Production Planning Optimization Problem

Problem Set 2 – Mixed Integer Linear Programming

Complete Problem Statement

A company manufactures three discrete products, labeled $j = 1, 2, 3$. Each product:

- Can be produced in quantities that must not exceed 60 units
- Incurs a fixed cost when production begins
- Has declining marginal profit rates (first units earn more profit than later units)
- Requires consumption of four different resources

The company must determine an optimal production plan that maximizes total profit while respecting all resource constraints and business rules.

Given Data

Resource availability and per-unit consumption:

Resource i	Product 1	Product 2	Product 3	Availability b_i
1	12	15	10	1500
2	15	14	12	1900
3	11	13	9	1800
4	13	12	15	1200

Fixed costs: $f_1 = 40$ EUR, $f_2 = 50$ EUR, $f_3 = 45$ EUR

Production capacity: Maximum 60 units per product

Marginal profit structure (piecewise linear):

Product 1: 4 EUR/unit (first 10 units), 3 EUR/unit (units 11-60)

Product 2: 6 EUR/unit (first 8 units), 4 EUR/unit (units 9-60)

Product 3: 5 EUR/unit (first 10 units), 2.5 EUR/unit (units 11-20), 1 EUR/unit (units 21-60)

Logical requirement: If Product 3 is produced, then Product 1 must also be produced.

Part (a): MILP Formulation

Sets

- $J = \{1, 2, 3\}$: Set of products
- $I = \{1, 2, 3, 4\}$: Set of resources
- K_j: Set of profit segments for product j
  - $K_1 = \{1, 2\}$ (Product 1 has 2 segments)
  - $K_2 = \{1, 2\}$ (Product 2 has 2 segments)
  - $K_3 = \{1, 2, 3\}$ (Product 3 has 3 segments)

Parameters

- f_j: Fixed cost for producing product j (EUR)
  - $f_1 = 40$, $f_2 = 50$, $f_3 = 45$
- b_i: Availability of resource i (units)
  - $b_1 = 1500$, $b_2 = 1900$, $b_3 = 1800$, $b_4 = 1200$
- a_{ij}: Consumption of resource i per unit of product j
- p_{jk}: Profit per unit for product j in segment k (EUR/unit)
- c_{jk}: Maximum capacity of product j in segment k (units)

Detailed segment parameters:

Product j	Segment k	Profit p_{jk}	Capacity c_{jk}
1	1	4	10
1	2	3	50
2	1	6	8
2	2	4	52
3	1	5	10
3	2	2.5	10
3	3	1	40

Decision Variables

- $x_j \in \mathbb{Z}_+$: Total units to produce of product j, $\forall j \in J$
- $y_j \in \{0, 1\}$: Binary decision to produce product j (1 if yes, 0 if no), $\forall j \in J$

- $v_{jk} \in \mathbb{Z}_+$: Units of product j produced and assigned to segment k, $\forall j \in J, k \in K_j$
- $w_{jk} \in \{0,1\}$: Binary indicator if segment k of product j is active (1 if yes, 0 if no), $\forall j \in J, k \in K_j, k \geq 2$

Total: 17 variables (10 integer + 7 binary)

Objective Function: Maximize Net Profit

$$\max Z = \sum_{j\in J} \sum_{k\in K_j} (p_{jk} \cdot v_{jk}) - \sum_{j\in J} (f_j \cdot y_j)$$

Expanded form:

$$\max Z = 4v_{11} + 3v_{12} + 6v_{21} + 4v_{22} + 5v_{31} + 2.5v_{32} + v_{33} - 40y_1 - 50y_2 - 45y_3$$

Business Logic: The objective is to maximize net profit. This is calculated as the total gross profit (the sum of segment quantities v_{jk} multiplied by their specific profit rates p_{jk}) minus the total fixed costs (the sum of fixed costs f_j for those products that are produced, where $y_j = 1$).

