実際の数理最適化問題を瞬時に解くための実装技術

久保幹雄

東京海洋大学

2014/8/27

発表の流れ

研究者による実際問題の実装のコツ

- なるべくプログラムを書かない
  - 学生に丸投げしてはいけない
  - 既存のライブラリを活用
- 書くにしてもなるべく短いプログラム
  - Python 言語
    ⇒「Python 言語によるプログラミング・イントロダクション」（近代科学社）
  - 正しい定式化の実装例を利用
    ⇒「あたらしい数理最適化」（近代科学社）
  - メタヒューリスティクスのプログラム例を利用
    ⇒「メタヒューリスティクスの数理」（共立出版）

解決のための手順

1. モデルの抽出
2. データ収集と確認
3. データの読み込みと解析
4. モデルの構築と実装
5. モデル実行（最適化）
6. 結果解析（可視化と評価）．そして 1. に戻る．

目次

モデルの抽出

1. 実務家の話を良く聞く
2. 現場の人の話も良く聞く
3. 問題を分類して整理し，解ける規模の問題に分解
4. 細かすぎるデータは集約
5. 既存のモデルに帰着させる
6. モデルの変更に強いアプローチを選択

目次

データ収集と確認

1. 正しいデータを集める
2. 欠損値を補う
3. 誤データを発見，修正する（⇐ データの可視化が必要）

例：販売量データ –ある日の需要がない場合には空白 –

在庫があるのに需要がない ⇒ 単に空白を 0 に置換

在庫がなくて販売できない ⇒ 前後の需要から真の需要量を推定

目次

データの読み込みと解析

pandas (Python Data Analysis Library) モジュールの利用

例：顧客に複数の営業マンが班を組んで訪問販売
顧客データ Custmer.csv

顧客名，訪問する班の必要レベル，班の最低人数，訪問時間

```
name,level,number,time
Kitty,A,3,5.03333
Panda,B,1,1.36667
Bear,C,2,2.2
Polar Bear,B,2,
Rabbit,B,2,7.03333
...
```

pandas で読み込み

```
import pandas as pd
df = pd.read_csv("Customer.csv")
```

1. pandas モジュールを pd という名前で読み込み
2. csv ファイルを読み込みデータフレームオブジェクト df（Excel のシートに対応）を生成

単位の変換

列 `time` は時間単位

⇒ 分単位に変換した列 `minutes` を作成！

データフレームの列は Python の辞書のようにアクセス可能

```
df["minutes"] = df["time"] * 60
```

データの確認

head メソッド（オブジェクトに付随する関数）を使う

```
df.head()
```

	name	level	number	time	minutes
0	Kitty	B	3	5.03333	301.9998
1	Panda	B	1	1.36667	82.0002
2	Bear	C	2	2.20000	132.0000
3	Polar Bear	B	2	NaN	NaN
4	Rabbit	B	2	7.03333	421.9998

NaN は “Not A Number” の略で pandas の欠損値

欠損値の処理

欠損値を他の顧客への訪問時間の平均値で置き換え

```
df["minutes"].fillna(value=
        df["minutes"].mean(),inplace=True)
```

分単位の訪問時間の平均を `mean` メソッドで得て，データフレームの `fillna` メソッドで欠損値を置き換え

クロス分析

Excel のピボットテーブルに対応

例：レベルと人数のクロス分析

```
print pd.crosstab(df["level"], df["number"])
```

number	1	2	3
level			
A	10	106	20
B	101	749	123
C	30	50	0

クラスの準備

```
class Customer():
    def __init__(self,name,level,number,minutes):
        self.name    = name
        self.level   = level
        self.number  = number
        self.minutes = minutes
    def __str__(self):
        ret="Name=%s  Level= %s Number=%s Time= %s"
            %(self.name,self.level,self.number,int(
            self.minutes))
        return ret
```

__init__はコンストラクタ

__str__は文字列を返すメソッド

```
>> print cust1
Name=Kitty  Level= B Number=3 Time= 301
```

顧客クラスの生成

顧客クラスのオブジェクトをリストに保管

```
for i in df.index:
    r = df.loc[i]
    c = Customer(r["name"], r["level"],r["number"],
        r["minutes"])
    Customers.append( c )
```

index メソッド：データフレームの行のリストを返す
loc メソッド：行データ r の抽出

pickle で保管

（データが多いときには）作成したクラスオブジェクトを pickle（漬け物）で保管

```
f = open('cust.dmp','w')
cPickle.dump( Customers, f )
f.close()
```

次回からは，直接顧客オブジェクトのリストをファイルから高速に読み込み

```
f = open('cust.dmp','r')
Customers = cPickle.load(f)
f.close()
```

目次

モデルの構築

- モデルの構築と解法の選択は同時に行う
  - 数理最適化ソルバー
  - 制約最適化ソルバー
  - 問題に特化したソルバー
  - メタヒューリスティクス
  - 列生成法
  - 分枝切除法
  - 数理最適化ソルバーベースのメタヒューリスティクス
  - ...
- 問題例に含まれる数値（データ）も重要
- 封筒の裏の計算を活用

