دانشگاه صنعتی شریف
دانشکده مهندسی صنایع

تمرین کدنویسی
الگوریتم تجزیه دنتزیگ‌ولف

درس:

روش‌های تجزیه در بهینه‌سازی

استاد:

سرکار خانم دکتر مریم رادمان

دانشجویان:

نازنین قائمی‌زاده (۴۰۲۲۰۰۸۴۲)
سیده طراوت بلادیان بهبهان (۴۰۳۲۰۵۳۳۳)

پاییز ۱۴۰۳

الگوریتم دنتزیگ‌ولف: ساختار بلوکی با یک محدودیت مشترک

مدل بهینه‌سازی مدنظر به صورت زیر قابل نمایش است:

$$
\begin{aligned}
\min\ & -x_1 - 8x_2 - 5y_1 - 6y_2 \\
& \boxed{x_1 + 4x_2 + 5y_1 + 2y_2 \le 7} \\
& \boxed{\begin{array}{ll} 5x_1 + x_2 & \le 5 \\ 2x_1 + 3x_2 & \le 6 \end{array}} \\
& \boxed{\begin{array}{ll} 1y_1 & \le 4 \\ 3y_1 + 4y_2 & \ge 12 \\ y_2 & \le 3 \end{array}} \\
& x_1, x_2, y_1, y_2 \ge 0
\end{aligned}
$$

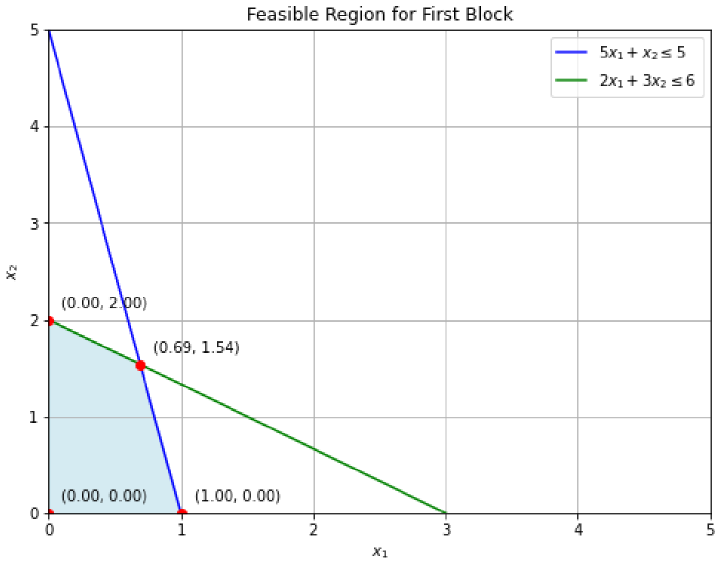
در راستای به کار بردن الگوریتم دنتزیگ‌ولف، لازم است بلوک‌های مسئله به صورت زیر شناسایی گردد:

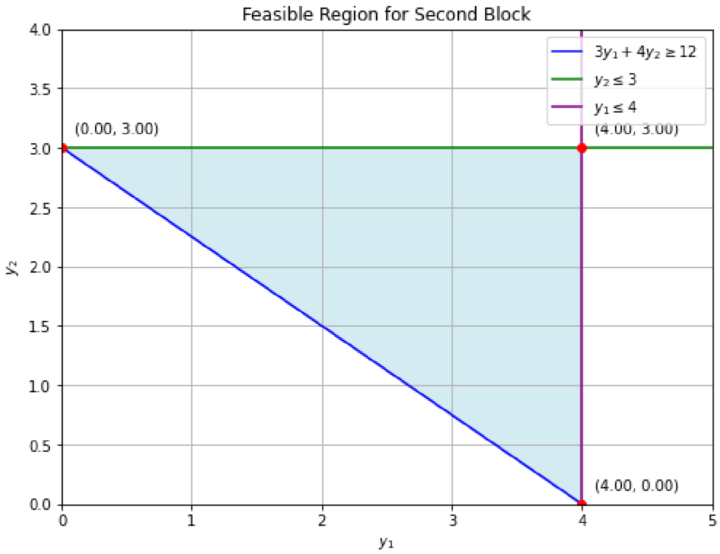
- محدودیت نخست، محدودیت مشترک مسئله می‌باشد.
- بلوک اول: $X \in P_1 \rightarrow X = \begin{pmatrix} x_1 \\ x_2 \end{pmatrix}$
- بلوک دوم: $Y \in P_2 \rightarrow Y = \begin{pmatrix} y_1 \\ y_2 \end{pmatrix}$

