Modelowanie Matematyczne, Projekt 3, Dane nr 1.6

Krzysztof Rudnicki, 307585

24 stycznia 2024

0 Wstęp

2 Fabryki, F1, F2
4 Magazyny, M1, M2, M3, M4
6 Klientów, K1, K2, K3, K4, K5, K6

Koszty dystrybucji towaru

zaopatruje	F1	F2	M1	M2	M3	M4
Magazyny						
M1	0.3	-				
M2	0.5	0.4				
M3	1.2	0.5				
M4	0.8	0.3				
Klientów						
K1	1.2	1.8	-	1.4	-	-
K2	-	-	1.2	0.3	1.3	-
K3	1.2	-	0.2	1.8	2.0	0.4
K4	2.0	-	1.7	1.3	-	2.0
K5	-	-	-	0.5	0.3	0.5
K6	1.1	-	2.0	-	1.4	1.5

1 Model Dwukryterialny

Zbiory

- $i, k \in F$ - Fabryki
- $i, l \in M$ - Magazyny
- $j \in K$ - Klienci

Parametery

- $C_{i,j}$ - Koszty transportu dóbr z punktów i (fabryka lub magazyn) do klienta j
- $C_{k,l}$ - Koszty transportu dóbr z fabryki i do magazynu k
- $P_{i,j}$ - Binarnie określa czy preferencja klienta zostąła spełniona (1) czy nie (0)
- S_j - Poziom satysfakcji klienta j - poziom satysfakcji liczymy w zależności od tego ile dany klient zamówił towaru
  Zadowolenie klientów którzy zamawiają więcej towarów jest traktowane priorytetowo:
  $$S_1 = \frac{50}{60}$$
  $$S_2 = \frac{10}{60}$$
  $$S_3 = \frac{40}{60}$$
  $$S_4 = \frac{35}{60}$$
  $$S_5 = \frac{60}{60}$$
  $$S_6 = \frac{20}{60}$$
- D_j - Zapotrzebowanie klienta j

Zmienne decyzyjne

- $x_{i,j}$ - Liczba dóbr (w tys. ton) przetransportowana z punktu i (fabryka lub magazyn) do klienta j
- $y_{k,l}$ - Liczba dóbr (w tys. ton) przetransportowana z fabryki k do magazynu l

Funkcja celu

1. Minimalizacja kosztów dystrybucji
$$Min(\sum_{i,j} C_{i,j} * x_{i,j} + \sum i, kC_{i,k} * y_{i,k})$$
2. Maksymalizacja satysfakcja klienta
W celu maksymalizacji satysfakcji klienta policzymy ile z dostaw do klientów odbyło się z preferencyjnych źródeł
$$Max(\sum_{j} S_j * P_{i,j} * x_{i,j})$$

Funkcja celu alpha beta

$$\alpha * (Min(\sum_{i,j} C_{i,j} * x_{i,j} + \sum i, kC_{i,k} * y_{i,k})) + \beta * (Max(\sum_{j} S_j * P_{i,j} * x_{i,j}))$$

Ograniczenia Miesięczne możliwości produkcyjne fabryk

$$\sum_{j} x_{F1,j} + \sum_{k} y_{F1,k} \leq 150 \tag{1}$$

$$\sum_{j} x_{F2,j} + \sum_{k} y_{F2,k} \leq 200 \tag{2}$$

Miesięczna ilośc obsługiwanego towaru przez magazyny

$$\sum_{j} x_{M1,j} \leq 70 \tag{3}$$

$$\sum_{j} x_{M2,j} \leq 50 \tag{4}$$

$$\sum_j x_{M3,j} \leq 100 \tag{5}$$

$$\sum_j x_{M4,j} \leq 40 \tag{6}$$

Spełnienie preferencji klienta

$$\sum_i x_{i,j} = D_j \tag{7}$$

Wartości niezerowe

$$x_{i,j}, y_{i,k} \geq 0 \tag{8}$$

2 Implementacja

Do implementacji użyty został python z biblioteką pulp https://coin-or.github.io/pulp/index.html Dzięki temu wykorzystujemy zarówno łatwość pythona jak i możliwości używania różnych solverów (CBC, GLPK, CPLEX, Gurobi...) przez pulpa

