SRA の技術情報誌
GSLetterNeo
Vol.167　2022 年 6 月

最適化入門
inspyred ではじめる群知能(2)

熊澤 努

株式会社 SRA 技術本部 先端技術研究室
株式会社 SRA ホールディングス 先端技術研究所

はじめに

前回(Vol.166[1])に引き続き、inspyred を使った進化計算を Python で体験します。前回は出張での訪問順序を決定する問題を解きました。今回はアリコロニー最適化法以外の進化計算技法である遺伝的アルゴリズムと分布推定アルゴリズムを試してみます。また、異なる考え方で作られた問題解決技法である焼きなまし法も使ってみましょう。

[1] https://www.sra.co.jp/Portals/0/files/gsletter/pdf/GSLetterNeoVol166.pdf (2022 年 6 月 16 日閲覧)

お土産を持ち帰ることができるか?

今回は、出張先で起こった問題を考えてみることにします。

あなたはある会社の営業担当社員として、47都道府県を巡る出張に出ています。予定していたすべての訪問先を無事に訪れ、最終日の今日は会社の同僚に渡すお土産を選んでいます。地元のお土産にすっかり夢中になってしまったあなたは、ついつい買いすぎてしまったようです。滞在しているホテルで、用意しておいた30kgまで詰め込めるスーツケースにお土産を詰め込んでみると、困ったことに入りきらない恐れが出てきました。重量の許す限りスーツケースに入れて持ち帰ることにするには、どのお土産をスーツケースに入れればよいでしょうか。詰め込めないお土産は配送することにします。

お土産は A から J までの 10 点あり、それぞれの重量は以下の表のとおりとします。

A	B	C	D	E	F	G	H	I	J
2kg	2kg	3kg	3kg	6kg	9kg	11kg	12kg	13kg	14kg

問題を扱いやすくするため、今回も少し簡単な仮定を導入します。

- ✓ スーツケースの制限は重量(30kg 以内)だけとする。スーツケースの容積やお土産の体積、配送コストなどの他の制限は考えない。
- ✓ スーツケースには、重量制限を超えない範囲で詰め込み可能な最大重量を詰め込むこととする。

今回取り上げる、制限重量以下に収まるように荷物を詰め込む問題はナップザック問題と呼ばれています。前回の巡回セールスマン問題と同様、ナップザック問題もコンピュータで解くことが難しい問題で、一般にしらみつぶしに解答を探す必要があります。今回の問題は簡単ですが、それでも$2^{10} = 1024$通りの中から最適なお土産の組合せを見つけ出さなければなりません。

この問題を、inspyred を使って三通りの方法で解いてみましょう。筆者の計算機環境は前回の記事と同じ Python 3.6、inspyred 1.0.1 です。

遺伝的アルゴリズムで解く

遺伝的アルゴリズム(GA)は代表的な進化計算技術の一つです。生物集団が世代を重ねるにつれて、環境に適応した遺伝子を獲得していく過程をヒントに作られました。GA にはたくさんのバリエーションが存在し、問題に応じて適切なものを選択する必要があります。今回は inspyred に実装されている GA をそのまま使ってみることにします。

お土産は、以下のように重量のリストで与えられているものとします。

```
item_table = [(2, 2), (2, 2), (3, 3), (3, 3), (6, 6), (9, 9), (11, 11),
              (12, 12), (13, 13), (14, 14)]
```

ここで、重量がタプル(組)になっているのは、inspyred では、制約事項である各お土産の重量と、制約の範囲内で最大化したい価値などの値を分けて書くことができるようなっているためです。ここでは価値もまた重量なので、両者を同じ値にします。

前回と同様に、プログラム中のコメントにつけた番号順に簡単に説明します。

kp_ga.py

```
import inspyred
import random
import time

# お土産詰め込み問題を遺伝的アルゴリズムで解く
def kp_ga(capacity, item_table):

    # (1) お土産詰め込み問題の生成
    problem = inspyred.benchmarks.Knapsack(capacity, item_table)

    # (2) 遺伝的アルゴリズム(GA)のソルバーの生成
    solver = inspyred.ec.GA(random.Random(time.time()))
    solver.terminator = inspyred.ec.terminators.generation_termination

    start_time = time.time()

    # (3) GAの実行
    solver.evolve(
        generator=problem.generator,
        evaluator=problem.evaluator,
        bounder=problem.bounder,
        maximize=problem.maximize,
        pop_size=100,
        max_generations=10,
        num_selected=5)

    elapsed_time = time.time() - start_time

    best = max(solver.archive)

    # (4) 最良解の出力
    print('********************')
    print('荷物の詰め込み:')
    print(best.candidate)
    print(f'重さ: {best.fitness} kg')
    print(f'実行時間: {elapsed_time} sec')
    print('********************')
```

(1) inspyred で用意されている問題であるナップザック問題を生成します。変数 capacity はスーツケースに詰めることのできる最大重量で、ここでは 30 [kg]です。

(2) GA を使って問題を解くソルバーを生成します。`solver.terminator` は GA の終了条件です。ここでは、一定世代子孫を生成したら終了するようにしています。

(3) GA を実行して最適解を計算します。evolve メソッドの引数の内、pop_size は一世代あたりの遺伝子の数、max_generations は世代数、num_selected は次の世代に残す遺伝子の数です。GA の他の設定値はすべてディフォルト値にしています。