در ادامه مدل اصلی مسئله، به فرم استاندارد تبدیل می‌شود:

$$
\begin{aligned}
\min\ & -x_1 - 8x_2 - 5y_1 - 6y_2 \\
& x_1 + 4x_2 + 5y_1 + 2y_2 + S = 7 \\
& 5x_1 + x_2 \le 5 \\
& 2x_1 + 3x_2 \le 6 \\
& 1y_1 \le 4 \\
& 3y_1 + 4y_2 \ge 12 \\
& y_2 \le 3 \\
& x_1, x_2, y_1, y_2, S_1 \ge 0
\end{aligned}
$$

از آن‌جایی که بلوک‌های مسئله یک فضای دوبعدی را نشان می‌دهند، بنابراین فضای جواب هر بلوک قابل ترسیم است.





فضای شدنی بلوک‌ها نشان می‌دهد در این مسئله هیچ جهت حدی وجود ندارد، بنابراین بازنویسی مدل به صورت زیر صورت می‌گیرد:

$$\min C_1X + C_2Y + C_3S$$

$$E_1X + E_2Y + S = 7$$

$$X \in P_1$$

$$Y \in P_2$$

همچنین می‌دانیم که:

$$X = \sum_{i=1}^{4} \alpha_i x_i \quad , \quad Y = \sum_{j=1}^{3} \beta_j y_j$$

$$\sum_{i=1}^{4} \alpha_i = 1 \quad , \quad \sum_{j=1}^{3} \beta_j = 1$$

بنابراین خواهیم داشت:

$$\min \sum_{i=1}^{4} (C_1 x_i)\alpha_i + \sum_{j=1}^{3} (C_2 y_j)\beta_j + C_3 S$$

$$\sum_{i=1}^{4} (E_1 x_i)\alpha_i + \sum_{j=1}^{3} (E_2 y_j)\beta_j + S = 7$$

$$\sum_{i=1}^{4} \alpha_i = 1$$

$$\sum_{j=1}^{3} \beta_j = 1$$

$$\alpha_i, \beta_j \geq 0$$

برای پیشبرد سایر محاسبات، لازم است ضرایب به صورت زیر مشخص شوند:

$$C_1 = (-1, -8) \qquad E_1 = (1, 4)$$

$$C_2 = (-5, -6) \qquad E_2 = (5, 2)$$

$$C_3 = (0)$$

کد الگوریتم دنتزیگ‌ولف، با استفاده از کتابخانه PYOMO نوشته شده است. در ابتدا لازم است داده‌های اولیه تعریف شوند. C_1 ضرایب مربوط به متغیرهای x در تابع هدف می‌باشد. C_2 نیز ضرایب مربوط به متغیرهای y در تابع هدف می‌باشد. E_1 و E_2 نیز ضرایب مربوط به متغیرهای بلوک اول و دوم در محدودیت مشترک می‌باشد. برای شروع الگوریتم نیاز است یک پایه موجه اولیه در نظر گرفته شود، بنابراین نقطه گوشه‌ای $\binom{0}{0}$ از فضای شدنی بوک اول و نقطه گوشه‌ای $\binom{0}{3}$ از فضای شدنی بلوک دوم مسئله انتخاب می‌گردد.

```
# Given data
C1 = np.array([-1, -8])  # Coefficients for x variables in the
master problem
C2 = np.array([-5, -6])  # Coefficients for y variables in the
master problem

E1 = np.array([1, 4])    # Coefficients in coupling constraint for x
variables
E2 = np.array([5, 2])    # Coefficients in coupling constraint for y
variables

initial_X1 = np.array([0, 0])  # Initial solution for x variables
initial_Y1 = np.array([0, 3])  # Initial solution for y variables
```

