SRA の技術情報誌
GS LetterNeo
Vol.158　2021 年 9 月

最適化入門
Google OR-Tools で始めよう!
(2)

熊澤 努

DX 技術本部 先端技術研究所

はじめに

Vol. 156 では、最適化問題を解くためのツールの一つである Google OR-Tools[1]を紹介し、実際に使いながら例題を解いてみました。今回も引き続き、OR-Tools を使い、少し異なる最適化問題を解いてみましょう。なお、OR-Tools のインストールの仕方は Vol.156 を参照してください。

荷物を箱に詰める

この記事では、複数の荷物を箱に詰める問題を考えます。ただし、使用する箱の数をできる限り少なくしたいとしましょう。引っ越しをするときに、荷物を箱詰めする状況を思い浮かべてください。

[1] https://developers.google.com/optimization

Aさんは今月引っ越しをすることになっています。あれこれと準備をしているうちに、引っ越しの前夜になりました。Aさんの目の前には新居に持っていく未整理の20個の荷物があります。20個の荷物の重量(kg)を下の表に示します。

1	**2**	**2**	**3**	**4**	**6**	**6**	**7**	**7**	**7**
8	**9**	**9**	**10**	**11**	**11**	**12**	**13**	**13**	**15**

これらの荷物を一つにつき30kgまで詰めることのできる箱に詰めて、引っ越しの前の荷物整理を終えたいと思います。運び出す手間を考え、Aさんは使用する箱の数を可能な限り少なくしたいと思っています。箱は何個あればよいでしょうか。各荷物はこれ以上分割できないとし、荷物や箱の大きさについては考えないことにします。

Vol.156 の記事と同様に、上の問題を最適化問題として定式化しましょう[2]。知りたいのは必要な箱の数ですが、この値を知るためには、各箱に詰める荷物を決めなくてはいけません。どの荷物をどの箱に詰めるかはまだわからないため、この情報は未知の値です。そこで、**ある荷物がある箱に入っているかどうかを表す変数の集まり**を二次元配列（行列）$(packings_{ij})$で記述します。

$$packings_{ij} = \begin{cases} 1, & \text{荷物}i\text{が箱}j\text{に入っている} \\ 0, & \text{荷物}i\text{が箱}j\text{に入っていない} \end{cases}$$

使用する箱の数を計算するために、**ある箱が使用されているかどうかを表す変数の集まり**を配列$(used_j)$で記述します。

$$used_j = \begin{cases} 1, & \text{箱}j\text{に荷物が 1 つ以上入っている} \\ 0, & \text{箱}j\text{に荷物が 1 つも入っていない} \end{cases}$$

この配列のサイズを必要な箱の最大数とすれば、要素として含まれる 1 の個数を数える、つまり、この配列の要素の和を求めることで、実際に使う箱の数を計算できます。箱の数が最大となるのは各箱に荷物を 1 つだけ入れた場合なので、20 個です。そのため、行列$(packings_{ij})$のサイズは20×20、配列$(used_j)$のサイズは20となります。

問題の制約条件は次の 2 種類です。

- ✓ **荷物の制約条件**: 各荷物を入れることのできる箱は 1 つしかありません。このことを制約条件として記述します。各荷物 i に対して、

$$packings_{i0} + packings_{i1} + \cdots + packings_{i19} = \sum_{j=0}^{19} packings_{ij} = 1$$

[2] 今回とりあげた箱に荷物を積める問題は、OR-Tools の公式サイトでも解説されています。
https://developers.google.com/optimization/bin/bin_packing

- ✓ **重量に関する制約条件**: 各箱には 30kg まで荷物を詰めることができるという条件を記述します。各荷物の重量を配列$(item_i)$で表すことにすると（$item_i$は荷物 i の重量で、例えば $item_0 = 1, item_1 = 2$）、$item_0 \times packings_{0j}$で荷物 0 を箱 j に入れたときの荷物の重量を表すことができます。各箱 j に対して、

$$1 \times packings_{0j} + 2 \times packings_{1j} + \cdots + 15 \times packings_{19j}$$
$$= item_0 \times packings_{0j} + item_1 \times packings_{1j} + \cdots + item_{19} \times packings_{19j}$$
$$= \sum_{j=0}^{19} item_i \times packings_{ij} \leq 30$$

