Prince Nwaekwu – Python & CPLEX ILOG Implementation and Comparison – Hard Constraint Programming

To formulate the linear programming model for task 6, considering all three types of contracts and their monthly rental costs, we can define the decision variables as follows:

Let:

x1 = Number of 12-month contracts

x2 = Number of 6-month contracts

x3 = Number of 3-month contracts

The objective is to minimize the total cost, which is the sum of the costs associated with each type of contract. The objective function can be expressed as:

Minimize:

Cost = 12 * 5000 * x1 + 6 * 6000 * x2 + 3 * 8000 * x3

The constraints are as follows:

Delivery Load Constraints: The total delivery load in each month should be covered by the rented trucks' capacity. This can be expressed as:

12 * 150 * x1 + 6 * 150 * x2 + 3 * 150 * x3 >= Monthly Delivery Load

Contract Duration Constraints: The number of contracts of each duration cannot exceed the available options. This can be expressed as:

x1 <= Maximum Number of 12-month Contracts

x2 <= Maximum Number of 6-month Contracts

x3 <= Maximum Number of 3-month Contracts

Non-negativity Constraints: The number of contracts cannot be negative. This can be expressed as:

x1 >= 0

x2 >= 0

x3 >= 0

Python Implementation:

```python
import cplex

model = cplex.Cplex()

numMonths = 12
num12MonthContracts = model.variables.add(
    lb=[0],
    ub=[cplex.infinity],
    types=[model.variables.type.integer],
    names=["num12MonthContracts"]
)
num6MonthContracts = model.variables.add(
    lb=[0],
    ub=[cplex.infinity],
    types=[model.variables.type.integer],
    names=["num6MonthContracts"]
)
num3MonthContracts = model.variables.add(
    lb=[0],
    ub=[cplex.infinity],
    types=[model.variables.type.integer],
    names=["num3MonthContracts"]
)

deliveryLoad = [423, 434, 580, 873, 1297, 1735, 2211, 1667, 1335, 850, 594, 423]

model.objective.set_linear("num12MonthContracts", 12 * 5000)
model.objective.set_linear("num6MonthContracts", 6 * 6000)
model.objective.set_linear("num3MonthContracts", 3 * 8000)
model.objective.set_sense(model.objective.sense.minimize)

load_constraint = [
    cplex.SparsePair(["num12MonthContracts"], [12 * 150]),
    cplex.SparsePair(["num6MonthContracts"], [6 * 150]),
    cplex.SparsePair(["num3MonthContracts"], [3 * 150])
]
model.linear_constraints.add(
    lin_expr=load_constraint,
    senses=["G", "G", "G"],
    rhs=[0, 0, 0]
)

model.solve()
```

```python
num12MonthContracts_value = model.solution.get_values("num12MonthContracts")
num6MonthContracts_value = model.solution.get_values("num6MonthContracts")
num3MonthContracts_value = model.solution.get_values("num3MonthContracts")

# Heuristic algorithm
remainingLoad = sum(deliveryLoad)

while remainingLoad > 0:
    if remainingLoad >= 5 * deliveryLoad[6]:
        num12MonthContracts_value += 1
        remainingLoad -= deliveryLoad[6]
    elif remainingLoad >= 2 * deliveryLoad[6]:
        num6MonthContracts_value += 1
        remainingLoad -= deliveryLoad[6]
    else:
        num3MonthContracts_value += 1
        remainingLoad -= deliveryLoad[6]

cost12MonthContracts = num12MonthContracts_value * 12 * 5000
cost6MonthContracts = num6MonthContracts_value * 6 * 6000
cost3MonthContracts = num3MonthContracts_value * 3 * 8000
totalCost = cost12MonthContracts + cost6MonthContracts + cost3MonthContracts

print("Number of 12-month contracts:", num12MonthContracts_value)
print("Number of 6-month contracts:", num6MonthContracts_value)
print("Number of 3-month contracts:", num3MonthContracts_value)
print("Associated cost:", totalCost)
```

Output Solution:

```
print("Number of 3-month contracts:", num3MonthContracts_value)
print("Associated cost:", totalCost)
```

```
Version identifier: 22.1.1.0 | 2023-02-09 | 22d6266e5
CPXPARAM_Read_DataCheck                          1
Found incumbent of value 0.000000 after 0.00 sec. (0.00 ticks)

Root node processing (before b&c):
  Real time             =    0.00 sec. (0.00 ticks)
Parallel b&c, 8 threads:
  Real time             =    0.00 sec. (0.00 ticks)
  Sync time (average)   =    0.00 sec.
  Wait time (average)   =    0.00 sec.
                          ------------
Total (root+branch&cut) =    0.00 sec. (0.00 ticks)
Number of 12-month contracts: 1.0
Number of 6-month contracts: 3.0
Number of 3-month contracts: 2.0
Associated cost: 216000.0
```

The output solution shows that, in order to meet the annual fuel oil delivery requirements, it is best to enter into one 12-month contract, three 6-month contracts, and two 3-month contracts, according to the data and limits provided. The total cost of these contracts together is 216,000 Euros.

Solution Comparisons

breakdown of the comparison from earlier ILOG solutions submitted (Solution 1, 2 and 3 from previous report):

Solution 1:

In solution 1, the result shows us that taking three 12-month contracts in the first two months (July and August) when only one 12-month contract is needed in the first month to manage the demand of the second month.

Taking 15 3-month contracts during the high-demand period when they are the most expensive.

The total cost is 6,877,000 euros, which is considered high.

Solution 2:

For Solution 3, Signing 15 twelve-month contracts in July and zero contracts for the other months.

This solution ensures the necessary number of trucks to carry the demanded quantity of fuel with minimum costs. The total cost is 900,000 euros.

Solution 3:

Recognizing that the initial constraints don't accurately model the problem.

Trying to account for all contracts signed in earlier months that are still in effect.

Facing syntax errors or exceeding the search space in the implementation.

Conclusion:

Comparing these solutions to your code's result:

The result from the Python solution in this document suggests taking 1 contract of 12 months, 3 contracts of 6 months, and 2 contracts of 3 months.

The associated cost is 216,000 euros.

Based on this comparison, it seems that this result offers a more cost-effective solution compared to Solution 1, where the cost is significantly higher. However, it's important to note that Solution 2, with a total cost of 900,000 euros, appears to be the most optimized solution among the options provided.