در کد ارائه‌شده، خط اول یک مدل ایجاد می‌کند. این مدل از نوع ConcreteModel است، که به معنی تعریف صریح تمامی اجزای مدل (متغیرها، محدودیت‌ها، و تابع هدف) در زمان ایجاد آن است. برخلاف مدل‌های انتزاعی، که ساختار مسئله جدا از داده‌ها تعریف می‌شود و داده‌ها بعداً به مدل اعمال می‌شوند، در مدل صریح تمام داده‌ها و ساختار مسئله به‌طور همزمان مشخص هستند. این انتخاب زمانی مناسب است که تمام جزئیات مسئله از پیش مشخص باشند، مانند مسئله ارائه‌شده در این کد. خط دوم، یک Suffix به مدل اضافه می‌کند که برای ذخیره اطلاعات اضافی مرتبط با حل مسئله استفاده می‌شود. در اینجا، برای ذخیره مقادیر دوگان محدودیت‌های مسئله تعریف شده است. به‌طور خاص، در الگوریتم دنتزیگ‌ولف، این مقادیر دوگان در تنظیم و حل زیرمسائل استفاده می‌شوند.

```
# Initialize RMP model
model = pyo.ConcreteModel()
model.dual = pyo.Suffix(direction=pyo.Suffix.IMPORT)
```

در کد، سه متغیر اصلی برای مدل محدود شده اصلی تعریف شده‌اند که شامل $\{S, \alpha_1, \beta_1\}$ است. این متغیرها با استفاده از دستور pyo.Var(within=pyo.NonNegativeReals) ایجاد می‌شوند، که بیان می‌کند این متغیرها باید مقادیر غیرمنفی داشته باشند. متغیر α_1 به‌عنوان وزن ترکیب خطی مقادیر x در مسئله اصلی عمل می‌کند، در حالی که β_1 وزن مشابهی برای مقادیر y است. این دو متغیر نقش اساسی در

بهینه‌سازی تابع هدف و محدودیت‌های مسئله ایفا می‌کنند، زیرا به مدل اجازه می‌دهند ترکیب خطی بهینه‌ای از مقادیر x و y پیدا کند. متغیر سوم، یک متغیر کمکی است که برای برقراری محدودیت مشترک استفاده می‌شود. این متغیر به حل‌کننده کمک می‌کند تا مسئله را حتی در شرایطی که محدودیت‌های سخت‌گیرانه وجود دارند، حل کند و در عین حال اطمینان حاصل شود که تمامی محدودیت‌ها برآورده می‌شوند. تعریف این سه متغیر پایه‌ای برای ساختار مدل RMP است و امکان مدیریت وزن‌دهی و برقراری محدودیت‌های مشترک را فراهم می‌کند.

```
# Define initial variables
model.alpha1 = pyo.Var(within=pyo.NonNegativeReals)
model.beta1 = pyo.Var(within=pyo.NonNegativeReals)
model.S = pyo.Var(within=pyo.NonNegativeReals)
```

در این بخش از کد، تابع هدف مدل محدود شده اصلی تعریف می‌شود. دستور pyo.Objective(...) تابع هدف را به عنوان یک ترکیب خطی از وزن‌دهی متغیرهای x و y تنظیم می‌کند. این ترکیب خطی با استفاده از ضرایب مربوط به متغیرها در تابع هدف و نقاط گوشه‌ای موجه اولیه محاسبه می‌شود. همچنین، این تابع با مقداردهی (sense=pyo.minimize) به حل‌کننده اطلاع می‌دهد که هدف از نوع کمینه‌سازی است.

```
# Objective function
model.Obj = pyo.Objective(expr=np.dot(C1, initial_X1) *
model.alpha1     +     np.dot(C2,     initial_Y1)     *     model.beta1,
sense=pyo.minimize)
```

