Programmation par contraintes

Rakotoarimalala Tisnjo

Librairies Python
(python-constraint,PULP et SCIPY)

Python-constraint

- Définition du problème:

```
problem = Problem()
```

- Définition des variables:

```
problem.addVariable(variable_name, domain)
```

- Ajout de contraintes:

```
problem.addConstraint(constraint_function, variables)
```

- Résolution du problème:

```
solutions = problem.getSolutions()
```

Exemple complet:

```
from constraint import Problem, AllDifferentConstraint

problem = Problem()
problem.addVariable('A1', ['chat', 'chien', 'canard'])
problem.addVariable('A2', ['chat', 'chien', 'canard'])
problem.addConstraint(AllDifferentConstraint())
solutions = problem.getSolutions()
```

n-queens en Python-constraint

```python
from constraint import Problem, AllDifferentConstraint

def n_queens(n):
  problem = Problem()

  # Création des variables représentant les colonnes où se trouvent les reines
  columns = range(n)
  rows = range(n)

  # Ajout des variables au problème
  for row in rows:
    problem.addVariable(row, columns)

  # Contraintes pour s'assurer qu'aucune reine ne se trouve sur la même colonne
  problem.addConstraint(AllDifferentConstraint())

  # Contraintes pour s'assurer qu'aucune reine ne se trouve sur la même
      diagonale
  for i in range(n):
    for j in range(i):
      problem.addConstraint(lambda xi, xj, i=i, j=j: abs(xi - xj) != i - j, (i,
      j))

  # Résolution du problème
  solutions = problem.getSolutions()

  # Affichage des solutions
  for solution in solutions:
    print([solution[row] for row in rows])

# Exemple d'utilisation pour un échiquier 8x8
n_queens(8)
```

n-queens en Python-constraint

Le point clé de cet exemple est la définition des contraintes

```
1     # Contraintes pour s'assurer qu'aucune reine ne se trouve sur la même
       diagonale
2     for i in range(n):
3       for j in range(i):
4         problem.addConstraint(lambda xi, xj, i=i, j=j: abs(xi - xj) != i - j, (i
       , j))
5
```

- Nous utilisons une fonction lambda pour définir la contrainte.
- La fonction prend deux arguments, ‘xi‘ et ‘xj‘, représentant les positions des reines sur l'échiquier.
- Les variables ‘i‘ et ‘j‘ sont capturées depuis la portée de la boucle ‘for‘ précédente pour représenter les indices des deux lignes que nous comparons.
- La contrainte vérifie que la différence absolue entre les colonnes des reines, ‘abs(xi - xj)‘, n'est pas égale à la différence entre les lignes correspondantes, ‘i - j‘.
- Si la différence absolue des colonnes est égale à la différence entre les lignes, cela signifie que les reines sont sur la même diagonale, ce qui est interdit dans le problème des N-Reines.

PULP

1. **Définition des variables** : Utilisez la classe `LpVariable` de PuLP pour définir les variables de décision de votre problème. Spécifiez les noms, les bornes et les types de variables nécessaires.
2. **Définition de l'objectif** : Utilisez la classe `LpProblem` pour définir votre problème d'optimisation. Spécifiez si vous souhaitez maximiser ou minimiser la fonction objectif.
3. **Ajout des contraintes** : Utilisez les méthodes de la classe `LpProblem` pour ajouter les contraintes (linéaires,d'égalité, d'inégalité) de votre problème.
4. **Résolution du problème** : Utilisez la méthode `solve()` pour résoudre le problème d'optimisation défini. Vous pouvez spécifier un solveur à utiliser, ou laisser PuLP choisir le solveur par défaut.
5. **Extraction des résultats** : Une fois le problème résolu, vous pouvez extraire les valeurs optimales des variables de décision en accédant à leurs attributs `varValue`.

Exemple d'utilisation de PULP

```python
from pulp import LpProblem, LpMinimize, LpVariable, LpStatus

#creation de problème
prob = LpProblem("Exemple_MILP", LpMinimize)

#creation des variables entières à calculer
x = LpVariable("x", cat="Integer")
y = LpVariable("y", cat="Integer")

#creation de la fonction objective
# Définition de la fonction objectif (Minimisation)
prob += 2 * x + 3 * y, "FonctionObjectif"

# Ajout des contraintes
prob += x + 2 * y >= 4, "Contrainte_1"
prob += 3 * x - y <= 6, "Contrainte_2"
prob += x >= 0, "Contrainte_3"
prob += y >= 0, "Contrainte_4"

# Résolution du problème
prob.solve()

# Affichage des résultats
print("Statut de la résolution apres:", LpStatus[prob.status])
print("Valeur optimale de la fonction objectif:", prob.objective.value())
print("Valeur optimale de x:", x.value())
print("Valeur optimale de y:", y.value())


