MP rep1 205713B 朝比奈太郎　2022/05/07

(1)

ナップサック問題 => キャリーケース問題

キャリーケース問題

某航空会社のプランでは、機内持ち込み可能な手荷物を7kg以内かつ三辺の合計値が115cm以内と決められている。この条件を活かして、7kgで体積56000cm(40cm, 40cm, 35cmとした場合)で一番効率よく機内持ち込み手荷物にするにはどうすればよいか。

<定式化>

キャリーケースの容量をB, 全部でN個の荷物, 荷物I の体積をAi, 重量をCi, 荷物iをキャリーケースに入れることをXi=1, 入れないことをXi=0とすると、

max Σ(i=1 to N)Ci･Xi  
subject to Σ(i=1 to N)Ai･Xi≦B , Xi∈{0,1} (i=1,･･･,N)

輸送問題 => 新聞配達問題

新聞配達問題

某新聞会社では２種類の新聞するための工場をそれぞれ1個ずつ計2つ保有しており、6つの支店に新聞を輸送している。各工場の保有量をai, 支店jでの1日需要をbj, 工場iから支店jにかかる輸送費を１種類あたりCIjとする。1日の総輸送費を最小にするように各工場から各支店への輸送量Xijを求めよ。

<定式化>

Min Z =Σ(i=1 to 2) Σ(i=1 to 6)Cij･Xij

Subject to Σ(j=1 to 6)Xij ≦ ai, i=1,2

Σ(i=1 to 3)Xij ≧ bi, j=1,2,3,4,5

Xij ≧ 0, (i=1,2 j=1,2,3,4,5,6)

(2)

ガウスジョルダン法を採択。連立方程式を行列に変化させて解く方法である。

|  |
| --- |
| import sys  import traceback  class GaussJorden:  def \_\_init\_\_(self):  self.a = [  [ 5, -4, 6, 8],  [ 7, -6, 10, 14],  [ 4, 9, 7, 74]  ]  self.n = len(self.a)  def exec(self):  """ Solving and display """  try:  self.\_\_display\_equations()  for k in range(self.n):  p = self.a[k][k]  for j in range(k, self.n + 1):  self.a[k][j] /= p  for i in range(self.n):  if i == k:  continue  d = self.a[i][k]  for j in range(k, self.n + 1):  self.a[i][j] -= d \* self.a[k][j]  self.\_\_display\_answers()  except Exception as e:  raise  def \_\_display\_equations(self):  """ Display of source equations """  try:  for i in range(self.n):  for j in range(self.n):  print("{:+d}x{:d} ".format(self.a[i][j], j + 1), end="")  print("= {:+d}".format(self.a[i][self.n]))  except Exception as e:  raise  def \_\_display\_answers(self):  """ Display of answer """  try:  for k in range(self.n):  print("x{:d} = {:f}".format(k + 1, self.a[k][self.n]))  except Exception as e:  raise  if \_\_name\_\_ == '\_\_main\_\_':  try:  obj = GaussJorden()  obj.exec()  except Exception as e:  traceback.print\_exc()  sys.exit(1) |

実行結果

テキスト

自動的に生成された説明

(3)総当たり解法プログラム

カレンダー

自動的に生成された説明

|  |
| --- |
| import time  start = time.time()  # 物の個数  n = 8  # 要領  capacity = 25  #物の重さ、価値  size = [3, 6, 5, 4, 8, 5, 3, 4]  price = [7, 12, 9, 7, 13, 8, 4, 5]  #最高の重さと価格と最適な組み合わせを記録する  max\_size = -1  max\_price = -1  combination = []  # iには2\*\*nまでの値が入る。  for i in range(2 \*\* n):  # 変数の初期化  tmp\_size = 0  tmp\_price = 0  tmp\_combination = []  over\_flag = False  for j in range(n):  # 2進数で計算。シフトして１ビットずつ判断。  is\_put = i >> (n - j - 1) & 1  # 値を入力  tmp\_combination.append(is\_put)  tmp\_size += is\_put \* size[j]  tmp\_price += is\_put \* price[j]  #print(tmp\_combination)  # capa を越えたらフラグを立てて break  if tmp\_size > capacity:  over\_flag = True  break  # over flag が立ってない かつ 暫定 max price より高いときに更新  if (not over\_flag) and tmp\_price > max\_price:  max\_price = tmp\_price  max\_size = tmp\_size  combination = tmp\_combination  print("合計が最大になる組み合わせ")  print(combination)  print("合計価格: ", max\_price)  print("合計サイズ: ", max\_size)  end = time.time() - start  print(f"{end:.3f}s") |

実行結果

テキスト

自動的に生成された説明

(4)beta1のナップサック問題総当たり解法

カレンダー

自動的に生成された説明

(3)と同じプログラムで重量と価格、荷物の数をそれぞれbeta1の問題に対応させる。

プログラムは(3)と同じ

実行結果

黒い背景に白い文字がある

低い精度で自動的に生成された説明

(5)高速化の方法、その際の実行時間 テキスト

自動的に生成された説明

(4)までは、自分のmacにインストールしている、python3.7.6を使用していたが、高速化を図るために仮想環境を構築し、そこにpython3.9.12 64bitをインストールした。その環境で実行すると、3.7.6を使用した時と比べて約4.5s短縮することができ、10.5sとなった。