組合せ最適化を体系的に知って Pythonで実行してみよう

(株) 構造計画研究所 斉藤努

1 はじめに

これから、誰でも**組合せ最適化**が使えるようにご案内したいと思います。

組合せ最適化を使うコツは、全体像を理解することです。どんな要素があるのか、それらがどのような関係にあるのかを知ることにより、解きたい問題についてアプローチできるようになります。

Python を使うことにより、簡単に、様々な最適化ができます。実際のコードを見ながら、あとで確認しましょう。最初に、身近な最適化の例から見ていきます。

> 先日、あなたが実家に帰ると、お土産に野菜を持って帰るよう言われました。東京の野菜は高いので、「儲かった」と喜びましたが、量が多すぎます。せいぜい 5kg しか持って帰れないとします。(宅配便は使えないことにしてください) また、野菜は切ったりすると傷むので、そのまま持って帰ることにします。

あなたは「1kg あたりの販売価格の高いものから選んでいこう」と考えました。実は、これは (そこそこいい方法ですが) 最適な方法ではありません。本当に最適な方法は、どうすればわかるでしょうか？

数理最適化を使えば、このような問題を解くことができます。

数理最適化では、数式を使って、問題を**数理モデル**で表します。先ほどの例は、

最大化	$\sum_i p_i x_i$	p_i:販売価格
	$\sum_i w_i x_i \leq 5$	w_i:重さ
	$\forall x_i \in \{0,1\}$	x_i:持って帰るかどうか

という数理モデルになります。

数理最適化は、**連続最適化**と**組合せ最適化**に分けられます。

連続最適化:	連続要素のみ
組合せ最適化:	連続以外の離散要素を含む

数理最適化 (組合せ最適化と連続最適化) 問題とは、数理モデルで表すことができる問題になります。

今回は、組合せ最適化に焦点を当ててご説明します。問題を数理モデルを表すことを**定式化**するといいます。

2 定式化

レッスン 1

組合せ最適化では、数理モデルを定式化する。

定式化をするには、3 つの要素を決める必要があります。

1. 何を決めたいのか？　「持って帰る野菜を決めたいです」
2. どうなるとうれしいのか？　「持って帰る野菜の販売価格の合計が高くなるとうれしいです」
3. 守らないといけないことは？　「持って帰る野菜を 5kg 以下にします」

この 3 つをそれぞれ、**変数**、**目的関数**、**制約条件**とよびます。先ほどの例で見てみましょう。

$$
\begin{array}{ll}
\text{目的関数:} & \sum_i p_i x_i \qquad \rightarrow \text{最大} \\
\text{制約条件:} & \sum_i w_i x_i \leq 5 \\
\text{変数:} & \forall x_i \in \{0, 1\}
\end{array}
$$

定式化できるようにするためには、慣れが必要です。今回は、いくつかの例を後で見ることにします。詳しく勉強する場合は、書籍「今日から使える!組合せ最適化」などを参考にしてください。

数理モデルから最適な答えを探すのは、(賢い人が作ってくれた) ツールを使います。このツールのことを最適化ソルバー、略して**ソルバー**とよびます。つまり、問題を（誰でもわかるように）表すだけで、ソルバーを使えば解 (答えのこと) を出すことができます。

最近では、このソルバーが使いやすく、性能がたいへんよくなってきました。みんなが、最適化が試せるようになってきているのです。

3 標準問題

先ほどの例の他に、どんな例があるでしょうか？あなたは実家に帰るときに、web で乗り換え駅を探索したかもしれません。

あなたの家の最寄り駅から、実家の最寄り駅までの最短路 (あるいは最安路) を探したい。

これも最適化問題です。最適化問題は、いろいろなところにたくさんあります。でも、問題ごとに定式化するのは、大変ですよね。実は、別々の問題でも、同じ定式化になることがよくあります。そうすると、わざわざ定式化する必要がなくなり、便利ですね。世の中によくある問題を**標準問題**とよぶことにします。