Subject to:

1. Segment Aggregation Constraints

$$x_j = \sum_{k\in K_j} v_{jk} \quad \forall j \in J$$

Expanded form:

$$x_1 = v_{11} + v_{12} \tag{1}$$
$$x_2 = v_{21} + v_{22} \tag{2}$$
$$x_3 = v_{31} + v_{32} + v_{33} \tag{3}$$

Business Logic: This constraint ensures that the total production of a product x_j is exactly equal to the sum of the quantities produced in each of its profit segments v_{jk}. This creates the link between segment-level decisions and total production.

2. Resource Capacity Constraints

$$\sum_{j\in J} a_{ij} x_j \leq b_i \quad \forall i \in I$$

Expanded form:

$$
\begin{aligned}
12x_1 + 15x_2 + 10x_3 &\leq 1500 \quad \text{(Resource 1)} \qquad (4)\\
15x_1 + 14x_2 + 12x_3 &\leq 1900 \quad \text{(Resource 2)} \qquad (5)\\
11x_1 + 13x_2 + 9x_3 &\leq 1800 \quad \text{(Resource 3)} \qquad (6)\\
13x_1 + 12x_2 + 15x_3 &\leq 1200 \quad \text{(Resource 4)} \qquad (7)
\end{aligned}
$$

Business Logic: Prevents exceeding resource capacity. The total consumption of each resource i (calculated from total production x_j) must be less than or equal to availability b_i. These constraints model physical or operational limitations.

3. Fixed Cost Logic (Big-M Method)

$$x_j \leq 60 \cdot y_j \quad \forall j \in J$$

Expanded form:

$$
\begin{aligned}
x_1 &\leq 60y_1 \qquad (8)\\
x_2 &\leq 60y_2 \qquad (9)\\
x_3 &\leq 60y_3 \qquad (10)
\end{aligned}
$$

Business Logic: This equation connects the production decision (y_j) with the quantity produced (x_j). If we don't produce ($y_j = 0$), then x_j must be 0. If we do produce ($y_j = 1$), then x_j can take any value up to its maximum of 60. This ensures fixed costs are only incurred when production actually occurs.

4. Production Logic Constraint

$$y_3 \leq y_1$$

Business Logic: Implements the business rule "if Product 3 is produced ($y_3 = 1$), then Product 1 must also be produced ($y_1 = 1$)." This could represent a technical dependency, supply chain requirement, or strategic decision.

5. Segment Capacity Constraints

$$v_{jk} \leq c_{jk} \quad \forall j \in J, k \in K_j$$

Business Logic: Each segment has a maximum capacity that cannot be exceeded. This reflects the piecewise nature of the profit structure where each price tier has limited availability.

6. Production-Segment Dependency

$$v_{jk} \leq c_{jk} \cdot y_j \quad \forall j \in J, k \in K_j$$

Business Logic: Segments can only be used if the product is being produced. If $y_j = 0$, then all segment quantities v_{jk} must be zero.

7. Sequential Segment Filling Logic

Previous segment must be filled:

$$v_{j,k-1} \geq c_{j,k-1} \cdot w_{jk} \quad \forall j \in J, k \in K_j, k \geq 2$$

Current segment limited by activation:

$$v_{jk} \leq c_{jk} \cdot w_{jk} \quad \forall j \in J, k \in K_j, k \geq 2$$

Activation requires production:

$$w_{jk} \leq y_j \quad \forall j \in J, k \in K_j, k \geq 2$$

Expanded form for Product 1:

$$v_{11} \geq 10 \cdot w_{12} \tag{11}$$
$$v_{12} \leq 50 \cdot w_{12} \tag{12}$$
$$w_{12} \leq y_1 \tag{13}$$

Expanded form for Product 2:

$$v_{21} \geq 8 \cdot w_{22} \tag{14}$$
$$v_{22} \leq 52 \cdot w_{22} \tag{15}$$
$$w_{22} \leq y_2 \tag{16}$$

Expanded form for Product 3:

$$v_{31} \geq 10 \cdot w_{32} \tag{17}$$
$$v_{32} \leq 10 \cdot w_{32} \tag{18}$$
$$w_{32} \leq y_3 \tag{19}$$
$$v_{32} \geq 10 \cdot w_{33} \tag{20}$$
$$v_{33} \leq 40 \cdot w_{33} \tag{21}$$
$$w_{33} \leq y_3 \tag{22}$$

