Decision Management Community Challenge Oct-2025

"Decision with two objectives"

https://dmcommunity.org/challenge/challenge-oct-2025/

Challenge Oct-2025

Decision with two objectives

Solutions

This challenge is offered by Dr. Meinolf Sellmann. A freelance webpage developer received a task. A client has a budget of $10,000 and wants a webpage developed with as many features as possible, but also maximizing the total value of these features:

Feature #	Costs	Value
1	5,000	7
2	4,000	6
3	3,000	5
4	2,000	4
5	1,000	3
6	1,000	2
7	1,000	1
8	1,000	1
9	1,000	1
10	1,000	1

The client leaves the decision of which features to the developer. She wants to delight the client and asks you which features would be best to select. We have the budget constraint and two objectives. Can you devise a rational way to trade them against each other?

Copilot has already produced for me a nice but procedural solution for this challenge. I decided to point "him" that "his" solution ignored the required balancing two objectives: value and number of features. Copilot nicely offered the following approach with a balancing objective (score):

Copilot

Great question! This is essentially a **multi-objective optimization problem** with a budget constraint. Let's break it down and devise a rational approach:

Step 1: Define the Problem

- **Budget constraint:** $10,000.
- **Objectives:**
    1. **Maximize the number of features** (quantity).
    2. **Maximize the total value of features** (quality).

Each feature will have:

- **Cost** (how much it consumes from the budget).
- **Value** (how much it contributes to client satisfaction or utility).

Step 2: Model the Trade-Off

This is similar to a **knapsack problem** with two objectives:

- **Objective 1:** Maximize sum of values.
- **Objective 2:** Maximize count of features.

We can combine these objectives into a **single score** using a weighted approach:

$$\text{Score(feature)} = \alpha \cdot \text{Value} + \beta \cdot \frac{1}{\text{Cost}}$$

Where:

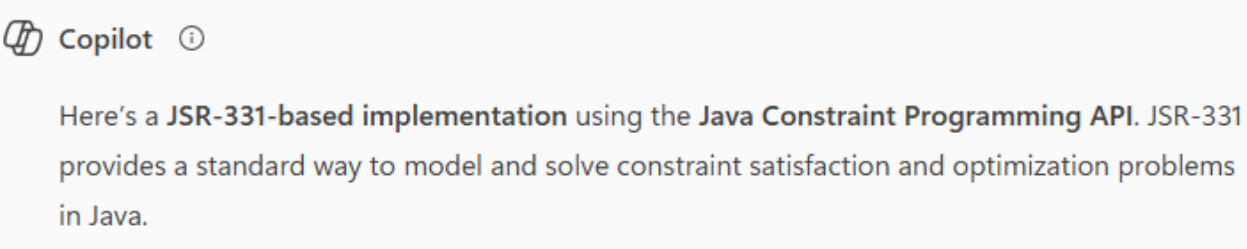
- **α** emphasizes value.
- **β** emphasizes quantity (lower cost means more features fit).

Alternatively, normalize both objectives and use a **weighted sum**:

$$\text{Total Score} = w_1 \cdot \frac{\text{Total Value}}{\text{Max Value}} + w_2 \cdot \frac{\text{Number of Features}}{\text{Max Features}}$$

Weights w_1, w_2 reflect how much the developer prioritizes delight (value) vs. quantity.

Then I asked Copilot to generate for me an implementation using the standard JSR-331. Copilot immediately answered:



Then it generated for me a nicely looking Java code with JSR-331 API. When I looked closely, I noticed that the generated code uses non-existing constructs. It is a **common deficiency with most LLM-based tools**: **what they present as a perfect solution is frequently a mix of useful components and "garbage".**

I added this code to my JSR-331 samples and naturally received many syntax errors. Being proficient in this standard, it was not difficult for me to quickly fix all of them. At the end of this post, I included the final working Java code with JSR-331 API for this challenge. Being oriented to Java programmers, it still is a nice, compact, and **declarative** solution.

Initially, this code used *weight for value = 8* and *weight for quantity = 2*. It produced the following results:

```
JSR-331 "Constraint Programming API" Release 2.3.3
JSR-331 Implementation based on Constrainer 5.4.1
Optimal solution is found. Objective: 152
Selected Features:
Feature 2 (Cost: 4000, Value: 6)
Feature 3 (Cost: 3000, Value: 5)
Feature 4 (Cost: 2000, Value: 4)
Feature 5 (Cost: 1000, Value: 3)
Total Cost: 10000
Total Value: 18
```

When I switched to *weight for value = 5* and *weight for quantity = 5*, I received the following results:

```
Optimal solution is found. Objective: 120
Selected Features:
Feature 3 (Cost: 3000, Value: 5)
Feature 4 (Cost: 2000, Value: 4)
Feature 5 (Cost: 1000, Value: 3)
Feature 6 (Cost: 1000, Value: 2)
Feature 8 (Cost: 1000, Value: 1)
Feature 9 (Cost: 1000, Value: 1)
Feature 10 (Cost: 1000, Value: 1)
Total Cost: 10000
Total Value: 17
```

```java
import javax.constraints.*;

public class FeatureSelection {
    public static void main(String[] args) {
        Problem p = ProblemFactory.newProblem("Feature Selection");

        // Data
        int[] costs = {5000, 4000, 3000, 2000, 1000, 1000, 1000, 1000, 1000, 1000};
        int[] values = {7, 6, 5, 4, 3, 2, 1, 1, 1, 1};
        int budget = 10000;

        int wValue = 8;   // Weight for value
        int wCount = 2;   // Weight for quantity
        int n = costs.length;

        // Decision variables: x[i] = 1 if feature i is selected
        Var[] x = p.variableArray("x", 0, 1, n);

        p.post(p.scalProd(costs, x), "<=", budget);

        // Objective: weighted sum
        Var valueObjective = p.scalProd(values, x);
        Var quantityObjective = p.sum(x);
        Var balancedObjective = valueObjective.multiply(wValue).plus(quantityObjective.multiply(wCount));

        // Solve
        Solver solver = p.getSolver();
        Solution solution = solver.findOptimalSolution(Objective.MAXIMIZE, balancedObjective);

        if (solution != null) {
            System.out.println("Selected Features:");
            int totalCost = 0;
            int totalValue = 0;
            for (int i = 0; i < n; i++) {
                if (solution.getValue("x-" + i) == 1) {
                    System.out.println("Feature " + (i + 1) +
                                       " (Cost: " + costs[i] + ", Value: " + values[i] + ")");
                    totalCost += costs[i];
                    totalValue += values[i];
                }
            }
            System.out.println("Total Cost: " + totalCost);
            System.out.println("Total Value: " + totalValue);
        } else {
            System.out.println("No solution found.");
        }
    }
}
```