در این بخش، سه محدودیت برای مدل تعریف شده است. اولین محدودیت نشان‌دهنده بازنویسی مجددی از محدودیت مشترک مسئله است. همچنین، محدودیت دوم و سوم نیز محدودیت‌های تحدب مربوط به بلوک اول و دوم را نشان می‌دهد که وزن‌های ترکیب خطی را به مقدار یک محدود می‌کند.

```
# Initial constraints
model.Constraint1  =  pyo.Constraint(expr=np.dot(E1,  initial_X1)  *
model.alpha1 + np.dot(E2, initial_Y1) * model.beta1 + model.S == 7)
model.Constraint2 = pyo.Constraint(expr=model.alpha1 == 1)
model.Constraint3 = pyo.Constraint(expr=model.beta1 == 1)
```

در این قسمت، متغیرهای کنترلی و مقادیر اولیه الگوریتم تنظیم می‌شوند. متغیر iteration تعداد دفعات تکرار الگوریتم را ردیابی می‌کند. متغیرهای alpha_count و beta_count به مدیریت تعداد ضرایب متناظر با نقاط گوشه‌ای هر بلوک می‌پردازد. دو متغیر منطقی x_done و y_done نیز تعریف شده است که بر خاتمه یافتن زیرمسائل نظارت دارد. پارامتر تولرانس نیز برای تعیین سطح پذیرش هزینه کاهش‌یافته زیرمسائل

تعریف شده است که مقادیر بسیار کوچک را به صفر تقریب می‌زند. دو لیست نیز برای ذخیره مقادیر بهینه بدست آمده از هر زیر مسائل به کار گرفته شده است.

```
# Initialize iteration trackers
iteration = 0
alpha_count = 1
beta_count = 1
x_done = False
y_done = False
tolerance = 1e-8 # Define tolerance for reduced costs

# Lists to store subproblem solutions
x_star_values = []
y_star_values = []
```

حلقه اصلی با استفاده از شرط while not (x_done and y_done) تا زمانی ادامه می‌یابد که هر دو زیرمسئله به شرایط خاتمه برسند. در هر تکرار ابتدا مسئله اصلی محدود حل می‌گردد و مقادیر دوگان متناظر محدودیت‌ها با استفاده از دستور model.dual استخراج می‌شود. این مقادیر دوگان برای تنظیم توابع هدف زیرمسائل استفاده می‌شوند. با این روش، حلقه به‌طور پویایی مسئله را بهینه‌سازی می‌کند تا زمانی که هیچ ستون جدیدی برای افزودن وجود نداشته باشد.

```
while not (x_done and y_done):
iteration += 1
print(f"\nIteration {iteration}:")

# Solve the RMP
solver.solve(model)

# Retrieve dual values
dual_values = [
model.dual[model.Constraint1],
model.dual[model.Constraint2],
model.dual[model.Constraint3]
         ]
```

زیرمسئله مربوط به بلوک اول در قالب یک تابع تعریف شده است. در این تابع، یک مدل PYOMO جدید برای متغیرهای x_1 و x_2 ساخته می‌شود. تابع هدف این زیرمسئله با استفاده از ضرایب C_1، E_1 و مقادیر دوگان بدست آمده در گام پیشین شکل می‌گیرد. دو محدودیت جدید نیز برای کنترل فضای شدنی بلوک اول تعریف می‌شوند. پس از حل زیر مسئله، مقادیر بهینه متغیرها و هزینه کاهش‌یافته محاسبه می‌شوند. این

مقادیر به الگوریتم بازگردانده می‌شوند تا به کمک آن‌ها مسئله اصلی محدود را به‌روزرسانی نماید. کلیه موارد توضیح داده شده، برای بلوک دوم نیز صادق است.

```
# Subproblem for x variables
def Subproblem_x(dual_values):
model_x = pyo.ConcreteModel()
model_x.x1 = pyo.Var(within=pyo.NonNegativeReals)
model_x.x2 = pyo.Var(within=pyo.NonNegativeReals)

# Objective function
model_x.Obj = pyo.Objective(expr=(C1[0] - dual_values[0] * E1[0]) *
model_x.x1  +  (C1[1]  -  dual_values[0]  *  E1[1])  *  model_x.x2,
sense=pyo.minimize)

# Constraints
model_x.constraints = pyo.ConstraintList()
model_x.constraints.add(5 * model_x.x1 + model_x.x2 <= 5)
model_x.constraints.add(2 * model_x.x1 + 3 * model_x.x2 <= 6)

solver.solve(model_x)
x1_star = pyo.value(model_x.x1)
x2_star = pyo.value(model_x.x2)
reduced_cost_x = pyo.value(model_x.Obj) - dual_values[1]

return (x1_star, x2_star), reduced_cost_x
```

