Complete_Algorithm

Benhao Gu

2026-01-08

```
# install.packages(c("ompr", "ompr.roi", "ROI", "ROI.plugin.highs"))
library(ompr)
library(ompr.roi)
library(ROI)
```

```
## ROI: R Optimization Infrastructure
```

```
## Registered solver plugins: nlminb, highs.
```

```
## Default solver: auto.
```

```
library(ROI.plugin.highs)
library(magrittr)
```

```
# Input format: DD/MM/YYYY
# Output format: Integer (Days since base_date)
date_to_int <- function(date_str, base_date) {
  dt <- as.Date(date_str, format = "%d/%m/%Y")
  # Return numeric difference + 1 (so the first day is Day 1, not Day 0)
  return(as.numeric(dt - base_date) + 1)
}
```

DATA INPUT

```
# Source Data Frame, assume in the order of preference, first will have the highest allocation priority
# 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 num
#   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", "01/02/2025", "01/02/2025", "01/04/2025",
#                 "01/01/2025", "01/01/2025", "01/05/2025", "01/07/2025", "01/02/2025"),
#   ValidTo = c("30/06/2025", "31/08/2025", "30/12/2025", "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, 10000, 20000, 10000, 18000, 36000, 5000, 14000, 10000)
# )
```

```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, 20000, 20000, 5000, 8000, 3000, 12000, 15000, 4000,
#              8000, 10000, 15000, 12000, 10000),
#   Date = c("10/08/2025", "20/12/2025", "25/12/2025", "01/05/2025", "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")
# )

# Funding Source
sources <- data.frame(
  ID = c("FS001", "FS002", "FS003", "FS004", "FS005", "FS006", "FS007"),
  Categories = I(list(
    c("Salary"),
    c("Equipment"),
    c("Travel"),
    c("Salary"),
    c("Equipment"),
    c("Travel"),
    c("Equipment")
  )),
  ValidFrom = c("01/01/2025", "01/02/2025", "01/04/2025", "01/01/2025", "01/07/2025", "01/01/2025", "01/
  ValidTo   = c("31/03/2025", "30/06/2025", "30/09/2025", "31/12/2025", "31/12/2025", "31/12/2025", "31/
  Amount = c(12000, 18000, 10000, 30000, 15000, 8000, 9000)
)

# Expense
expenses <- data.frame(
  ID = c("E001","E002","E003","E004","E005","E006","E007","E008","E009","E010"),
  Category = c("Salary","Equipment","Salary","Travel","Equipment","Salary","Travel","Equipment","Salary
  Amount = c(9000, 40000, 8000, 6000, 15000, 20000, 7000, 9000, 12000, 5000),
  Date = c("15/01/2025","20/02/2025","10/03/2025","05/04/2025","15/05/2025",
           "10/06/2025","20/07/2025","30/08/2025","01/10/2025","15/11/2025")
)




# Dynamically setting the time
# 1. Collect all date columns from both dataframes
all_dates <- c(sources$ValidFrom, sources$ValidTo, expenses$Date)
# 2. Convert to Date objects to find the minimum
date_objects <- as.Date(unique(all_dates), format = "%d/%m/%Y")
# 3. Find the earliest date
global_min_date <- min(date_objects, na.rm = TRUE)

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

```r
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)
compatibility_again <- 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$ValidFrom[i], global_min_date)
    s_to   <- date_to_int(sources$ValidTo[i], global_min_date)
    e_date <- date_to_int(expenses$Date[j], global_min_date) # 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)
    time_match_modified <- (e_date < s_to)

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

    if (cat_match && time_match_modified) {
      compatibility_again[i, j] <- 1
    } else {
      compatibility_again[i, j] <- 0
    }
  }
}
```

MODEL CONSTRUCTION

```r
# 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

# Define the Model
model <- MIPModel() %>%

  # --- Variables ---
```

```
  # x[i,j]: Amount source i pays for expense j (Continuous, Non-negative)
  add_variable(x[i, j], i = 1:n_sources, j = 1:n_expenses, type = "continuous", lb = 0) %>%

  # y[j]: Binary indicator if expense j is fully paid (0 or 1)
  add_variable(y[j], j = 1:n_expenses, type = "binary") %>%

  # --- Objective Function ---
  # Maximize Sum(Weights * y)
  set_objective(sum_expr(weights[j] * y[j], j = 1:n_expenses), "max") %>%