レッスン 2

標準問題を理解する。

関連する標準問題を集めて、標準問題クラスという枠組みを作ります。標準問題クラスは、7 つあります。標準問題は、よく使われるものを厳選しました。

野菜の選び方は**ナップサック問題**、乗り換え駅探索は、**最短路問題**といいます。標準問題は、よく研究もされているので、多くの場合、効率的な解法があります。あるいは、定式化がされているので、すぐ解くことができます。あとで、やってみましょう。ここで、あげている全ての標準問題の実行例は、次のサイトをご覧ください。

http://

4 数理問題

最近、私がやっているコンテナの仕事のお話しをします。

世界中の人たちが、いろいろなものを安く買えるのはコンテナ輸送のおかげです。中国などで生産したものを日本やアメリカやヨーロッパに、大量に安く運べるからです。でも、空のコンテナが、どんどんたまります。また中国に戻さないといけません。いつ、どこからどこに戻すかを決めるのが、**最小費用流問題**になります。

ところが、最小費用流問題で表せない制約条件もあります。1 つが、カボタージュとよばれるものです。カボタージュというのは、国内のみの輸送を自国業者のみに規制することです。

標準問題で対応できない場合は、数理モデルをそのまま扱います。数理モデルで表された問題を**数理問題**とよびます。標準問題は、数理問題として見ることもできます。数理問題は、標準問題より汎用的な分類になります。

標準問題クラス	標準問題
グラフ・ネットワーク問題	最小全域木問題
	最大安定集合問題
	最大カット問題
	最小頂点被覆問題
	最短路問題
	最大流問題
	最小費用流問題
経路問題	運搬経路問題
	巡回セールスマン問題
集合被覆・分割問題	集合被覆問題
	集合分割問題
スケジューリング問題	ジョブショップ問題
	勤務スケジューリング問題
切出し・詰込み問題	ナップサック問題
	ビンパッキング問題
	n 次元パッキング問題
配置問題	施設配置問題
	容量制約なし施設配置問題
割当・マッチング問題	2 次割当問題
	一般化割当問題
	最大マッチング問題
	重みマッチング問題
	安定マッチング問題

レッスン 3

数理問題を理解する。

数理問題には、何があるのでしょうか？

数理問題の違いは、変数や目的関数や制約の違いによります。数理問題が異なると、解法 (アルゴリズム) が異なります。この解法の種類によって、解きやすさが全く違います。また、数理問題ごとにソルバーも異なることがあります。なるべく、簡単に解ける数理問題に持って行けるかが、重要になります。数理問題は、親子関係になります。親から見ていきましょう。

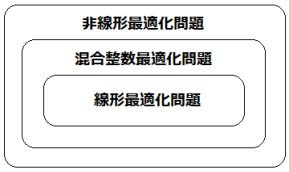

図 1: 数理問題

全ての最適化問題は、非線形最適化問題です。なぜ、わざわざ非線形最適化というのでしょうか？それは、線形最適化問題を非線形最適化として扱うのは非効率的だからです。非線形最適化という言葉には、非線形な目的関数か非線形な制約条件があることを暗黙に示唆しています。

線形最適化問題

数理問題で最も解きやすいのは、線形最適化問題です。最小費用流問題も線形最適化問題です。変数が**連続変数**で、目的関数と制約条件が線形 (1 次式) で表される問題です。これは、かなり大きな問題でも解けるようになってきました。

混合整数最適化問題

目的関数と制約条件は線形ですが、**離散変数**も許した問題を混合整数最適化問題とよびます。ナップサック問題や最短路問題も混合整数最適化問題です。

実務でよく現れる問題で、私の扱っている問題の多くは、これになります。難しい問題ですが、最近では、定式化や問題の規模次第で解けるようになってきました。

「混合整数最適化問題が解けるようになってきた」これこそが、最適化のしきいが下がってきた要因といえるでしょう。数理最適化初心者にとって、1 つの目標が、簡単な混合整数最適化問題の定式化することになるでしょう。

ただし、実務で、混合整数最適化問題を扱うのは難しく、いろいろ工夫が必要になることもあります。

非線形最適化問題

非線形最適化は、さらに難しい問題です。しかし、小規模であれば、現状でも解けるようになってきました。また、非線形最適化の中でも非線形凸最適化は、大規模なものでも扱えますが、ここでは省略します。

この 3 種類の例は後ほど見てみます。

5　標準問題と数理問題の関係

全ての数理最適化問題は、いずれかの数理問題になります。数理問題は、組合せ最適化問題の大分類と見ることができます。しかし、この分類はおおざっぱ過ぎます。生物の分類に例えると、脊椎動物や無脊椎動物のようなものです。