```

SCIPY

1. Scipy est plutôt une bibliothèque pour les problèmes d'optimisation linéaires ou non linéaires
2. Par contre on peut l'utiliser avec des formulations plus ou moins complexes

SCIPY

1. **Importation de SciPy** : `scipy.optimize`, `scipy.optimize.minimize`,`scipy.linprog`,...
2. **Définition de la fonction objectif** : Écrivez une fonction Python qui représente la fonction que vous souhaitez optimiser.
3. **Définition des contraintes** : Définissez les contraintes qui doivent être respectées dans votre problème d'optimisation. Les contraintes peuvent être des contraintes d'égalité ou d'inégalité.
4. **Définition du problème d'optimisation** : Utilisez les fonctions fournies par SciPy pour définir votre problème d'optimisation en spécifiant la fonction objectif, les contraintes et les bornes des variables de décision, le cas échéant.
5. **Résolution du problème** : Appelez la fonction d'optimisation appropriée de SciPy pour résoudre votre problème d'optimisation. Assurez-vous de spécifier le type de méthode d'optimisation à utiliser et de passer les contraintes comme arguments.
6. **Analyse des résultats** : Une fois que le problème d'optimisation est résolu, analysez les résultats pour obtenir les valeurs optimales des variables de décision et les valeurs optimales de la fonction objectif.

SCIPY: n-queens

- **Variables de décision**:

$$x_{i,j} = \begin{cases} 1 & \text{si une reine est placée dans la case } (i,j) \\ 0 & \text{sinon} \end{cases}$$

où i et j varient de 1 à N, la taille de l'échiquier.

- **Fonction objective**: Dans le problème des N-Reines, nous cherchons simplement à trouver une solution, nous n'avons donc pas de fonction objective à maximiser ou minimiser.
- **Contraintes de ligne et de colonne**:

$$\sum_{j=1}^{N} x_{i,j} = 1 \quad \forall i \in \{1,2,...,N\}, \sum_{i=1}^{N} x_{i,j} = 1 \quad \forall j \in \{1,2,...,N\}$$

- **Contraintes de diagonale**:

$$\sum_{i-j=k} x_{i,j} \leq 1 \;\; \forall k \in \{-N+1,-N+2,...,N-1\}, \sum_{i+j=k} x_{i,j} \leq 1 \;\; \forall k \in \{2,3,...,2N\}$$

SCIPY: n-queens

Ces formulations deviennent par exemple pour $n = 8$

- **Contraintes sur les diagonales principales**: Les diagonales principales vont du coin supérieur gauche au coin inférieur droit de l'échiquier. Les contraintes sur les diagonales principales vérifient que chaque diagonale principale ne contient pas plus d'une reine :

$$\sum_{i-j=k} x_{i,j} \leq 1 \quad \forall k \in \{-7, -6, ..., 6, 7\}$$

- **Contraintes sur les contre-diagonales**: Les contre-diagonales vont du coin supérieur droit au coin inférieur gauche de l'échiquier. Les contraintes sur les contre-diagonales vérifient que chaque contre-diagonale ne contient pas plus d'une reine :

$$\sum_{i+j=k} x_{i,j} \leq 1 \quad \forall k \in \{2, 3, ..., 14\}$$

n-queens sous Scipy

```
1  import numpy as np
2  from scipy.optimize import linprog
3
4  # Fonction pour générer les contraintes pour les diagonales principales
5  def generate_main_diagonal_constraints(N):
6    A_main_diag = []
7    for k in range(1-N, N):
8      # Générer une ligne de contrainte pour chaque diagonale principale
9      diag_row = [1 if i - j == k else 0 for i in range(N) for j in range(N)]
10     A_main_diag.append(diag_row)
11   return np.array(A_main_diag)
12
13 # Fonction pour générer les contraintes pour les contre-diagonales
14 def generate_counter_diagonal_constraints(N):
15   A_counter_diag = []
16   for k in range(2, 2*N+1):
17     # Générer une ligne de contrainte pour chaque contre-diagonale
18     diag_row = [1 if i + j == k else 0 for i in range(N) for j in range(N)]
19     A_counter_diag.append(diag_row)
20   return np.array(A_counter_diag)
21
```

