SRA の技術情報誌
GS LetterNeo
Vol.156　2021 年 7 月

最適化入門

Google OR-Tools で始めよう!

熊澤 努

DX 技術本部 先端技術研究所

はじめに

Google OR-Tools[1]は最適化という分野の問題を解くオープンソースソフトウェアです。最適化問題は、与えられた制約の下で最適な値を求める問題のことで、スケジューリングや仕事の割り当ての問題などに広く応用されています。最適化問題は解くことが難しいことが多いですが、そのような難しい問題も OR-Tools が自動的に解いてくれます。OR-Tools は Python、C++、Java、C#で使うことができます。今回は、Python で OR-Tools を使い、最適化の世界を体験してみます。

OR-Tools をインストールする

OR-Tools は pip でインストールすることができます。Python 環境上で次のコマンドを実行しましょう。

```
> pip install ortools
```

今回の記事では、執筆時点で最新のバージョン 9.0.9048 を使います。

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

最適化問題を解いてみる

最適化とは、複雑な制約の下で最良の解を求める技術です。最適化の例として、次の問題を OR-Tools を使って解いてみます(この問題は筆者が今回の記事のために考えたものです)。

ある会社で進行中の開発プロジェクトが、このままでは納期に間に合わない見込みであることが分かりました。そこで、プロジェクトマネージャはA、B、Cの3名の技術者にアルバイトとして急遽参加してもらうことにしました。Aさん、Bさん、Cさんは熟練度が異なり、1時間当たりに書くプログラムがそれぞれ、2000行、1850行、1750行です。これを各人の生産性の指標として、プロジェクトマネージャは、3人の時給をそれぞれ1500円、1200円、1150円としました。

プロジェクトには次のような遵守しなくてはならない制約条件があります。

- ✓ **予算の関係上、納品までに3人に支払う給料の合計は10万円以内でなくてはならない。**
- ✓ **3人は1時間単位で勤務する。例えば、0.1時間(6分)や0.5時間(30分)働くということは認めない。**
- ✓ **納品まで就業可能な時間は40時間だが、それぞれに事情があり、プロジェクトにフルタイムで参加できるのはCさんだけである。AさんとBさんは30時間までなら参加できる。**
- ✓ **3人の就業時間の偏りを少なくするため、Aさんの就業時間はBさんの2倍以内、Bさんの就業時間はCさんの2倍以内、さらに、Cさんの就業時間はAさんの1.5倍以内に抑える。**

納期に間に合わせるためには、3人の生産量（3人が書くプログラムの行数の合計）が最大になるように3人の就業時間を決めなくてはいけません。3人の就業時間をどのように定めればよいでしょうか。

最初に、上の問題を最適化問題として定式化する必要があります。知りたいのは A さん、B さん、C さんがプロジェクトで働く時間なので、それぞれx時間、y時間、z時間とおきます。問題文の制約条件をx、y、zの式で書き表していきます。

- ✓ **給料の制約条件**: 3 人の時給から、実際に支払う給料の合計は$1500x + 1200y + 1150z$となるので、10 万円以下という制約は次の不等式で表すことができます。

$$1500x + 1200y + 1150z \leq 100000$$

- ✓ **時間の制約条件**: 3 人は 1 時間単位で働くことから、x、y、zは整数値しか取りません。このことを$x, y, z \in \mathbb{Z}$と書きます。A さんと B さんは 30 時間まで、C さんは納期まで 40 時間まで働くことができることは次の不等式で表します。

$$0 \leq x, y \leq 30$$
$$0 \leq z \leq 40$$

A さんの就業時間が B さんの 2 倍以内という条件を不等式で書くと次のようになります。

$$x \leq 2y$$

同様に、B さんと C さんの就業時間の偏りの条件は次の通り書き表すことができます。

$$y \leq 2z$$
$$z \leq 1.5x$$

次に、最大化したい生産量を式で書きます。この式はx, y, zの関数になることから、目的関数といいます。A さんがx時間働く場合の生産量は$2000x$となります。B さん、C さんの生産量もそれぞれ$1850y, 1750z$と書けるので、3 人の生産量の合計は、次の通りになります。max.は最大化するという意味でつけた記号です。

$$\max. 2000x + 1850y + 1750z$$

以上をまとめると、最適化問題は次のように数式で書くことができます。

- ✓ **目的関数:**

$$\boldsymbol{\max. 2000x + 1850y + 1750z}$$

- ✓ **制約条件:**

$$\boldsymbol{1500x + 1200y + 1150z \leq 100000}$$
$$\boldsymbol{0 \leq x, y \leq 30}$$
$$\boldsymbol{0 \leq z \leq 40}$$
$$\boldsymbol{x \leq 2y}$$
$$\boldsymbol{y \leq 2z}$$
$$\boldsymbol{z \leq 1.5x}$$
$$\boldsymbol{x, y, z \in \mathbb{Z}}$$