標準問題は、小分類と見ることができます。生物で言うと、は虫類やほ乳類でしょうか。標準問題の方が身近に感じることができますので、覚えやすいでしょう。

図 2: 標準問題と数理問題の構造

- 1 つの標準問題クラスでも、別々の数理問題を含むことがあります。
- 定式化は生物でいう DNA のような設計図といえます。しかし、ある生物の DNA は 1 つだけですが、1 つの問題を別々の問題としてとらえることができます。

- 数理問題や標準問題へのとらえ方によって、ソルバー (すなわち解法) が変わります。
- 一般的に標準問題のソルバーの方が数理問題のソルバーより効率がよいです。

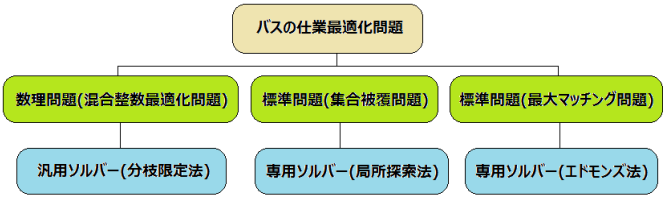

図 3: 標準問題や数理問題へのとらえ方の例

数理問題について補足します。生物における脊椎動物と無脊椎動物は、別々のグループですが、数理問題の分類では、基本的に包含関係になっています。すなわち、全ての問題は非線形最適化問題になっていて、その中に混合整数最適化問題があり、さらにその中に線形最適化問題があります。つまり、線形最適化問題は、混合整数最適化問題でもあり、非線形最適化問題でもあります。また、混合整数最適化問題は、非線形最適化問題でもあります。このことは、ソルバーを適用する際にもいえます。すなわち、非線形最適化ソルバーは、非線形最適化問題、混合整数最適化問題、線形最適化問題を全て扱えます。混合整数最適化ソルバーは、一般の非線形最適化問題は扱えず、混合整数最適化問題、線形最適化問題を扱えます。線形最適化ソルバーは、線形最適化問題のみ扱えます。習慣的に、混合整数最適化ソルバーと線形最適化ソルバーは、1 つのソルバーになっていることがよくあります。

非線形最適化ソルバーは、制約を考慮できるものとできないものがありますが、実務者にとっては、制約を考慮できるソルバーだけ気にしていれば十分と思われるので、ここでは制約を考慮できることを前提にしています。

ソルバーは、より小さい領域に対する方が性能がよくなります。線形最適化問題は、線形最適化ソルバーを利用する方が、早く最適解が得られます。線形最適化問題に対し、非線形最適化ソルバーを用いると、「初期解を指定する必要がある」「計算時間が膨大にかかる」などのデメリットがあります。

混合整数最適化問題は、混合整数線形最適化問題ともよびます。しかし、「整数」が入っているので非線形です。しかし、整数を除いて考えれば線形なので、混合整数線形最適化問題とよびます。

最適化の研究者の間では、はるかに細かい分類がありますが、多くの実務者にとっては「非線形最適化」「混合整数最適化」「線形最適化」の 3 種類で十分でしょう。

研究者は、線形最適化を LP(Linear Programing)、混合整数最適化を MIP もしくは MILP(Mixed Integer Linear Programing)、非線形最適化を NLP(Non Linear Programing) とよびます。Programing は、もともと「計画」と訳されていましたが、最近では最適化とよぶようになってきましたので、ここでも最適化で統一しています。最適化は Optimization なので、今後はよびかたも変わるかもしれません。

6　ビジネスでの実例

標準問題と数理問題を知ることが、組合せ最適化を使う上で重要になります。

ビジネスにおける最適化問題も、標準問題としてとらえることができます。標準問題に落とし込むことで、ビジネスの関係者間でのコミュニケーションの助けにもなります。

いろいろな事例と標準問題の対応を知ることが、標準問題を見つけるための訓練になるでしょう。

標準問題を数理問題として、とらえることも重要です。数理問題の違いによって、解きやすさが変わってくるからです。ここであげている事例の数理問題は、一部を除き混合整数最適化問題になります。ただし、「空箱の輸送コスト最適化」は近似的に線形最適化問題として解いています。また、「大規模データベース配置」は非線形最適化問題としています。

