Tutorial VI - Minimizing Split Orders in E-Commerce

Applied Optimization with Julia

Introduction

We are consulting for FitGear, a rapidly growing e-commerce company specializing in fitness equipment and athletic wear. FitGear is facing a challenge: customer complaints about split orders have increased by 40% in the last quarter. When a customer orders multiple items, they sometimes receive multiple packages from different warehouses, leading to higher shipping costs and frustrated customers.

This tutorial explores how to optimize SKU allocation across warehouses to minimize split orders using mathematical optimization.

---

1. Coappearance Matrix

First, we need to understand which products are frequently purchased together. We'll use a coappearance matrix. It shows how frequently items appear together in customer orders. For each pair of items (i,j), the matrix element Q[i,j] represents the number of orders containing both items i and j. Let's start with a small sample of FitGear's historical order data and determine the coappearance matrix of the transactional data set manually by hand.

t_{mi}	A	B	C	D
1	1	1	1	0
2	1	1	1	0
3	1	1	0	0
4	1	0	0	1
5	1	0	0	1
6	1	0	0	1
7	1	0	0	1
8	0	0	1	1

What is the resulting coappearance matrix from the transactional data set? Please provide it in the following cell as matrix by changing the `0`s to the correct values.

```
Q = [
    0 0 0 0;
```

```
    0 0 0 0;
    0 0 0 0;
    0 0 0 0;
]
```

---

2. Maximizing Coappearances

FitGear currently operates two warehouses - one in Hamburg and one in Berlin. Each warehouse has limited storage capacity:

- Hamburg can store 3 different product types (SKUs)
- Berlin can also store 3 different product types

Based on the coappearance matrix you have determined in the previous assignment, use the model from the lecture to maximise the coappearances manually by hand. What is the resulting objective function value of the assignment? Please provide it in the following cell.

```
objective_value_by_hand =
```

---

3. Counting Split Parcels

Based on the output of the optimisation, we don't know the number of split orders yet. Calculate the number of split orders manually from your previous solution.

```
number_of_split_parcels =
```

> **i** Bonus Question
>
> Is the solution of the QMKP the optimal solution for the underlying split parcel minimisation problem? Try to come up with another solution that has less split or the same number of split parcels.

---

4. Implementing the model

As FitGear continues to grow, the operations team needs a solution that can handle their full product catalog. They now have:

- 12 popular fitness products
- 40 recent customer orders
- Two warehouses with different capacities:
  - Hamburg: 7 SKUs
  - Berlin: 6 SKUs

Thus, we'll now implement a scalable solution using Julia and JuMP that can handle larger datasets.

```
skus = [
    "Running Shoes",
    "Athletic Socks",
    "Water Bottle",
    "Yoga Mat",
    "Resistance Bands",
    "Sports Bag",
    "Protein Powder",
    "Exercise Shorts",
    "Training Shirt",
    "Fitness Tracker",
    "Foam Roller",
    "Weight Gloves"
]

warehouses = ["Hamburg", "Berlin"]

capacity = [7, 6]

T = [
    # First set of transactions (1-20)
    1 1 0 0 0 0 0 0 0 0 0 0;  # shoes + socks
    1 1 0 0 0 1 0 0 0 0 0 0;  # shoes + socks + sports bag
    0 0 0 1 1 0 0 0 0 0 1 0;  # yoga mat + bands + foam roller
    0 0 1 1 0 0 0 0 0 0 0 0;  # water bottle + yoga mat
    0 0 0 0 0 0 1 0 0 0 0 0;  # just protein powder
    0 0 0 0 0 0 1 0 0 0 0 1;  # protein powder + gloves
    0 0 0 0 0 0 0 1 1 0 0 0;  # shorts + shirt combo
    0 0 0 0 1 0 0 1 1 0 0 0;  # workout outfit + bands
    0 0 0 0 0 0 0 0 0 1 0 0;  # just fitness tracker
    1 1 1 0 0 1 0 1 1 0 0 0;  # full running gear set
    0 0 0 1 1 0 0 0 0 0 1 1;  # home gym basics
    0 0 0 0 0 0 1 1 1 0 0 1;  # gym starter pack
    0 0 1 0 1 0 0 0 0 0 0 0;  # water bottle + bands
    1 1 0 0 0 0 0 1 1 0 0 0;  # running outfit complete
    0 0 0 1 1 0 0 0 0 0 1 0;  # yoga equipment set
    0 0 0 0 0 0 1 0 0 0 0 1;  # protein powder + gloves
    0 0 1 0 0 0 1 0 0 0 0 0;  # water bottle + protein
    1 0 0 0 0 1 0 0 0 1 0 0;  # shoes + bag + tracker
    0 0 0 0 1 0 0 1 1 0 0 0;  # workout outfit + bands
    0 0 0 1 0 0 0 0 0 0 1 0;  # yoga mat + foam roller
    1 1 0 0 0 0 0 1 1 0 0 0;  # shoes + socks + workout clothes
    1 1 0 0 0 1 0 0 0 1 0 0;  # shoes + socks + bag + tracker
    0 0 0 1 1 0 0 0 0 0 1 1;  # yoga mat + bands + foam roller + gloves
    0 0 1 1 1 0 0 0 0 0 0 0;  # water bottle + yoga mat + bands
    0 0 1 0 0 0 1 0 0 0 0 1;  # protein powder + water bottle + gloves
    0 0 0 0 1 0 1 0 0 0 0 1;  # protein powder + bands + gloves
    0 0 0 0 0 0 0 1 1 1 0 0;  # shorts + shirt + tracker
    1 0 0 0 1 0 0 1 1 0 0 0;  # shoes + workout outfit + bands
    0 0 1 0 0 0 0 0 0 1 0 0;  # water bottle + tracker
    1 1 1 0 0 1 0 1 1 1 0 0;  # deluxe running gear set
    0 0 0 1 1 0 1 0 0 0 1 1;  # advanced home gym set
    0 0 1 0 0 0 1 1 1 0 0 1;  # gym starter pack with water bottle
    0 0 1 0 1 0 0 0 0 1 0 0;  # water bottle + bands + tracker
```

```
    1 1 0 0 0 1 0 1 1 0 0 0;  # running outfit with bag
    0 0 0 1 1 0 0 0 0 0 1 1;  # yoga equipment set with gloves
    0 0 1 0 0 0 1 0 0 0 1 1;  # protein set with foam roller
    0 0 1 0 0 1 1 0 0 0 0 0;  # water bottle + protein + bag
    1 0 0 0 0 1 0 0 0 1 1 0;  # shoes + bag + tracker + foam roller
    0 0 0 0 1 0 0 1 1 1 0 0;  # workout outfit + bands + tracker
    0 0 0 1 0 0 0 0 0 1 1 0   # yoga mat + tracker + foam roller
]
```