ここまでで、x, y, zを求めるプログラムを書く準備が整いました。OR-Tools を使いながら、上の数式を一つずつ書き下したプログラムを書いていきます。なお、上の問題は変数が整数の値しか取らないため、整数計画問題と呼ばれています。一般に整数計画問題を解くのは簡単ではありません。今回は、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います。

解を求めるプログラムが次に示す `IP_Ex.py` です。

IP_Ex.py

```
from ortools.linear_solver import pywraplp

# (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
# Aさんの就業時間
x = solver.IntVar(0, 30, 'x')
# Bさんの就業時間
y = solver.IntVar(0, 30, 'y')
# Cさんの就業時間
z = solver.IntVar(0, 40, 'z')

# (3) 制約条件の記述
# 給料の制約条件
solver.Add(1500 * x + 1250 * y + 1150 * z <= 100000)

# 時間の制約条件
solver.Add(x <= 2 * y)
solver.Add(y <= 2 * z)
solver.Add(z <= 1.5 * x)

# (4) 目的関数の記述
# 生産性を最大化
solver.Maximize(2000 * x + 1850 * y + 1750 * z)

# (5) 解を求める
status = solver.Solve()

# (6) 結果の出力
if status == pywraplp.Solver.OPTIMAL:
    result = f'''最適の労働時間: あり
Aさんの労働時間: {x.solution_value()}時間
Bさんの労働時間: {y.solution_value()}時間'
Cさんの労働時間: {z.solution_value()}時間'
最大生産量: {solver.Objective().Value()}行'''
else:
    result = '最適の労働時間: なし'
print(result)
```

プログラムの重要な箇所には、番号を付けたコメントを付けてあります。番号の順に詳しく見ていきましょう。

(1) まず、問題を解くソルバーオブジェクトを生成します。文字列'SCIP'は実際に使うソルバーの名称です。SCIP[2]は整数計画問題を含む広い範囲の問題を解くことができます。OR-Tools に組み込まれているので、改めてインストールする必要はありませ

[2] https://www.scipopt.org/

ん。生成したソルバーオブジェクトに未知の変数、制約式、目的関数を追加していきます。

(2) 値を求めたい変数x, y, zを生成します。各変数は整数値なので、整数値変数を生成するメソッド `IntVar` を呼び出しています。`IntVar` には、第 1 引数から順に取りうる値の下限、上限、変数名を指定します。これによって、例えば、変数xを「$0 \leq x \leq 30$であるような整数変数」とすることができます。

(3) 制約条件をソルバーに追加します。OR-Tools では、上で説明した給料と時間の制約条件式をそのまま Python プログラムで書くことができます。

(4) 目的関数をソルバーに追加します。目的関数も数式を Python プログラムでそのまま書くことができます。今回の記事では生産量を最大化するので、`Maximize` メソッドを呼び出します。最小化する場合には `Minimize` メソッドを呼びます。

(5) ソルバーの `Solve` メソッドを呼んで解を求めます。このメソッドの返り値は解を求めるのに成功したかどうかといったステータス情報です。解は常に存在するとは限らない点に注意してください。例えば、上の給料の制約条件に加えて「給料の合計は 15 万円以上とする」という矛盾した制約条件を追加した場合には、解は存在しません。

(6) 結果を出力します。ステータスが `pywraplp.Solver.OPTIMAL` とは、3 人の最適な労働時間を計算することができたことを意味しています（最適解といいます）。その場合には、`solution_value` メソッドで各変数の値を取得して表示します。さらに、最適解のときの目的関数の値、つまり生産量も表示しています。最適解を求めることができなかった場合には、各変数の値を取得できません。そのまま終了します。

`IP_Ex.py` を実行すると、以下のように最適解を求めることができます。A さんが 21 時間、B さんが 29 時間、C さんが 28 時間プロジェクトで働けば最大の生産量が得られることが分かりました。

```
最適の労働時間：あり
Aさんの労働時間：21.0時間
Bさんの労働時間：29.0時間
Cさんの労働時間：28.0時間
最大生産量：144650.0行
```

おわりに

今回は、Google OR-Tools を使って整数計画問題という最適化問題を解いてみました。OR-Tools には、他に巡回セールスマン問題などのコンピュータを使っても解くのが難しいとされている問題を解くライブラリが準備されています。最適化問題を解くには、最初に人間が定式化する必要があり、特に現実の複雑な問題では難しいことも多いです。しかし、一

旦数式で問題を記述してしまえば、このようなツールで自動的に解くことが可能です。ぜひ試してみてください。

GSLetterNeo Vol.156
2021 年 7 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