Listing 1: Import bilbioteki pulp i bibliotek używanych do wykresu

```
import pulp
import sys
```

Listing 2: Iniicjalizacja modelu

```
model = pulp.LpProblem("Optimal_Distribution", pulp.LpMinimize)
```

Listing 3: Zdefiniowanie zmiennych i parametrów

```
fabryki = ['F1', 'F2']
magazyny = ['M1', 'M2', 'M3', 'M4']
dostawcy = fabryki + magazyny

klienci = ['K1', 'K2', 'K3', 'K4', 'K5', 'K6']

koszt = {
    'F1': {'M1': 0.3, 'M2': 0.5, 'M3': 1.2, 'M4': 0.8,
    'K1': 1.2, 'K2': 999, K3':␣1.2,␣'K4':␣2.0,
␣␣␣␣␣␣␣␣'K5':␣999,␣'K6':␣1.1},
␣␣␣␣␣␣␣␣'F2':␣{'M1':␣999,␣'M2':␣0.4,␣'M3':␣0.5,␣'M4':␣0.3,
```

```
        'K1': 1.8, 'K2': 999, 'K3': 999, 'K4': 999,
        'K5': 999, 'K6': 999},
        'M1': {'K1': 999, 'K2': 1.2, 'K3': 0.2, 'K4': 1.7,
        'K5': 999, 'K6': 2.0},
        'M2': {'K1': 1.4, 'K2': 0.3, 'K3': 1.8, 'K4': 1.3,
        'K5': 0.5, 'K6': 999},
        'M3': {'K1': 999, 'K2': 1.3, 'K3': 2.0, 'K4': 999,
        'K5': 0.3, 'K6': 1.4},
        'M4': {'K1': 999, 'K2': 999, 'K3': 0.4, 'K4': 2.0,
        'K5': 0.5, 'K6': 1.6}
    }
    P = pulp.LpVariable.dicts(
        "P", [(i, j) for i in dostawcy for j in klienci],
        cat='Binary')
    maksymalne_zamowienie = 60
    poziom_satysfakcji = {
        'K1': 50 / maksymalne_zamowienie,
        'K2': 10 / maksymalne_zamowienie,
        'K3': 40 / maksymalne_zamowienie,
        'K4': 35 / maksymalne_zamowienie,
        'K5': 60 / maksymalne_zamowienie,
        'K6': 20 / maksymalne_zamowienie}
    preferencja_klienta = {
        'K1': ['F2'], 'K2': ['M1'], 'K3': ['M2', 'M3'],
        'K4': ['F1'], 'K5': [], 'K6': ['M3', 'M4']}
    suma_satysfakcji = pulp.lpSum(
        [poziom_satysfakcji[j] * P[(i, j)]
        for i in dostawcy for j in klienci])
```

Listing 4: Zmienne decyzyjne

```python
x = pulp.LpVariable.dicts(
    "x",
    [(i, j) for i in dostawcy for j in klienci],
    lowBound=0, cat='Integer')
y = pulp.LpVariable.dicts("y",
[(i, k) for i in fabryki for k in magazyny],
lowBound=0, cat='Integer')
```

Listing 5: Funkcje celu

```
koszt_dystrybucji = pulp.lpSum(
    [koszt[i][j] * x[(i, j)]
    for i in dostawcy for j in klienci])
koszt_magazynowania = pulp.lpSum(
    [koszt[i][k] * y[(i, k)]
    for i in fabryki for k in magazyny])
alpha = 0.5
beta = 0.5
model += alpha
* (koszt_dystrybucji + koszt_magazynowania)
- beta * suma_satysfakcji
```