Business Logic: This set of constraints forces profit segments to be filled in order (highest profit first). The first equation ensures that segment k cannot be activated ($w_{jk} = 1$) until the previous segment ($k - 1$) is completely full. The second equation ensures that no production can occur in a segment ($v_{jk} > 0$) unless it is active ($w_{jk} = 1$). The third equation ensures segments can only be active if the product is being produced. Together, these constraints prevent "cherry-picking" lower-profit segments while leaving higher-profit segments unfilled.

Complete Model Summary

Model Type: Mixed Integer Linear Program (MILP)

Size:

- 17 decision variables (10 integer + 7 binary)
- 37 constraints

Key Modeling Techniques:

- Fixed charge modeling using binary variables and Big-M constraints
- Piecewise linear profit functions with sequential segment filling
- Logical constraints for production dependencies

Part (b): Gurobi Implementation and Solution Analysis

Python Code Implementation

The following code implements the mathematical model in Gurobi-Python:

—

```
from gurobipy import *

# --- 1. Define Sets and Indices ---
n = 3  # Number of products
m = 4  # Number of resources
products = range(1, n + 1)
resources = range(1, m + 1)

# --- 2. Problem Parameters ---

# F[j]: Fixed cost for producing product j
F = {1: 40, 2: 50, 3: 45}

# b[i]: Availability of resource i
b = {1: 1500, 2: 1900, 3: 1800, 4: 1200}

# a[i][j]: Consumption of resource i per unit of product j
a = {
    1: {1: 12, 2: 15, 3: 10},
    2: {1: 15, 2: 14, 3: 12},
    3: {1: 11, 2: 13, 3: 9},
    4: {1: 13, 2: 12, 3: 15}
}

# Xmax: Maximum production limit per product (used for Big-M)
Xmax = 60

# segments[j]: List of (profit, capacity) tuples for each profit segment k of product j
segments = {
    1: [(4, 10), (3, 50)],
    2: [(6, 8), (4, 52)],
    3: [(5, 10), (2.5, 10), (1, 40)]
}

# --- 3. Model Creation ---
model = Model('ProductionPlanning')

# --- 4. Variable Definitions ---
# y[j]: BINARY variable (1 if product j is produced, 0 otherwise)
y = model.addVars(products, vtype=GRB.BINARY, name="y")

# x[j]: INTEGER variable (total quantity of product j to produce)
x = model.addVars(products, vtype=GRB.INTEGER, lb=0, name="x")

# v[j, k]: INTEGER variable (quantity produced of product j in segment k)
v = model.addVars([(j, k) for j in products for k in range(len(segments[j]))],
                  vtype=GRB.INTEGER, lb=0, name="v")

# w[j, k]: BINARY variable (1 if segment k (for k>0) of product j is active)
# Note: w[j,1] activates the *second* segment. The first segment (k=0) is activated by y[j].
w = model.addVars([(j, k) for j in products for k in range(1, len(segments[j]))],
                  vtype=GRB.BINARY, name="w")

# --- 5. Constraint Definitions ---

# (R1) Production aggregation: Total production x[j] is the sum of its segment productions v[j,
model.addConstrs((x[j] == quicksum(v[j, k] for k in range(len(segments[j]))))
                 for j in products, name="R1_Aggregation")

# (R2) Resource capacity: Total consumption of each resource must not exceed its availability
model.addConstrs((quicksum(a[i][j] * x[j] for j in products) <= b[i])
                 for i in resources, name="R2_Resources")

# (R3) Fixed cost logic (Big-M): If y[j]=0, then x[j]=0. If y[j]=1, x[j] <= 60.
model.addConstrs((x[j] <= Xmax * y[j]) for j in products, name="R3_BigM_FixedCost")

# (R4) Business logic: If Product 3 is produced (y[3]=1), then Product 1 must be produced (y[1]
model.addConstr(y[3] <= y[1], name="R4_Logic_P3_P1")

# (R5) Piecewise segment logic
for j in products:
    num_segments = len(segments[j])
    for k in range(num_segments):
        _, seg_cap = segments[j][k]