事例	標準問題クラス	標準問題
空箱の輸送コスト最適化	グラフ・ネットワーク問題	最小費用流問題
ビークル間連携配送最適化	経路問題	運搬経路問題
船舶スケジューリング最適化	集合被覆・分割問題	集合被覆問題
店舗シフトスケジューリング	スケジューリング問題	スケジューリング問題
3 次元パッキング最適化	切出し・詰込み問題	n 次元パッキング問題
避難施設配置最適化	配置問題	施設配置問題
大規模データベース配置	割当・マッチング問題	一般化割当問題
バス仕業作成最適化	割当・マッチング問題	最大マッチング問題

空箱の輸送コスト最適化

背景

物を箱に入れて輸送しているときに、需要地と供給地に偏りがあると、空箱が需要地に溜まっていきます。この空箱を供給地に戻さないといけません。コンテナやパレットやレンタカーなど、いろいろな分野で見られます。

問題

グラフ上において、複数の需要点と複数の供給点があります。需要点から供給点へ、費用を最小にするフロー (流量) を求めます。

- 標準問題の最小費用流問題となります。解法としては、負閉路除去法などがあります。
- 事例では、数理問題の線形最適化問題として定式化し、汎用ソルバーで解いています。理由は、追加の制約があること。混合整数最適化問題とすると、計算時間がかかりすぎることによります。整数性を無視しているため、0.5 個のような解が出ることがあります。

ビークル間連携配送最適化

背景

物資の保管所から避難場所に物資を運びたい。輸送手段は、陸海空全て使うことができます。また、フェリーなどの定期便も利用できます。

問題

品物、輸送元、輸送先、数量、輸送期限からなる配送オーダーを満たす配送計画を求めます。配送計画では、いつ、何を、どうやって運ぶかを決めます。オーダーは 1 回で運んでもよいし、複数の輸送手段で連携して運んでもよいです。

- 標準問題の運搬経路問題となります。配送計画問題ともよばれます。運搬経路問題は、多くのソフトウェアが売られていますが、局所探索法をベースにした解法が用いられることが多いです。
- 事例では、貪欲法の一種である挿入法を用いています。数理問題の混合整数最適化問題になりますが、問題が大きすぎるので汎用ソルバーで解くことはできません。

船舶スケジューリング最適化

背景

船を使って、生産工場から倉庫に物資を輸送しています。倉庫からは、定期的にトラックで需要地に運んでいます。

問題

倉庫が空にならないように、生産工場から倉庫へ輸送する船のスケジュールを求めます。

- 船のスケジュールの候補を採用するしないを変数とすることにより、標準問題の集合被覆問題とすることができます。解法としては、分枝限定法、動的最適化などがあります。
- 事例では、混合整数最適化問題に定式化して汎用ソルバーで解いています。

店舗シフトスケジューリング

背景

全国の店舗では、店長が毎月末に翌月の社員の勤務スケジュールを作成しています。様々な制約があるため、作成には時間がかかっています。

問題

各店舗ごとに、社員の勤務スケジュールを作成します。勤務スケジュールでは、社員ごと日ごとのシフトを決定します。シフトは、日勤、休み、早番、遅番などがあります。制約としては、各社員の休みの希望や各日ごとの最低シフト数やシフトの禁止パターンなどがあります。

- 標準問題のスケジューリング問題となります。
- 事例では、混合整数最適化問題として定式化し、近似解法の制約計画ソルバーを用いて解いています。

3 次元パッキング最適化

背景

航空貨物のほとんどは、パレット上にパッキングして、航空機内の特殊な形状に合わせて詰め込んでいます。

問題

様々な大きさの貨物をパレット上に効率よく詰め込む方法を求めます。一番下に置かないといけないとか上に積んではいけないとか様々な制約があります。

- 標準問題の n 次元パッキング問題となります。解法としては、貪欲法などがあります。
- 事例では、2 次元の貪欲法を 3 次元に拡張した解法を独自に開発しています。