پس از محاسبه هزینه کاهش‌یافته برای هر زیرمسئله، الگوریتم شرایط خاتمه را ارزیابی می‌کند. اگر هزینه کاهش‌یافته هر بلوک غیرمنفی باشد، متغیرهای منطقی x_done و y_done به مقدار TRUE تغییر می‌کنند. در غیر این صورت نقاط گوشه‌ای جدید به لیست ساخته شده اضافه شده و الگوریتم ادامه می‌یابد.

```
x_star, reduced_cost_x = Subproblem_x(dual_values) if not x_done else
(None, 0)
y_star, reduced_cost_y = Subproblem_y(dual_values) if not y_done else
(None, 0)

# Treat very small reduced costs as zero
reduced_cost_x  =  0  if  abs(reduced_cost_x)  <  tolerance  else
reduced_cost_x
reduced_cost_y  =  0  if  abs(reduced_cost_y)  <  tolerance  else
reduced_cost_y

print(f"Reduced cost X: {reduced_cost_x}")
print(f"Reduced cost Y: {reduced_cost_y}")
```

```
# Update termination conditions
if reduced_cost_x >= 0:
x_done = True
else:
x_star_values.append(x_star)

if reduced_cost_y >= 0:
y_done = True
else:
y_star_values.append(y_star)
```

اگر هزینه کاهش‌یافته متغیرهای جدید منفی باشد، ستون جدیدی به مسئله اصلی محدود اضافه می‌شود. این فرآیند شامل ایجاد یک متغیر جدید α_i یا β_j، افزودن آن به تابع هدف و به‌روزرسانی محدودیت‌های مسئله است. این ستون‌ها تأثیرات ترکیب خطی جدیدی را به مدل اضافه می‌کنند و به الگوریتم اجازه می‌دهند به سمت جواب بهینه حرکت کند. برای هر ستون جدید، شمارنده مربوط به متغیرهای alpha_count و beta_count افزایش می‌یابد.

```
# Add new variables if reduced cost is negative
if not x_done and reduced_cost_x < 0:
alpha_count += 1
new_alpha = pyo.Var(within=pyo.NonNegativeReals)
model.add_component(f'alpha{alpha_count}', new_alpha)
model.Obj.expr += np.dot(C1, x_star) * new_alpha
model.Constraint1.set_value(model.Constraint1.body    +    np.dot(E1,
x_star) * new_alpha == 7)
model.Constraint2.set_value(model.Constraint2.body + new_alpha == 1)

if not y_done and reduced_cost_y < 0:
beta_count += 1
new_beta = pyo.Var(within=pyo.NonNegativeReals)
model.add_component(f'beta{beta_count}', new_beta)
model.Obj.expr += np.dot(C2, y_star) * new_beta
model.Constraint1.set_value(
model.Constraint1.body + np.dot(E2, y_star) * new_beta == 7)
model.Constraint3.set_value(model.Constraint3.body + new_beta == 1)
```