  # --- Constraints ---

  # 1. Supply Constraint: Sum of allocations from Source i <= Source Amount
  add_constraint(sum_expr(x[i, j], j = 1:n_expenses) <= sources$Amount[i], i = 1:n_sources) %>%

  # 2. Demand Linking: Sum of allocations to Expense j == Expense Amount * y[j]
  # If y[j]=1, we must pay full amount. If y[j]=0, we pay 0.
  add_constraint(sum_expr(x[i, j], i = 1:n_sources) == expenses$Amount[j] * y[j], j = 1:n_expenses) %>%

  # 3. Compatibility Constraint
  # If compatibility[i, j] == 0, then x[i, j] must be 0
  add_constraint(x[i, j] == 0, i = 1:n_sources, j = 1:n_expenses, compatibility[i, j] == 0)
```

OPTIMIZE

```
result <- solve_model(model, with_ROI(solver = "highs"))
```

RESULT # Partial filling function

```
apply_greedy_fill <- function(result, sources, expenses, compatibility) {

  n_sources <- nrow(sources)
  n_expenses <- nrow(expenses)

  # A. Reconstruct the Optimal Allocation Matrix (Fully Funded Only)
  x_sol_raw <- get_solution(result, x[i, j])
  mat_x <- matrix(0, nrow = n_sources, ncol = n_expenses)
  for (r in 1:nrow(x_sol_raw)) {
    mat_x[x_sol_raw$i[r], x_sol_raw$j[r]] <- x_sol_raw$value[r]
  }

  # B. Calculate Remaining Capacity per Source
  current_source_usage <- rowSums(mat_x)
  source_remaining <- sources$Amount - current_source_usage
  source_remaining[source_remaining < 1e-6] <- 0 # Fix floating point dust

  # C. Identify Unfunded Expenses (Binary y[j] == 0)
  y_sol_raw <- get_solution(result, y[j])
  # Order by j to ensure we respect priority (index 1 is highest priority)
  y_sol_raw <- y_sol_raw[order(y_sol_raw$j), ]
  unfunded_indices <- y_sol_raw$j[y_sol_raw$value < 0.5]

  # D. The Greedy Loop
```

```
  for (j in unfunded_indices) {
    amount_needed <- expenses$Amount[j]

    # Try to find money in compatible sources
    for (i in 1:n_sources) {
      if (amount_needed < 1e-6) break

      # Check compatibility AND available funds
      if (compatibility_again[i, j] == 1 && source_remaining[i] > 1e-6) {
        take_amount <- min(amount_needed, source_remaining[i])

        # Update Matrix & Balances
        mat_x[i, j] <- mat_x[i, j] + take_amount
        source_remaining[i] <- source_remaining[i] - take_amount
        amount_needed <- amount_needed - take_amount
      }
    }
  }

  return(mat_x)
}
```

Table output function

```
print_financial_report <- function(mat_x, sources, expenses) {

  n_sources <- nrow(sources)
  n_expenses <- nrow(expenses)

  cat("\n============================================================\n")
  cat(sprintf("%-60s\n", "                    FINAL SOLUTION REPORT                    "))
  cat("============================================================\n")

  # Calculate Status based on matrix totals
  expense_total_alloc <- colSums(mat_x)
  idx_full <- which(expense_total_alloc >= expenses$Amount - 1e-5)
  idx_partial <- which(expense_total_alloc > 1e-5 & expense_total_alloc < expenses$Amount - 1e-5)
  idx_none <- which(expense_total_alloc <= 1e-5)

  # --- 1. FULLY FUNDED ---
  cat("\n--- Fully Funded Expenses ---\n")
  if(length(idx_full) > 0) {
    for (j in idx_full) {
      cat(sprintf("%s: %s ($%s)\n", expenses$ID[j], expenses$Category[j], format(expenses$Amount[j], bi
    }
  } else { cat("None.\n") }

  # --- 2. PARTIALLY FUNDED ---
  cat("\n--- Partially Funded Expenses ---\n")
  if(length(idx_partial) > 0) {
    for (j in idx_partial) {
```

```
      filled_amt <- expense_total_alloc[j]
      percent <- (filled_amt / expenses$Amount[j]) * 100
      cat(sprintf("%s: %s ($%s / $%s) - %.1f%% Covered\n", expenses$ID[j], expenses$Category[j], format
    }
  } else { cat("None.\n") }