避難施設配置最適化

背景

津波に備えて、避難ビルを整備することにします。避難ビルには、物資を保管するため費用がかかります。

問題

全ての避難者を収容可能として、費用を最小にする避難ビルを選択します。

- 標準問題の施設配置問題となります。解法としては、分枝限定法などがあります。
- 事例では、混合整数最適化問題として定式化し、汎用ソルバーで解いています。目的関数として、全ての避難者の避難完了時刻を最小にすると、非常に難しい問題となります。その場合は、近似的に避難完了時刻を制約にして解くことがあります。

大規模データベース配置

背景

コールセンターでは、全国から多くの問い合わせが来ます。問い合わせに答えるため個人データがデータベースに格納されています。個人データは、地方ごとにまとめますが、保存先のストレージは、複数あります。

問題

個人データの保存するストレージ先を選択します。そのときに、ストレージごとのアクセスが平準化されるようにします。

- 標準問題の一般化割当問題の変形となります。
- 事例では、数理問題の非線形最適化問題とし、局所探索法で解きました。平準化問題を混合整数最適化問題用の汎用ソルバーで解くのは効率が悪くなります。逆に、局所探索法は平準化問題に対して効率よく解けます。

バス仕業作成最適化

背景

バスのダイヤを組合わせて仕業を作成します。1 つの仕業は 1 人のドライバーが対応します。労働基準法も満足しなければいけません。

問題

バスのドライバーの人数を最小になる仕業を作成します。

- 標準問題の最大マッチング問題となります。解法としては、エドモンズ法などがあります。
- 事例では、混合整数最適化問題として定式化し、汎用ソルバーで解いています。

7 Python による実行例

レッスン 4

実際にやってみよう。

ソフトウェアのインストールは、10 章を見てください。Python は、2.7 でも 3.4 でもどちらでも同じプログラムで動きます。

ナップサック問題

ナップサック問題

ナップサックに、いくつかの荷物を詰込みます。詰込む荷物の容量 (size) の和がナップサックの容量 (capacity) を超えないように、荷物の価値 (weight) の和を最大にします。

Listing 1: ナップサック問題 (動的最適化)

```
from ortoolpy import knapsack
size = [21, 11, 15, 9, 34, 25, 41, 52]
weight = [22, 12, 16, 10, 35, 26, 42, 53]
capacity = 100
knapsack(size, weight, capacity)
```

```
>>>
(105, [0, 1, 3, 4, 5])
```

選択された荷物の価値の総和 (105) と選択した荷物の順番 ([0, 1, 3, 4, 5]) が得られます。

最短路問題

最短路問題

グラフ (9 章参照) において、始点から終点までの経路の中で最も短い経路を探します。

Listing 2: 最短路問題

```
import networkx as nx
g = nx.fast_gnp_random_graph(8, 0.26, 1)
nx.dijkstra_path(g, 0, 2)
>>>
[0, 1, 6, 3, 5, 2]
```

8 個の点からなるランダムなグラフを作成し、ノード 0 からノード 2 への最短路となるノードのリスト ([0, 1, 6, 3, 5, 2]) が得られます。

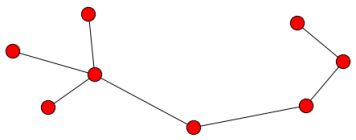

図 4: 最短路問題

最小費用流問題

最小費用流問題

グラフ (9 章参照) において、点ごとに需要量と供給量が与えられます。辺ごとにはフローの上限である容量が決められています。供給点から需要点へ費用が最小となるフローを求めます。

Listing 3: 最小費用流問題

```
import networkx as nx
g = nx.fast_gnp_random_graph(8, 0.2, 1, True)
g.node[1]['demand'] = -2 # 供給
g.node[7]['demand'] = 2 # 需要
g.adj[2][7]['capacity'] = 1 # 容量
result = nx.min_cost_flow(g).items()
for i, d in result:
    for j, f in d.items():
        if f: print((i, j), f)
>>>
(1, 2) 2
(2, 3) 1
```

(2, 7) 1
(3, 7) 1

8 個の点からなるランダムなグラフを作成します。ノード 1 から 2 の供給、ノード 7 に 2 の需要、ノード 2 からノード 7 への辺の容量を 1 にします。

最小費用流となるフローの辞書が得られます。

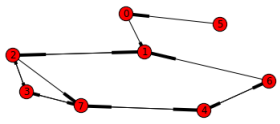

図 5: 最小費用流問題

最小費用流問題 (定式化)