        # (R5a) Production in a segment v[j,k] cannot exceed that segment's capacity
        model.addConstr(v[j, k] <= seg_cap, name=f"R5a_SegmentCapacity_{j}_{k}")

        # (R5b) Production in a segment v[j,k] can only be > 0 if the product is produced (y[j]
        model.addConstr(v[j, k] <= seg_cap * y[j], name=f"R5b_SegmentActivation_{j}_{k}")

    # Sequential filling logic (only applies to segments k=1 and up)
    for k in range(1, num_segments):
        prev_cap = segments[j][k - 1][1]  # Capacity of the PREVIOUS segment (k-1)
        curr_cap = segments[j][k][1]      # Capacity of the CURRENT segment (k)
        # (R5c) To activate segment k (w[j,k]=1), the previous segment (k-1) must be full
        model.addConstr(v[j, k - 1] >= prev_cap * w[j, k], name=f"R5c_SequentialFill_{j}_{k}")

        # (R5d) Production in segment k (v[j,k]) can only be > 0 if segment k is active (w[j,k]
        model.addConstr(v[j, k] <= curr_cap * w[j, k], name=f"R5d_SequentialActivation_{j}_{k}'

        # (R5e) (Redundant but clear) Segment k can only be active if the product is produced
        model.addConstr(w[j, k] <= y[j], name=f"R5e_Logic_w_y_{j}_{k}")

# --- 6. Objective Function Definition ---

# Gross profit = Sum of (profit_per_unit * quantity_in_segment) for all segments
profit = quicksum(segments[j][k][0] * v[j, k]
                  for j in products for k in range(len(segments[j])))
# Total fixed costs = Sum of fixed costs for all activated products
fixed_costs = quicksum(F[j] * y[j] for j in products)

# Set Objective: Maximize Net Profit (Gross Profit - Fixed Costs)
model.setObjective(profit - fixed_costs, GRB.MAXIMIZE)

# --- 7. Solver Parameters (Optional) ---
# Disabling these forces Gurobi to solve the "pure" model, which is useful for academic analysi
model.setParam(GRB.Param.Presolve, 0)
model.setParam(GRB.Param.Heuristics, 0)
model.setParam(GRB.Param.Cuts, 0)

# --- 8. Optimize the Model ---
model.optimize()

# --- 9. Print the Solution Report ---
print("\n" + "=" * 65)
print(" OPTIMAL SOLUTION SUMMARY ")
print("=" * 65 + "\n")

if model.status == GRB.OPTIMAL:
    total_profit = 0
    total_fixed = 0

    print(f"Optimal net profit: {model.ObjVal:.2f} \n")

    # Product-level summary (decision and total quantity)
    print("Product-level summary:")
    for j in products:
        print(f"   Product {j}:")
        print(f"     y_{j} = {int(round(y[j].X))}")  # Production decision (0 or 1)
        print(f"     x_{j} = {x[j].X:.0f} units")  # Total quantity
        if y[j].X > 0.5:
            print(f"     Fixed cost paid: {F[j]} ")
            total_fixed += F[j]
        else:
            print(f"     Fixed cost: not incurred (y_{j}=0)")
        print("")

    # Segment-level summary (breakdown of production)
    print("Segment-level production:")
    for j in products:
        for k in range(len(segments[j])):
            qty = v[j, k].X
            if qty > 0.5:  # Only show segments that were used
                price = segments[j][k][0]

                print(f"     v_{j}{k+1} = {qty:.0f} units")  # v_j1, v_j2, ...
                print(f"         at {price} /unit")

                total_profit += price * qty
    print("")

    # Resource utilization summary
    print("Resource utilization:")
    for i in resources:
        used = sum(a[i][j] * x[j].X for j in products)
        print(f"     Resource {i}: used {used:.1f} / available {b[i]}")

    # Final financial summary
    print("\nProfit breakdown:")
    print(f"     Gross profit: {total_profit:.2f} ")
    print(f"     Fixed costs:  {total_fixed:.2f} ")
    print(f"     Net profit:   {total_profit - total_fixed:.2f} ")

    print(f"\nNode count: {model.NodeCount}")  # B&B nodes explored
else:
    print(f"No optimal solution found (status {model.status})")
```