n-queens sous Scipy (suite)

```
1  # Taille de l'échiquier
2  N = 8
3  # Coefficients de la fonction objective (pas de fonction objective)
4  c = [0] * (N*N)
5  # Matrice des coefficients des contraintes (contraintes de ligne et de colonne)
6  A_row_col = np.eye(N*N)
7  # Termes constants des contraintes (1 reine par ligne et par colonne)
8  b_row_col = np.ones(N*N)
9  # Contraintes sur les diagonales principales
10 A_main_diag = generate_main_diagonal_constraints(N)
11 b_main_diag = np.ones(A_main_diag.shape[0])
12 # Contraintes sur les contre-diagonales
13 A_counter_diag = generate_counter_diagonal_constraints(N)
14 b_counter_diag = np.ones(A_counter_diag.shape[0])
15 # Empilage des contraintes
16 A = np.vstack((A_row_col, A_main_diag, A_counter_diag))
17 b = np.concatenate((b_row_col, b_main_diag, b_counter_diag))
18 # Bornes des variables (0 ou 1 pour chaque variable)
19 x_bounds = [(0, 1) for _ in range(N*N)]
20 # Résolution du problème d'optimisation linéaire
21 result = linprog(c, A_ub=A, b_ub=b, bounds=x_bounds)
22 # Affichage de la solution
23 print("La solution trouvée est:")
24 print(np.round(result.x.reshape((N, N))))
25
```

Exemple d'utilisation de Scipy pour des problèmes d'optimisation

```
1  from scipy.optimize import minimize
2
3  def fun(x):
4    return (x-2)**2
5
6  def fun2Variable(x):
7    return (x[0]-2)**2 + (x[1]-2)**2
8  #fonction fun à partir de x2
9  x0 = 0
10 result = minimize(fun, x0)
11 print(result)
12
13 #le resultat se trouve dans
14 print(result.x)
15
16 #pour la fonction à deux variables
17 x0 = [0,0]
18 result = minimize(fun2Variable,x0)
19 print(result)
20
```

Exemple d'utilisation de Scipy pour des problèmes d'optimisation

```
1  from scipy.optimize import minimize
2  #minimise à pour forme égalité sur les contraintes ou superieur ou égale 0 sur
       les inégalités
3  def objective(x):
4    return (x[0]-3)**2 + (x[1]+1)**2
5  #fonction pour la contrainte
6  def contrainte_fonction(x):
7    return x[0]+x[1]-5
8  def contrainte_positivite_x(x):
9    return x[0]
10 def contrainte_positivite_y(x):
11   return x[1]
12
13 bound = [(0,5),(0,5)]
14 #construction de la contrainte avec son type
15 contrainte = [{'type':'eq','fun':contrainte_fonction},
16         {'type':'ineq', 'fun':contrainte_positivite_x},
17         {'type':'ineq', 'fun':contrainte_positivite_y}
18         ]
19 #point de départ
20 x0= [0,0]
21 result = minimize(objective, x0,constraints = contrainte, bounds=bound)
22 print(result)
23
```

Algo d'optimisation les plus utilisés dans Scipy.minimize

1. **Trust-constr**: convient aux problèmes de minimisation avec des contraintes d'égalité et d'inégalité même pour des contraintes non linéaires.
2. **COBYLA**: convient aux problèmes de minimisation avec des contraintes d'inégalité.
3. **SLSQP**: adapté aux problèmes de minimisation avec des contraintes d'égalité et d'inégalité pour résoudre des problèmes non linéaires et non convexes.
4. ...

UTILISATION:

```
result = minimize(objective,x0, constraints=contraintes,options={'disp': True},
     method='COBYLA')
```

Exercice

- Supposons qu'on a une liste de médicaments $M = \{m_1, m_2, \ldots m_n\}$ de prix respectifs $p_1, p_2, \ldots p_n$.
- Supposons aussi qu'on a une liste de symptômes $S = \{s_1, s_2, \ldots s_t\}$.
- Supposons que les maladies d'un patient P peut être quantifiées selon ses symptômes $S_p = \{S_{p,s_1}, S_{p,s_2}, \ldots S_{p,s_t}\}$
- Chaque médicament a un effet sur les différents symptômes $E = \left(E_{m_i,s_j}\right)_{1 \leq i \leq n, 1 \leq j \leq t}$ avec E_{m_i,s_j} est l'effet du mécidament m_i sur le symptôme s_j.
- **Questions:** Pour les symptômes d'un patient, trouver les combinaisons de médicaments à lui donner pour le guérir avec le prix minimal (utiliser PULP et SCIPY(deux programmes demandés)).