最小費用流問題を定式化すると、次のようになります。

$$\text{最小費用流問題} \quad \left| \begin{array}{ll} \text{最小化} & \sum_{e \in E} c_e x_e \\ \text{制約条件} & \sum_{e \in v_{out}} x_e - \sum_{e \in v_{in}} x_e = f(v) \ \forall v \in V \\ & 0 \leq x_e \leq u_e \ \forall e \in E \end{array} \right.$$

ただし、c_e は辺の費用を、u_e は辺の容量を、$f(v)$ は点の需要量 (正) または供給量 (負) を表しています。また、v_{out} は点 v から出る辺の集合を、v_{in} は点 v に入る辺の集合を表しています。

これを PuLP で表すと次のようになります。

Listing 4: 最小費用流問題 (定式化)

```
import networkx as nx
g = nx.fast_gnp_random_graph(8, 0.2, 1, True)
g.node[1]['demand'] = -2
g.node[7]['demand'] = 2
g.adj[2][7]['capacity'] = 1

from pulp import *
m = LpProblem()
x = {e:LpVariable('v%d'%i, 0, g.edge[e[0]][e[1]]\
      .get('capacity', None)) for i, e in enumerate(g.edges())}
m += lpSum(g.edge[e[0]][e[1]].get('weight', 1) * x[e] for e in g.edges())
for v in g.nodes():
    m += lpSum(x[e] for e in g.in_edges(v)) \
       - lpSum(x[e] for e in g.out_edges(v)) == g.node[v].get('demand', 0)
m.solve()
for e in g.edges():
    if value(x[e]): print(e, value(x[e]))
>>>
(1, 2) 2.0
(2, 3) 1.0
(2, 7) 1.0
(3, 7) 1.0
```

最小費用流問題は、数理問題の線形最適化問題になります。

ナップサック問題 (定式化)

ナップサック問題を定式化すると、次のようになります。

$$\text{ナップサック問題} \left| \begin{array}{ll} \text{最大化} & \sum_{i \in N} w_i x_i \\ \text{制約条件} & \sum_{i \in N} s_i x_i <= c \\ & x_i \in 0,1 \ \forall i \in N \end{array} \right.$$

ただし、N は荷物の集合を、w_i は荷物の価値を、s_i は荷物の容量を、c はナップサックの容量を表しています。

これを PuLP で表すと次のようになります。

Listing 5: ナップサック問題 (定式化)

```
from pulp import *
size = [21, 11, 15, 9, 34, 25, 41, 52]
weight = [22, 12, 16, 10, 35, 26, 42, 53]
capacity = 100
m = LpProblem(sense=LpMaximize)
x = [LpVariable('v%d'%i, cat=LpBinary) for i in range(len(size))]
m += lpDot(weight, x)
m += lpDot(size, x) <= capacity
m.solve()
print(value(m.objective), [i for i in range(len(size)) if value(x[i]) > 0])
>>>
105.0 [0, 1, 3, 4, 5]
```

ナップサック問題は、数理問題の混合整数最適化問題になります。

ポートフォリオ最適化問題

ポートフォリオ最適化問題

銘柄を購入する割合 (x_i) を決めます。投資に対する利益の下限 (t/円) を条件として、リスク (分散 (v_i)) を最小化します。ただし、各銘柄間は独立とします。

ポートフォリオ最適化問題を定式化すると、次のようになります。

$$\text{ポートフォリオ最適化問題} \left| \begin{array}{ll} \text{最小化} & \sum_{i \in N} v_i x_i^2 \\ \text{制約条件} & \sum_{i \in N} p_i x_i >= t \\ & \sum_{i \in N} x_i = 1 \\ & 0 \leq x_i \leq 1 \ \forall i \in N \end{array} \right.$$