Listing 6: Ograniczenia

```
for i in fabryki:
    model += pulp.lpSum(
        [x[(i, j)] for j in klienci]
        + [y[(i, k)] for k in magazyny])
        <= mozliwosci_fabryki[i]

for k in magazyny:
    model += pulp.lpSum(
        [x[(k, j)] for j in klienci]
        ) <= pojemnosc_magazynu[k]

for j in klienci:
    model += pulp.lpSum(
        [x[(i, j)] for i in dostawcy]) == zamowienia_klientow[j]
```

Listing 7: Rozwiązanie problemu

```
model.solve()
```

Listing 8: Przedstawienie wyników

```
for v in model.variables():
    print(v.name, "=", v.varValue)
```

3 Rozwiązanie efektywne

Aby zdefiniować rozwiązanie efektywne sprawdzamy sumaryczy obu funkcji celu dla wartości α i *beta* od 1 do 10
w tym celu napisany został kod który modyfikuje wartości α i β w pętli

Listing 9: Wyznaczanie alpha i beta

```
koszt_wyniki = []
zadowolenie_wyniki = []
wyniki_funkcji = []
maksymalny_wynik = 0;
for alpha in range(0, 11):
    beta = 10 - alpha + sys.float_info.epsilon
    alpha /= 10.0
    beta /= 10.0
    # Update objective function
    model.objective = alpha * koszt_dystrybucji
    - beta * suma_satysfakcji

    # Solve the model
    model.solve()
    print(alpha)

    # Record the wyniki
    calkowity_koszt = pulp.value(koszt_dystrybucji)
    calkowite_zadowolenie = pulp.value(suma_satysfakcji)
    wynik_funkcji = pulp.value(alpha * koszt_dystrybucji
    - beta * suma_satysfakcji)
    koszt_wyniki.append(calkowity_koszt)
    zadowolenie_wyniki.append(calkowite_zadowolenie)
    if wynik_funkcji > maksymalny_wynik:
        maksymalny_wynik = wynik_funkcji
    wyniki_funkcji.append(wynik_funkcji)

print("maksymalny_wynik", maksymalny_wynik, wyniki_funkcji)
```

Najwyższy wynik został uzyskany dla $\alpha = 10$ i $\beta = 0$ i wynosił on **156.5**

Symulacja procesu podejmowania decyzji Przeprowadzone zostały symulacje dla 5 sytuacji:

1. Tylko minimalizacja kosztów $\alpha = 1.0$ $\beta = 0.0$
2. Priorytet na minimalizacji kosztów $\alpha = 0.8$ $\beta = 0.2$
3. Równy podział $\alpha = 0.5$ $\beta = 0.5$
4. Priorytet na satysfakcji klientów $\alpha = 0.2$ $\beta = 0.8$
5. Tylko satysfakcja klientów $\alpha = 0.0$ $\beta = 1.0$

Ponownie w celu przeprowadzenia symulacji napisano kod w pythonie

```
# Pseudo-code, assuming model setup as previously discussed
scemariusze = [
    (1.0, sys.float_info.epsilon),
    (0.8, 0.2),
    (0.5, 0.5),
    (0.2, 0.8),
    (sys.float_info.epsilon, 1.0)]
wyniki = []

for alpha, beta in scemariusze:
    model.objective = alpha * koszt_dystrybucji
        - beta * suma_satysfakcji

    model.solve()

    calkowity_koszt = pulp.value(koszt_dystrybucji)
    calkowite_zadowolenie = pulp.value(suma_satysfakcji)
    wyniki.append(
        (alpha, beta, calkowity_koszt, calkowite_zadowolenie)
        )

for idx, (alpha, beta, koszt, zadowolenie) in enumerate(wyniki):
    print(f"Krok {idx+1}:")
    print(f"  koszt: {alpha}, zadowolenie: {beta}")
    print(f"
```

```
␣␣␣␣␣Calkowity␣koszt:␣{koszt},
␣␣␣␣␣zadowolenie␣klienta:␣{zadowolenie}\n")
```