(4) GA が求めた詰め込み結果と、その時のスーツケース内のお土産の総重量、実行時間を出力します。

筆者の PC で関数 kp_ga を実行すると、下のような結果が得られました。

```
荷物の詰め込み:
[0, 1, 1, 1, 0, 1, 0, 1, 0, 0]
重さ: 29 kg
実行時間: 0.008977413177490234 sec
```

最初のリストは発見した解で、スーツケースに入れて持ち帰るお土産が 1 、発送するお土産が 0 になっています。つまり、持ち帰るお土産は B、C、D、F、H の五点です。それらの総重量が 29kg となりました。

分布推定アルゴリズムで解く

分布推定アルゴリズム(EDA)は、GA と同様に集団を用いた進化計算の一種です。ここでは、Inspyred に実装されている EDA を使って問題を解いてみます。以下のように、お土産詰め込み問題を EDA で解くプログラムは GA の場合とほぼ同様に書くことができます。

kp_eda.py

```
import inspyred
import random
import time

# お土産詰め込み問題を分布推定アルゴリズムで解く
def kp_eda(capacity, item_table):

    problem = inspyred.benchmarks.Knapsack(capacity, item_table)

    # (1) 分布推定アルゴリズム(EDA)のソルバーの生成
    solver = inspyred.ec.EDA(random.Random(time.time()))
    solver.terminator = inspyred.ec.terminators.evaluation_termination

    start_time = time.time()

    # (2) EDAの実行
    solver.evolve(
        generator=problem.generator,
        evaluator=problem.evaluator,
        bounder=problem.bounder,
        maximize=problem.maximize,
        pop_size=100,
        max_evaluations=100)

    elapsed_time = time.time() - start_time

    best = max(solver.archive)

    print('********************')
    print('荷物の詰め込み:')
    print(best.candidate)
    print(f'重さ: {best.fitness} kg')
    print(f'実行時間: {elapsed_time} sec')
    print('********************')
```

(1) EDA を実行するソルバーを生成します。GA とは異なる終了条件を採用しており、一定回数計算を実行したら終了することにします。

(2) EDA を実行します。変数 pop_size は集団の大きさ、つまり、一度に生成する解の個数、max_evaluations は計算の実行回数です。

EDA では以下の解が得られました。

```
荷物の詰め込み:
[1, 1, 0, 1, 0, 1, 0, 0, 1, 0]
重さ: 29 kg
実行時間: 0.013962507247924805 sec
```

GA の時とは異なる組合せのお土産が選ばれていることが分かります。A、B、D、F、I の五点、合計 29kg をスーツケースで持ち帰ることになります。

焼きなまし法で解く

最後は焼きなまし法（シミュレーテッド・アニーリング、SA）です。SA は GA や EDA とは異なり集団を活用せず、一つの解を逐次更新して最適な解に近づけていく方法です。Inspyred には SA も用意されているので、お土産の選択に使ってみます。

kp_sa.py

```python
import inspyred
import random
import time

# お土産詰め込み問題を焼きなまし法で解く
def kp_sa(capacity, item_table):

    problem = inspyred.benchmarks.Knapsack(capacity, item_table)

    # (1) 焼きなまし法(SA)のソルバーの生成
    solver = inspyred.ec.SA(random.Random(time.time()))
    solver.terminator = inspyred.ec.terminators.average_fitness_termination

    start_time = time.time()

    # (2) SAの実行
    solver.evolve(
        generator=problem.generator,
        evaluator=problem.evaluator,
        bounder=problem.bounder,
        maximize=problem.maximize,
        temparature=1000)

    elapsed_time = time.time() - start_time

    best = max(solver.archive)

    print('********************')
    print('荷物の詰め込み:')
    print(best.candidate)
    print(f'重さ: {best.fitness} kg')
    print(f'実行時間: {elapsed_time} sec')
    print('********************')
```

(1) 焼きなまし法のソルバーを生成しています。ここでの終了条件は、「生成される解が十分に最良解に近くなったら終了する」、とします。これは、解の改善の見込みがなくなるまでは解の生成を続けるということです。

(2) SA では変数 temperature(温度)の初期値を指定する必要があります。この変数は解の更新の仕方に影響を与えます。今回取り上げたような難しい問題では、少し高めの値に設定しておきます。

SA で求めた解を以下に示します。

```
荷物の詰め込み:
[0, 1, 0, 0, 0, 0, 0, 1, 0, 1]
重さ: 28 kg
実行時間: 0.0029947757720947266 sec
```

スーツケースで持ち帰るお土産は B、H、J の三点です。総重量は 28kg と GA や EDA より少ない値になりました。今回の出張では、総重量が 29kg となっている GA または EDA の結果を採用することにしましょう。

おわりに

お土産の問題は、前回取り上げたアリコロニー最適化法でも解くことができます。このように、進化計算は様々な問題解決に使える汎用性の高い技法です。なお、多くの進化計算技術には、ユーザが設定するパラメータが定められています（SA における温度の初期値など）。パラメータの値によってアルゴリズムの挙動に変化が出るため、適切な値を探るチューニングを事前に行うなど、実行を制御するための事前プロセスが必要です。パラメータについては、記事で詳しく扱うことができなかったので、多くを inspyred のディフォルト値としました。進化計算を試してみる際には、パラメータの値にも着目していただければと思います。

GSLetterNeo Vol.167
2022 年 6 月 20 日発行
発行者　株式会社 SRA 技術本部 先端技術研究室

編集者　熊澤努　方 学芳
バックナンバー https://www.sra.co.jp/public/sra/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