پس از خاتمه یافتن الگوریتم، مقادیر نهایی تابع هدف و متغیرهای بهینه x و y محاسبه می‌شوند. این مقادیر با استفاده از ترکیب خطی ستون‌های ایجاد شده و وزن‌های مربوط به ضرایب نقاط گوشه‌ای متناظر با هر ستون است. مقدار تابع هدف نیز محاسبه و چاپ می‌شود تا کارایی الگوریتم ارزیابی شود. این بخش نشان‌دهنده نتیجه‌گیری نهایی و کاربرد موفق الگوریتم دنتزیگ-ولف برای مسئله ارائه‌شده است.

```
# Final results
print("\n--------------------")
print("\nFinal RMP result:")
print(f"Objective Value: {pyo.value(model.Obj)}")

# Retrieve values of alpha and beta
alpha_values = [pyo.value(var) for var in model.component_data_objects(pyo.Var) if var.name.startswith("alpha")]
beta_values = [pyo.value(var) for var in model.component_data_objects(pyo.Var) if var.name.startswith("beta")]

# Print the values
for var in model.component_data_objects(pyo.Var):
print(f"{var.name} = {pyo.value(var)}")
```

```
# Calculate X and Y
X = np.zeros(len(C1))
Y = np.zeros(len(C2))

for i, alpha in enumerate(alpha_values):
x_star = initial_X1 if i == 0 else x_star_values[i - 1]
X += alpha * np.array(x_star)

for j, beta in enumerate(beta_values):
y_star = initial_Y1 if j == 0 else y_star_values[j - 1]
Y += beta * np.array(y_star)

print("\n--------------------")
print("\nFinal calculated values:")
print(f"X = {X}")
print(f"Y = {Y}")
```

در تحلیل نتایج الگوریتم دنتزیگ‌ولف، در تکرار اول هزینه کاهش‌یافته هر دو بلوک منفی است، که نشان می‌دهد از هر دو زیرمسئله متغیر واردشونده وجود دارد. اما در تکرار دوم هر دو هزینه کاهش‌یافته به صفر رسیدند، که نشان‌دهنده عدم وجود ستون‌های بهبوددهنده و دستیابی به جواب بهینه است. مقدار بهینه تابع هدف ۲۰- واحد است و ضرایب متناظر با نقاط گوشه‌ای و همچنین مقدار بهینه متغیرهای مسئله به شرح زیر قابل‌نمایش هستند.

```
Iteration 1:
Reduced cost X: -16.0
Reduced cost Y: -20.0

Iteration 2:
Reduced cost X: 0
Reduced cost Y: 0

--------------------

Final RMP result:
Objective Value: -20.0
alpha1 = 0.875
beta1 = 1.0
S = 0.0
alpha2 = 0.125
beta2 = 0.0

--------------------

Final calculated values:
X = [0.   0.25]
Y = [0. 3.]
```

کد دیگری نیز، مشابه با همین الگوریتم برای مسئله زیر با ساختار بلوکی و دو محدودیت مشترک ارائه شده است، که تنها در ورودی‌های اولیه مسئله، فضای شدنی بلوک‌های مسئله و تعداد دوگان‌های متناظر با محدودیت‌های مسئله متفاوت است.

$$\min -2x_1 - 3x_2 - 2y_1 + y_2$$

$$\boxed{\begin{array}{lcr} x_1 \quad\quad + \; y_1 & & \leq 2 \\ x_1 + \; x_2 & + 2y_2 & \leq 3 \end{array}}$$

$$\boxed{\begin{array}{lr} x_1 + 2x_2 & \leq 5 \\ x_1 & \leq 2 \end{array}}$$

$$\boxed{\begin{array}{r} 2y_1 + \; y_2 \leq 6 \\ -1y_1 + \; y_2 \leq 2 \end{array}}$$

$$x_1, x_2, y_1, y_2 \geq 0$$

نتایج حاصله نیز به شرح زیر است:

```
Iteration 1:
Reduced cost X: -8.5
Reduced cost Y: -6.0

Iteration 2:
Reduced cost X: -4.285714285714275
Reduced cost Y: 0

Iteration 3:
Reduced cost X: 0
Reduced cost Y: 0

--------------------

Final RMP result:
Objective Value: -11.500000000000002
alpha1 = 0.0
beta1 = 0.3333333333333333
S1 = 0.0
S2 = 0.5
alpha2 = 0.0
beta2 = 0.6666666666666667
alpha3 = 1.0

--------------------

Final calculated values:
X = [0.  2.5]
Y = [2. 0.]
```