Gurobi Output

```
Set parameter Presolve to value 0
Set parameter Heuristics to value 0
Set parameter Cuts to value 0
Gurobi Optimizer version 12.0.3 build v12.0.3rc0 (mac64[x86] - Darwin 24.6.0 24G231)

CPU model: Intel(R) Core(TM) i5-1030NG7 CPU @ 1.10GHz
Thread count: 4 physical cores, 8 logical processors, using up to 8 threads

Non-default parameters:
Heuristics  0
```

```
Cuts  0
Presolve  0

Optimize a model with 37 rows, 17 columns and 75 nonzeros
Model fingerprint: 0x8028db11
Variable types: 0 continuous, 17 integer (7 binary)
Coefficient statistics:
  Matrix range     [1e+00, 6e+01]
  Objective range  [1e+00, 5e+01]
  Bounds range     [1e+00, 1e+00]
  RHS range        [8e+00, 2e+03]
Variable types: 0 continuous, 17 integer (7 binary)

Root relaxation: objective 2.983077e+02, 8 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  298.30769    0    5          -  298.30769      -     -    0s
     0     0  286.76923    0    5          -  286.76923      -     -    0s
     0     2  286.76923    0    5          -  286.76923      -     -    0s
*    3     6               2     239.0000000  286.71389  20.0%   3.0    0s
*   12    13               5     283.0000000  285.33333  0.82%   1.8    0s
*   16    13               4     284.0000000  285.33333  0.47%   1.6    0s

Explored 27 nodes (47 simplex iterations) in 0.03 seconds (0.00 work units)
Thread count was 8 (of 8 available processors)

Solution count 3: 284 283 239

Optimal solution found (tolerance 1.00e-04)
Best objective 2.840000000000e+02, best bound 2.840000000000e+02, gap 0.0000%

=================================================================
 OPTIMAL SOLUTION SUMMARY
=================================================================

Optimal net profit: 284.00 EUR

Product-level summary:
  Product 1:
     y_1 = 1
     x_1 = 36 units
     Fixed cost paid: 40 EUR

  Product 2:
     y_2 = 1
     x_2 = 60 units
     Fixed cost paid: 50 EUR

  Product 3:
     y_3 = 0
     x_3 = 0 units
     Fixed cost: not incurred (y_3=0)

Segment-level production:
     v_11 = 10 units at 4 EUR/unit
     v_12 = 26 units at 3 EUR/unit
     v_21 = 8 units at 6 EUR/unit
     v_22 = 52 units at 4 EUR/unit

Resource utilization:
     Resource 1: used 1332.0 / available 1500
     Resource 2: used 1380.0 / available 1900
     Resource 3: used 1176.0 / available 1800
     Resource 4: used 1188.0 / available 1200

Profit breakdown:
     Gross profit: 374.00 EUR
     Fixed costs:  90.00 EUR
     Net profit:   284.00 EUR

Node count: 27
```

—

Solution Analysis and Gurobi Statistics

Branch-and-Bound Tree Exploration

The Gurobi solver explored the solution space using the Branch-and-Bound algorithm. Let us analyze the search process in detail:

Root node relaxation:

At the root node (node 0), Gurobi solves the LP relaxation by allowing all integer and binary variables to take fractional values. This provides an upper bound on the optimal integer solution.

- **Initial LP bound:** 298.31 EUR
- **After additional analysis:** 286.77 EUR
- **Iterations at root:** 8 simplex iterations

The drop from 298.31 to 286.77 indicates that even without presolve, heuristics, or automatic cuts, the solver improved the bound through basic constraint propagation and dual variable analysis.