  # --- 3. UNFUNDED ---
  cat("\n--- Unfunded Expenses ---\n")
  if(length(idx_none) > 0) {
    for (j in idx_none) {
      cat(sprintf("%s: %s ($%s) - Due: %s\n", expenses$ID[j], expenses$Category[j], format(expenses$Amou
    }
  } else { cat("None.\n") }

  cat(sprintf("\nSummary: %d Full / %d Partial / %d Missed\n", length(idx_full), length(idx_partial), le

  # --- 4. 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 1:n_expenses) {
      val <- mat_x[i, j]
      if (val > 1e-6) {
        cat(sprintf("%-8s -> %-8s %-15s $%s\n", sources$ID[i], expenses$ID[j], expenses$Category[j], for
      }
    }
  }

  # --- 5. 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")
  total_unused <- 0
  for (i in 1:n_sources) {
    used <- sum(mat_x[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=",", ns
  }
  cat("------------------------------------------------------------------------\n")
  cat(sprintf("TOTAL UNUSED FUNDS: $%s\n", format(total_unused, big.mark=",")))
}
```

Output DataFrame

```
create_financial_dfs <- function(mat_x, sources, expenses) {

  n_sources <- nrow(sources)
```

```r
n_expenses <- nrow(expenses)

# --- 1. Allocations DataFrame ---
# Find all non-zero entries in the matrix
# which(..., arr.ind=TRUE) returns a matrix of [row_index, col_index]
alloc_idx <- which(mat_x > 1e-6, arr.ind = TRUE)

# Construct the dataframe directly from indices
df_allocations <- data.frame(
  SourceID = sources$ID[alloc_idx[, 1]],
  ExpenseID = expenses$ID[alloc_idx[, 2]],
  ExpenseCategory = expenses$Category[alloc_idx[, 2]],
  AllocatedAmount = mat_x[alloc_idx]
)
# Optional: Sort by Source then Expense
df_allocations <- df_allocations[order(df_allocations$SourceID, df_allocations$ExpenseID), ]


# --- 2. Expense Status DataFrame ---
# Calculate how much was allocated to each expense (Column Sums)
expense_filled_amounts <- colSums(mat_x)

df_expenses_status <- expenses
df_expenses_status$FilledAmount <- expense_filled_amounts

# Determine status: Fully Filled if allocated >= requested (minus tiny error)
df_expenses_status$IsFilled <- expense_filled_amounts >= (expenses$Amount - 1e-5)

# Add a readable Status column
df_expenses_status$Status <- ifelse(df_expenses_status$IsFilled, "Full",
                                    ifelse(df_expenses_status$FilledAmount > 1e-6, "Partial", "Unfunded"))


# --- 3. Funds Summary DataFrame ---
# Calculate how much each source used (Row Sums)
source_used_amounts <- rowSums(mat_x)

df_funds_summary <- data.frame(
  SourceID = sources$ID,
  InitialAmount = sources$Amount,
  UsedAmount = source_used_amounts,
  RemainingAmount = sources$Amount - source_used_amounts
)
# Clean up negative zeros
df_funds_summary$RemainingAmount[df_funds_summary$RemainingAmount < 0] <- 0


# Return all 3 as a named list
return(list(
  allocations = df_allocations,
  expenses = df_expenses_status,
  funds = df_funds_summary
))
```

```
}
```

```
if (result$status == "optimal" || result$status == "success") {

  # Partial fill
  final_matrix <- apply_greedy_fill(result, sources, expenses, compatibility)

  # Print report in R
  print_financial_report(final_matrix, sources, expenses)

  # Generate DataFrames
  dfs <- create_financial_dfs(final_matrix, sources, expenses)

  df_allocations <- dfs$allocations
  df_expenses_status <- dfs$expenses
  df_funds_summary <- dfs$funds