ただし、N は銘柄の集合を、p_i は銘柄の 1 円当たりの利益を表しています。

これを PuLP で表すと次のようになります。

Listing 6: ポートフォリオ最適化問題

```
import numpy as np, scipy.optimize as so
p = [31, 86, 29, 73, 46, 39, 58] # 利益 / 円
v = [10, 60, 25, 50, 35, 30, 40] # 分散 / 円
t = 50 # 目標利益 / 円
so.fmin_slsqp(lambda x: sum(v*x*x), np.zeros(len(p)),
    eqcons=[lambda x: sum(x) - 1], ieqcons=[lambda x: sum(p*x) - t])
>>>
```

```
Optimization terminated successfully.    (Exit mode 0)
            Current function value: 4.50899167487
            Iterations: 14
            Function evaluations: 136
            Gradient evaluations: 14
array([ 0.26829785,  0.13279566,  0.09965076,  0.1343941 ,  0.11783349,
        0.11506705,  0.13196109])
```

ポートフォリオ最適化問題は、数理問題の非線形最適化問題になります。

8 演習

次の問題は、どの標準問題になるでしょうか？

演習 1

物流センターから 5 カ所の店舗にオーダー表による品物を 2 台のトラックで運びます。トラックの総燃料費が最小となる配送方法を求めます。

演習 2

営業から大きさの違うたくさんの箱をコンテナに詰めて運びたいので、見積りを知りたいと聞かれました。全ての箱が 1 つのコンテナに詰込みできるかどうかを調べます。

演習 3

津波に備えて、高層ビルを避難場所に指定します。避難ビルには、物資保管場所を確保するため費用がかかります。全ての住民が避難可能の上で、必要な避難ビル数を最小にします。

9 グラフ・ネットワーク

グラフ (graph) とは、点 (vertex) と点同士を結ぶ辺 (edge) で構成される構造です。現実の関係を抽象的に表現するためによく用いられます。

$G = (V, E)$ とは、「グラフ (G) が点集合 (V) と辺集合 (E) で構成される」ことを表します。

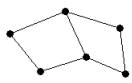

図 6: グラフの例

辺が一方通行のものを有向グラフ、そうでないものを無向グラフといいます。ある点から、別の点へ続く一連の辺を経路 (path) といいます。経路の始点と終点が繋がっているものを閉路 (cycle) といいます。

グラフ上の辺に何かしらを流れ (flow) を考えたものをネットワーク (network) といいます。道路ネットワーク、通信ネットワークなどがあります。

10 ソフトのインストール

インストールが簡単にできる方法として、Anaconda を利用する方法をおすすめします。Anaconda のインストールは、「http://continuum.io/downloads」からインストーラをダウンロードして実行してください。

Anaconda には、Python 本体と科学技術計算用の多くのパッケージが含まれています。Anaconda 以外に必要なソフトは、数理最適化のモデラーである PuLP と OR 用ライブラリになります。

PuLP のインストールは「pip install pulp」とします。

OR 用ライブラリのインストールは「pip install ortoolpy」とします。

11　注意すべきポイント

実務で最適化を使う上で注意すべきポイントがいくつかあります。現在の最適化技術では、現実の課題を完全にモデル化することはできません。重要度の低い部分を切り捨てて、モデルをシンプルにしないと解くことができません。従って、最適化の結果は、そのままでは現実には使えないことも多くなります。その場合、どういう問題が生じるのかを確認するために、検証をすることが重要になります。問題をいち早く見つけるために結果の見せ方も重要です。

12　余談

生物では、リンネによる二命名法 (例えば人類はホモ・サピエンス) が一般的ですが、最適化でも、数理問題・標準問題のとらえ方はこれに似ているかもしれません。

数理問題か標準問題かのどちらとしてとらえるかにより、ソルバーが変わりますが、ユーザーが判断して選択する必要があります。将来は、自動で判別できるようになるかもしれません。すなわち、数理問題としてソルバーに与えると、解析していずれかの標準問題であることがわかれば、自動的により効率的な解法を用いるようになるかもしれません。

東ロボくんをご存じでしょうか？国立情報学研究所が中心となって、「ロボットは東大に入れるか」をテーマにした研究プロジェクトです。

実は、標準問題による最適化アプローチは、東ロボくんのアプローチと似ているところがあります。それは、千差万別の問題を類型問題としてとらえて解くというところです。

この試みを見ていると、日本語で問題を記述するだけで、自動で最適化して解けるようになる日が、いずれ来ると思われます。

しかし、それまでは自分で考えて、手を動かす必要があるでしょう。また、そうすることによって、徐々に、より複雑な問題への対処法を身につけることができると思います。