Implement the model in Julia and solve it for the given data set.

> **i** Each SKU can only be allocated to one warehouse!
>
> The retailer has another requirement: each SKU can only be allocated to one warehouse. Make sure to include this requirement in your model.

Use the Juniper solver

If you don't have Juniper, Ipopt and/or HiGHS installed, add the solver via Pkg.add("Juniper") and Pkg.add("Ipopt") and Pkg.add("HiGHS"). Juniper is a solver for nonlinear problems, that can be used in combination with Ipopt and HiGHS to solve mixed-integer quadratic problems. If you want to use SCIP, you can also do this by adding JuMP and SCIP and then change the solver in the model definition to `warehouse_model = Model(SCIP.Optimizer())`. Note, that this does not work automatically in Windows, as you will have to install the SCIP binaries manually. On Mac and Linux, it should work out of the box.

First, we start by defining the model.

```
# Definition of the warehouse model
using JuMP, Ipopt, HiGHS, Juniper
ipopt = optimizer_with_attributes(Ipopt.Optimizer, "print_level" => 0)
highs = optimizer_with_attributes(HiGHS.Optimizer, "output_flag" => false)
warehouse_model = Model(
    optimizer_with_attributes(
        Juniper.Optimizer,
        "nl_solver" => ipopt,
        "mip_solver" => highs,
    ),
)
```

```
A JuMP Model
├ solver: Juniper
├ objective_sense: FEASIBILITY_SENSE
├ num_variables: 0
├ num_constraints: 0
└ Names registered in the model: none
```

Compute the coappearance matrix

Next, compute the coappearance matrix based on the transactional data provided in `T`.

```
# YOUR CODE BELOW
```

```
# Assert whether the coappearance matrix is correct.
@assert Q == Q' "The coappearance matrix is not symmetric. Have you
transposed the transactional data set?"
@assert Q[1,1] == 11 "The coappearance matrix is not correct. Have you
multiplied the transposed transactional data set with itself?"
println("Great! The coappearance matrix is correct.")
```

Define the decision variable

Now, define the decision variable for the SKU allocation. Please name the variable `X`.

```
# YOUR CODE BELOW
```

```
# Assert whether the decision variable is correct.
@assert typeof(X) <: AbstractArray{VariableRef} "The decision variable X
should be an array of JuMP variables"
@assert all(is_binary.(X)) "The decision variable X should be defined as
binary"
@assert size(X) == (length(skus), length(warehouses)) "The decision
variable X should have dimensions [skus × warehouses]"
println("Great! The decision variable is correctly defined as a binary
variable with proper dimensions.")
```

Define the objective function

Then, define the objective function to maximize the coappearance.

```
# YOUR CODE BELOW
```

```
# Assert whether the objective function is correct.
@assert typeof(objective_function(warehouse_model)) <: QuadExpr "The
objective function should be a quadratic expression of JuMP variables"
println("Great! The objective function is correctly defined.")
```

