travel          $5,000.00
##
## --- Remaining Fund Balances ---
## Fund ID  Initial    Used       Remaining  Allowed Categories
## ------------------------------------------------------------------
## FS001    15,000     15,000     0          Salary
## FS002    12,000     12,000     0          Equipment
## FS003    8,000      8,000      0          Travel
## FS004    20,000     20,000     0          Salary, Travel
## FS005    10,000     10,000     0          Equipment, Travel
## FS006    18,000     18,000     0          Salary
## FS007    36,000     36,000     0          Equipment
## FS008    5,000      0          5,000      Travel
## FS009    14,000     9,000      5,000      Salary, Equipment
## FS010    10,000     10,000     0          Salary, Equipment, Travel
## ------------------------------------------------------------------
## TOTAL UNUSED FUNDS: $10,000
```