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

```
##
## ============================================================
##                     FINAL SOLUTION REPORT
## ============================================================
##
## --- Fully Funded Expenses ---
## E001: Salary ($9,000)
## E003: Salary ($8,000)
## E004: Travel ($6,000)
## E005: Equipment ($15,000)
## E006: Salary ($20,000)
## E007: Travel ($7,000)
## E008: Equipment ($9,000)
## E010: Travel ($5,000)
##
## --- Partially Funded Expenses ---
## E002: Equipment ($18,000 / $40,000) - 45.0% Covered
## E009: Salary ($5,000 / $12,000) - 41.7% Covered
##
## --- Unfunded Expenses ---
## None.
##
## Summary: 8 Full / 2 Partial / 0 Missed
##
## --- Allocation Details ---
## Source   Expense  Exp. Category   Amount Allocated
## ------------------------------------------------------------
## FS001    -> E001     Salary          $4,000.00
## FS001    -> E003     Salary          $8,000.00
## FS002    -> E002     Equipment       $3,000.00
## FS002    -> E005     Equipment       $15,000.00
## FS003    -> E004     Travel          $6,000.00
```

```
## FS003    -> E007     Travel          $4,000.00
## FS004    -> E001     Salary          $5,000.00
## FS004    -> E006     Salary          $20,000.00
## FS004    -> E009     Salary          $5,000.00
## FS005    -> E002     Equipment       $6,000.00
## FS005    -> E008     Equipment       $9,000.00
## FS006    -> E007     Travel          $3,000.00
## FS006    -> E010     Travel          $5,000.00
## FS007    -> E002     Equipment       $9,000.00
##
## --- Remaining Fund Balances ---
## Fund ID  Initial    Used       Remaining  Allowed Categories
## ------------------------------------------------------------
## FS001    12,000     12,000     0          Salary
## FS002    18,000     18,000     0          Equipment
## FS003    10,000     10,000     0          Travel
## FS004    30,000     30,000     0          Salary
## FS005    15,000     15,000     0          Equipment
## FS006    8,000      8,000      0          Travel
## FS007    9,000      9,000      0          Equipment
## ------------------------------------------------------------
## TOTAL UNUSED FUNDS: $0
```

```
# SourceID: ID of the funding source (e.g., FS001).
# ExpenseID: ID of the expense being paid (e.g., E004).
# ExpenseCategory: The category of the expense (e.g., Salary).
# AllocatedAmount: The exact dollar amount transferred.
df_allocations
```

```
##    SourceID ExpenseID ExpenseCategory AllocatedAmount
## 1     FS001      E001          Salary            4000
## 6     FS001      E003          Salary            8000
## 3     FS002      E002       Equipment            3000
## 8     FS002      E005       Equipment           15000
## 7     FS003      E004          Travel            6000
## 10    FS003      E007          Travel            4000
## 2     FS004      E001          Salary            5000
## 9     FS004      E006          Salary           20000
## 13    FS004      E009          Salary            5000
## 4     FS005      E002       Equipment            6000
## 12    FS005      E008       Equipment            9000
## 11    FS006      E007          Travel            3000
## 14    FS006      E010          Travel            5000
## 5     FS007      E002       Equipment            9000
```

```
# All original columns (ID, Category, Amount, Date) plus:
# IsFilled: A Boolean (TRUE/FALSE) indicating if the optimization solver selected this expense.
df_expenses_status
```

```
##      ID  Category Amount       Date FilledAmount IsFilled  Status
## 1  E001    Salary   9000 15/01/2025         9000     TRUE    Full
## 2  E002 Equipment  40000 20/02/2025        18000    FALSE Partial
## 3  E003    Salary   8000 10/03/2025         8000     TRUE    Full
```

```
## 4  E004    Travel   6000 05/04/2025         6000     TRUE    Full
## 5  E005 Equipment  15000 15/05/2025        15000     TRUE    Full
## 6  E006    Salary  20000 10/06/2025        20000     TRUE    Full
## 7  E007    Travel   7000 20/07/2025         7000     TRUE    Full
## 8  E008 Equipment   9000 30/08/2025         9000     TRUE    Full
## 9  E009    Salary  12000 01/10/2025         5000    FALSE Partial
## 10 E010    Travel   5000 15/11/2025         5000     TRUE    Full
```

```
# SourceID: ID of the fund.
# InitialAmount: The starting budget.
# UsedAmount: Total allocated in this solution (sum(x_matrix[i, ])).
# RemainingAmount: What is left over (Initial - Used).
df_funds_summary
```

```
##   SourceID InitialAmount UsedAmount RemainingAmount
## 1    FS001         12000      12000               0
## 2    FS002         18000      18000               0
## 3    FS003         10000      10000               0
## 4    FS004         30000      30000               0
## 5    FS005         15000      15000               0
## 6    FS006          8000       8000               0
## 7    FS007          9000       9000               0
```

```
# knitr::purl(input = "Complete_Algorithm_Alternative.Rmd", output = "Complete_Algorithm_Alternative_V3
```