This 11.54 EUR reduction (3.87% improvement) shows that the continuous relaxation initially overestimates profit significantly.

Branching strategy:

After the root node, Gurobi created two child nodes (depth 1) by branching on a fractional variable. The algorithm explored 27 nodes total before proving optimality.

Finding Feasible Solutions

The solver found three integer feasible solutions during the search:

Node	Solution Value	Best Bound	Gap	Progress
3	239.00 EUR	286.71 EUR	20.0%	First feasible
12	283.00 EUR	285.33 EUR	0.82%	Major improvement
16	284.00 EUR	285.33 EUR	0.47%	Near-optimal
27	284.00 EUR	284.00 EUR	0.0%	Proven optimal

Solution progression analysis:

1. **Node 3 (239 EUR):** The first feasible solution appears at depth 2 in the branch-and-bound tree. This solution has a 20% optimality gap, indicating significant room for improvement. This initial solution likely represents a conservative production plan, possibly producing only one product to avoid multiple fixed costs.
2. **Node 12 (283 EUR):** A dramatic improvement of 44 EUR (18.4% increase) occurs at node 12, depth 5. The gap narrows to just 0.82%. This suggests the solver has identified the key structure of the optimal solution – likely the correct product mix (Products 1 and 2) but perhaps not yet the exact quantities.
3. **Node 16 (284 EUR):** A small refinement of 1 EUR improves the solution to 284 EUR with only a 0.47% gap. At this point, we are essentially at the optimal solution, but the solver must continue to verify this by exploring remaining branches.
4. **Node 27 (284 EUR, proven optimal):** After exploring 11 more nodes (nodes 17-27), Gurobi proves that 284 EUR is indeed optimal. The gap closes completely when the best bound matches the incumbent solution.

Computational Efficiency Metrics

Problem size and structure:

- **Constraints:** 37 rows
- **Variables:** 17 columns (0 continuous, 17 integer, of which 7 are binary)

- **Non-zeros:** 75 (matrix density: 11.9%)
- **Model fingerprint:** 0x8028db11 (unique identifier for reproducibility)

Coefficient ranges:

The coefficient statistics reveal the numerical characteristics of the model:

- **Matrix range:** $[1, 60]$ – well-conditioned (ratio 60:1)
- **Objective range:** $[1, 50]$ – moderate variation in profit coefficients
- **Bounds range:** $[1, 1]$ – uniform (binary variables)
- **RHS range:** $[8, 2000]$ – resource constraints span different scales

The relatively narrow ranges indicate the problem is numerically stable. No extreme coefficient values exist that could cause numerical difficulties. A ratio of 250:1 in the RHS is moderate and does not require scaling.

Solution time performance:

- **Total runtime:** 0.03 seconds
- **Work units:** 0.00 (effectively instant on modern hardware)
- **Thread count:** 8 threads used (parallel processing)
- **Nodes explored:** 27
- **Simplex iterations:** 47 total (average 1.74 iterations per node)

The extremely fast solution time (30 milliseconds) reflects both the small problem size and the efficiency of modern MILP solvers. The low iteration count per node (1.74) suggests most nodes were pruned quickly through bound analysis rather than requiring extensive LP subproblem solving.

Why This Problem Solved Quickly

Several factors contributed to the rapid solution:

1. **Tight LP relaxation:** The gap between the initial LP bound (286.77) and optimal integer solution (284.00) is only 2.77 EUR (0.97%). This indicates the piecewise constraints create a formulation where the LP relaxation closely approximates the integer solution. Tight relaxations dramatically reduce the branch-and-bound tree size.
2. **Small variable count:** With only 17 integer variables (7 binary), the solution space is manageable. The maximum theoretical nodes would be $2^7 = 128$ for the binary variables, but effective branching strategies reduced actual exploration to 27 nodes (78.9% reduction).

3. **Strong constraints:** The resource constraints are binding (Resource 4 uses $1188/1200 = 99\%$ capacity), which limits the feasible region and helps prune branches early. Many potential solutions violate resource limits and can be eliminated without full exploration.
4. **Dominated alternatives:** Product 3 is clearly inferior to Products 1 and 2 in terms of profit per unit of resource consumed. This structural property allows the solver to quickly eliminate branches where Product 3 is produced.