Define the constraints

To ensure that each SKU is allocated to one warehouse, add the first constraint.

```
# YOUR CODE BELOW
```

```
# Assert whether the single allocation constraint is correct.
@assert num_constraints(warehouse_model, AffExpr, MOI.EqualTo{Float64}) ==
length(skus) "The single allocation constraint should have one constraint
for each SKU"
println("Great! The single allocation constraint is correctly defined.")
```

To ensure that the capacity of each warehouse is not exceeded, add the second constraint.

```
# YOUR CODE BELOW
```

```
# Assert whether the capacity constraint is correct.
@assert num_constraints(warehouse_model, AffExpr, MOI.LessThan{Float64}) ==
length(warehouses) "The capacity constraint should have one constraint for
each warehouse"
println("Great! The capacity constraint is correctly defined.")
```

Solve the model

Finally, solve the model with a solve statement.

```
# YOUR CODE BELOW
```

```
# Assert whether the model is solved correctly.
@assert termination_status(warehouse_model) == MOI.OPTIMAL ||
termination_status(warehouse_model) == MOI.LOCALLY_SOLVED "The model should
either be solved with an optimal solution or with a locally optimal
solution"
@assert isapprox(objective_value(warehouse_model), 141) "The objective
value should approximaly be 141. Have you correctly implemented the
objective function?"
println("Great! The model is correctly solved.")
```

Print the results

The following code prints the objective value, the SKU allocation and the warehouse capacities based on your optimal solution.

```
println("Objective value: ", objective_value(warehouse_model))
println()
println("SKU Allocation:")
for i in skus
    for k in warehouses
        if value(X[i,k]) > 0.1
            println("SKU: ", i, " Allocation: ", k)
        end
    end
end
```

```
println()
println("Warehouse Capacities:")
for k in 1:length(warehouses)
    println("Warehouse: ", warehouses[k], " Capacity: ", capacity[k], "
Used: ", sum(value.(X[i,warehouses[k]]) for i in skus))
end
```

Count the number of split orders

Based on the output of the optimisation, we still don't know the number of split orders. The following code calculates the number of split orders from the SKU allocation for the optimal solution.

```
# Binary matrix indicating whether a SKU is allocated to a warehouse.
X_values = [value(X[i,j]) > 0 ? true : false for i in skus, j in
warehouses]

# Function to count the number of split and regular parcels.
function count_split_orders(X_input, transactional_data)
    # Initialize the counters for split and regular parcels.
    split_parcels = 0
    regular_parcels = 0

    # Iterate over each transaction in the transactional data set.
    for t in 1:size(transactional_data,1)

        # Check if the first warehouse can fulfill the transaction.
        if all(X_input[:,1] .>= transactional_data[t,:])
            regular_parcels += 1

        # Check if the second warehouse can fulfill the transaction.
        elseif all(X_input[:,2] .>= transactional_data[t,:])
            regular_parcels += 1

        # If neither warehouse can fulfill the transaction, it is a split.
        else
            split_parcels += 1
            regular_parcels += 1
        end
    end
    return split_parcels, regular_parcels
end

# Count the number of split and regular parcels for the optimal solution.
split_parcels, regular_parcels = count_split_orders(X_values, T)

# Count the number of split and regular parcels for 100 random solutions.
split_parcels_random = []
regular_parcels_random = []
for trial in 1:100
    X_random = [rand(Bool) for i in 1:length(skus), j in
1:length(warehouses)]
```

```
    split_parcels_random_trial, regular_parcels_random_trial =
count_split_orders(X_random, T)
    push!(split_parcels_random, split_parcels_random_trial)
    push!(regular_parcels_random, regular_parcels_random_trial)
end
split_parcels_random = sum(split_parcels_random)/100
regular_parcels_random = sum(regular_parcels_random)/100

# Print the number of split and regular parcels.
println("Number of split orders (optimal): ", split_parcels)
println("Number of regular orders (optimal): ", regular_parcels)
println()

# Print the number of split and regular parcels for the random solution.
println("Number of split orders (random): ", split_parcels_random)
println("Number of regular orders (random): ", regular_parcels_random)
```

---

5. Analyzing the Results

FitGear's operations team needs to understand the business implications of this optimization:

- Which product categories tend to cluster together in the same warehouse? For example, do workout clothes tend to be stored together?
- How many fewer split shipments would FitGear have compared to their current random allocation?
- How might this new allocation affect warehouse operations and picking efficiency?

Your answer goes here, thinking from the business perspective. A few sentences are fully sufficient!

```
# YOUR ANSWER BELOW
#=



=#
```

Solutions

You will likely find solutions to most exercises online. However, I strongly encourage you to work on these exercises independently without searching explicitly for the exact answers to the exercises. Understanding someone else's solution is very different from developing your own. Use the lecture notes and try to solve the exercises on your own. This approach will significantly enhance your learning and problem-solving skills.

Remember, the goal is not just to complete the exercises, but to understand the concepts and improve your programming abilities. If you encounter difficulties, review

the lecture materials, experiment with different approaches, and don't hesitate to ask for clarification during class discussions.

Bibliography