上の制約式を少し書き換えることで、使わない箱について記述することができます。箱を使わない場合は 0 kg までしか詰められないと考えると、配列$(used_j)$の要素を使って以下のように書くことができます。

$$\sum_{j=0}^{19} item_i \times packings_{ij} \leq 30 \times used_j$$

最後に、最小化したい箱の数を式で次のように表します。最小化する対象のことを目的関数といいます。

$$\text{min. } used_0 + used_1 + \cdots + used_{19} = \sum_{j=0}^{19} used_j$$

min.は最小化するという意味です。

以上をすべてまとめると、引っ越しの箱詰めの問題は以下のように表すことができます。

- ✓ **目的関数:**

$$\textbf{min.} \sum_{j=0}^{19} \boldsymbol{used_j}$$

- ✓ **制約条件:**

$$\sum_{j=0}^{19} \boldsymbol{packings_{ij}} = \mathbf{1} \quad (\mathbf{0} \leq \boldsymbol{j} \leq \mathbf{19}),$$

$$\sum_{j=0}^{19} \boldsymbol{item_i \times packings_{ij}} \leq \mathbf{30} \times \boldsymbol{used_j} \quad (\mathbf{0} \leq \boldsymbol{i} \leq \mathbf{19})$$

$$\boldsymbol{packing_{ij}} \textbf{ は0または1 } (\mathbf{0} \leq \boldsymbol{i, j} \leq \mathbf{19}),$$

$$\boldsymbol{used_j} \textbf{ は0または1 } (\mathbf{0} \leq \boldsymbol{j} \leq \mathbf{19}).$$

後は、OR-Tools を使ってこの問題を解く Python プログラムを書くだけです。次の pack.py を見てください。

pack.py

```
from ortools.linear_solver import pywraplp

# 荷物の重量
items = [1, 2, 2, 3, 4, 6, 6, 7, 7, 7, 8, 9, 9, 10, 11, 11, 12, 13, 13, 15]

# 箱の最大数（20個）
max_packs = len(items)

# (1) 問題を解くソルバーの生成
solver = pywraplp.Solver.CreateSolver('SCIP')

# (2) 未知変数の定義
# 荷物iを箱jに詰めるならpackings[i][j] = 1 そうでなければ 0
packings = [[solver.BoolVar(f'{i} in {j}') for i in range(len(items))]
for j in range(max_packs) ]

# 箱jを使用するならused[j] = 1 そうでなければ 0
used = [solver.BoolVar(f'{j} is used') for j in range(max_packs)]

# (3) 制約条件の記述
# 各荷物は一つの箱に詰める
for i in range(len(items)):
    solver.Add(sum(packings[i][j] for j in range(max_packs)) == 1)

# 各箱に詰める荷物の重量の合計は30kg以内
for j in range(max_packs):
    solver.Add(sum(items[i]*packings[i][j] for i in range(len(items))) <=
30*used[j])

# (4) 目的関数の記述
# 使用する箱の個数を最小化
solver.Minimize(solver.Sum(used))

# (5) 解を求める
status = solver.Solve()

# (6) 結果の出力
if status == pywraplp.Solver.OPTIMAL:
    result = f'荷物の個数: {round(solver.Objective().Value())}個'
else:
    result = '最適な荷物の個数: なし'
print(result)
```

今回の箱詰めの問題で使用する荷物の重量の一覧と箱の最大数（20 個）を、それぞれ変数 items と max_packs に格納しています。以下、コメントにつけた番号順にプログラムを説明します。

(1) **問題を解くソルバーの生成**: 問題を解くソルバーオブジェクトを生成します。Vol.156 のときと同じように、OR-Tools に組み込まれている整数計画問題を解くソルバーを使っています。
(2) **未知変数の定義**: 値を求めたい変数リスト packings と used を生成します。それぞれの要素は 0 または 1 の値をとる 2 値変数なので、ソルバーの BoolVar メソッドにより、各変数を Bool 型としています。BoolVar の引数は変数につける名前です。
(3) **制約条件の記述**: 荷物の制約条件と重量に関する制約条件をソルバーに追加します。
(4) **目的関数の記述**: 最小化する目的関数である使用する箱の数の算出式をソルバーに追加します。今回は最小化を実行するため、`Minimize` メソッドを呼び出します。
(5) **解を求める**: packings と used の各値を求めるため、ソルバーの `Solve` メソッドを呼びだします。変数 status には、最適化を実行できたかどうかを表す情報が格納されます。
(6) **結果の出力**: 必要な箱の数の最小値を求めることに成功した場合(`pywraplp.Solver.OPTIMAL`)、箱の数を出力します。目的関数の値は浮動小数点数のため、Python の組込み関数 round を使って整数値にしています。

`pack.py` を実行すると、下のように必要な箱の数が 6 個であることが分かります。各変数の値を出力することで、どの荷物をどの箱に詰めるかを知ることもできますが、ここでは割愛します。

```
荷物の個数: 6個
```

おわりに

今回は、Google OR-Tools を使って引っ越しの際の荷物の箱詰めの問題を解いてみました。
身近な問題を OR-Tools で解くことができることを実感していただけたら幸いです。

GSLetterNeo Vol.158
2021 年 9 月 20 日発行
発行者　株式会社 SRA 先端技術研究所

編集者　熊澤努　方 学芬
バックナンバー http://www.sra.co.jp/gsletter
お問い合わせ　gsneo@sra.co.jp

株式会社SRA
〒171-8513 東京都豊島区南池袋 2-32-8

夢を。Yawaraka Innovation
やわらかいのべーしょん