Impact of Disabled Features

The parameters `Presolve=0`, `Heuristics=0`, and `Cuts=0` were disabled to observe the pure branch-and-bound algorithm. Let us analyze what each would have contributed:

Presolve: Would have reduced problem size by:

- Eliminating redundant constraints
- Tightening variable bounds
- Detecting implied integer variables
- Fixing dominated variables

Heuristics: Would have attempted to find good feasible solutions quickly at the root node, potentially finding the 284 EUR solution immediately without exploring the 239 EUR solution first.

Cuts: Would have added valid inequalities to tighten the LP relaxation, possibly improving the initial bound beyond 286.77 EUR and reducing the number of nodes explored.

Despite disabling these features, the problem solved in 27 nodes and 0.03 seconds, demonstrating that the formulation itself is strong. With default settings, this problem would likely solve in under 10 nodes.

Optimal Solution Interpretation

Production decisions:

Product	Produce?	Quantity	Fixed Cost	Segment Details
1	Yes ($y_1 = 1$)	36 units	40 EUR	10 at 4 EUR, 26 at 3 EUR
2	Yes ($y_2 = 1$)	60 units	50 EUR	8 at 6 EUR, 52 at 4 EUR
3	No ($y_3 = 0$)	0 units	0 EUR	Not produced

Why Product 3 is not produced:

Product 3 has the lowest profit rates (5, 2.5, 1 EUR per unit) and high resource consumption (10, 12, 9, 15 units per product). The opportunity cost of resources is too high – those resources generate more profit when allocated to Products 1 and 2.

Additionally, the logical constraint $y_3 \leq y_1$ is non-binding in the optimal solution. Even if this constraint were removed, Product 3 would still not be produced due to economic inferiority.

Resource utilization:

Resource	Used	Available	Utilization
1	1332	1500	88.8%
2	1380	1900	72.6%
3	1176	1800	65.3%
4	1188	1200	99.0%

Bottleneck analysis:

Resource 4 is the binding constraint at 99.0% utilization (only 12 units of slack). This is the bottleneck that limits production. If we had one more unit of Resource 4, we could potentially increase profit.

The shadow price of Resource 4 (from dual analysis) would tell us exactly how much additional profit we could gain per unit of this resource. Resources 1, 2, and 3 have slack capacity and therefore zero shadow prices.

Financial summary:

$$\begin{aligned}
\text{Product 1 revenue:}\quad & 10 \times 4 + 26 \times 3 = 118 \text{ EUR} \\
\text{Product 2 revenue:}\quad & 8 \times 6 + 52 \times 4 = 256 \text{ EUR} \\
\text{Total revenue:}\quad & 374 \text{ EUR} \\
\text{Fixed costs:}\quad & 40 + 50 = 90 \text{ EUR} \\
\text{Net profit:}\quad & 374 - 90 = 284 \text{ EUR}
\end{aligned}$$

Managerial Insights

1. **Product mix strategy:** Focus production on Products 1 and 2. Do not develop or market Product 3 unless profit margins improve substantially or resource constraints change.

2. **Capacity investment:** Resource 4 is the bottleneck. Any capacity expansion efforts should prioritize this resource. Increasing Resource 4 capacity by even small amounts could significantly improve profitability.

3. **Product 2 maximization:** Product 2 is produced at maximum capacity (60 units), generating 256 EUR in revenue. This is the most valuable product and should be prioritized in production planning.

4. **Pricing strategy:** The piecewise profit structure is being fully utilized. The company successfully sells all units in both higher-profit and lower-profit segments for both products, indicating pricing power and market depth.

5. **Sensitivity analysis:** The 99% utilization of Resource 4 suggests that small variations in resource availability or product requirements could significantly impact the optimal solution. Conducting sensitivity analysis on Resource 4 availability and its shadow price would provide valuable insights for strategic planning.