解法の選択

- 実務的な問題はほぼ 100% $\mathcal{NP}$-困難
- すべての問題例を解けるような解法はムリ
- 将来どのような問題例が入力されるかは分からない
- データ解析によって得られた標準的な問題例で大規模のものを解けるようにする！

解法選択の指針

- メンテナンスが必要 ⇒ 汎用ソルバー
- 腕に自信があるなら ⇒ メタ
- 汎用ソルバーで解けないくらい難しい ⇒ メタ
- 予算がない ⇒ 無料のソルバーかメタ
- ほとんどの変数が実数変数で，双対ギャップが比較的小さい（凸二次）混合整数計画問題 ⇒ MIP ソルバー
- 組合せ的な構造のみの離散最適化問題で，かつ近似でも良い ⇒ メタに基づく制約最適化ソルバー
- 組合せ的な構造のみの離散最適化問題で，対称性が強い ⇒ メタに基づく制約最適化ソルバー
- 順序付けを主目的としたスケジューリングタイプの問題 ⇒ スケジューリング最適化ソルバー

解法選択の指針 (2)

「あたらしい数理最適化 -Gurobi と Python 言語で解く- 」のサポートページの実験的解析を参照

- 多制約ナップサック問題，施設配置問題，二次錐最適化問題（e.g., ポートフォリオ） $\Rightarrow$ MIP ソルバー
- グラフ分割，最大安定集合，グラフ彩色問題，スケジューリング問題，配送計画問題 $\Rightarrow$ メタ
- ロットサイズ決定問題，非線形関数を含んだ整数最適化問題 $\Rightarrow$ 問題例の規模による

例：営業班による顧客の訪問

- 1 週間の営業班の編成を最適化したい.
- 毎日，班編成は変更しても良い.
- 営業マンのレベルは A,B,C の順で，上位のレベルを持つ営業マンは下位のレベルも有する.
- 班のレベルは，班に含まれる営業マンで最もスキルが高い人のレベルとする.
- 1 つの顧客は 1 つの班で処理する.
- 1 つの班が 1 日に訪問する顧客は，できるだけ地理的に近い必要がある.
- 班が 1 日に稼働できる時間の上限は決まっている.

クロス分析より，顧客の必要とする班の人数は最大 3 人，
レベルは 3 種類 (A,B,C)
$\Rightarrow$ 可能な班（パターン）を事前に列挙

定式化に必要な記号

t : 期（日）を表す添え字.

U_{pt}: 期 t での班 p の作業時間の上限.

J : 顧客の集合.

L : レベルの集合.

N_l : レベル l を持つ営業マンの人数.

C_p: パターン p の班で処理可能な顧客の集合.

n_p : パターン p の人数.

PAT : パターン集合.

PAT_l : 最上位レベル l の営業マンを含むパターンの集合.

M_i : 顧客 i の訪問時間.

d_{ij} : 顧客 i, j 間の移動距離.

変数

y_{jpt} : 顧客 j をパターン p の班が期 t に訪問するとき 1，それ以外は 0.

X_{pt} : 期 t にパターン p の班を編成するとき 1，それ以外は 0 .

定式化

$$
\begin{aligned}
\text{min.} \quad & \sum_{p,t} \sum_{i<j} d_{ij} y_{ipt} y_{jpt} & \\
\text{s.t.} \quad & \sum_{p \in PAT} n_p X_{pt} \leq \sum_{l} N_l & \forall t \\
& \sum_{p \in PAT_l} X_{pt} \leq N_l & \forall l, t \\
& \sum_{j \in C_p} M_j y_{jpt} \leq U_{pt} X_{pt} & \forall p, t \\
& \sum_{p,t} y_{jtp} = 1 & \forall j
\end{aligned}
$$

- 班が期に訪問する顧客間の距離の合計を最小化
- 班編成のための制約条件（2 本）
- 作業時間上限
- 顧客割り当て

ソルバーの選択

非凸の二次の目的関数をもつ整数最適化，解の対称性あり
⇒ 制約最適化ソルバー SCOP (Slver for COnstraint Programming) を利用

- メタヒューリスティクスのタブーサーチを利用（頑強）
- Python インターフェイス（楽）
- 制約逸脱ペナルティの和の最小化
- 変数は領域と呼ばれる集合から 1 つの値を選択
  - MIP ソルバー: $\{0, 1\}$ の領域を持つ変数 y_{jtp}
  - SCOP: 顧客 j を訪問可能なパターン p と期 t の組の集合を領域とした変数 Y_j
  - ⇒ 変数の数の大幅削減 + 顧客割り当て制約が不要

目次

可視化

可視化モジュール matplotlib を利用
pylab インターフェイスは商用の MATLAB と同様
例：顧客訪問時間（分単位）の度数分布表

```
import pylab
pylab.hist(df["minutes"])
```

ネットワークの描画と簡単な最適化には networkX モジュールを利用

まとめ

- OR の普及のためには研究者によるヘルプが重要
- 実際問題を短時間で解くにはコツがある
- Python 言語を使うと，データ解析，最適化，可視化まで 1 つの環境内でできる！
- 参考：
  - 「Python 言語によるプログラミング・イントロダクション」（近代科学社）
  - 「あたらしい数理最適化」（近代科学社）
  - 「メタヒューリスティクスの数理」（共立出版）