Python言語による サプライ・チェイン・アナリティクス

久保　幹雄

ここでは，Python 言語を利用してサプライ・チェインの解析（アナリティクス）を行うためのテクニックについて，具体的な例（可視化，需要予測，ロジスティック・ネットワーク設計，スケジューリング）を挙げながら解説する．

キーワード：サプライ・チェイン，可視化，最適化，需要予測，スケジューリング，Python 言語

1. はじめに

本稿では，Python 言語を利用してサプライ・チェインの解析（アナリティクス）を行うためのテクニックについて述べる．ここでアナリティクスとは，米国 OR 学会 (INFORMS) のホームページや刊行物を見ればわかるように，OR の別名にほかならない．最近の OR は，古典的な手法だけでなく，データサイエンスや機械学習を駆使して問題解決を行うための学問体系に進化しているのである．

一般に，アナリティクスは以下の五つに分類できる．

1. 記述的：データを集計し，レポートを表示する．
2. 診断的：データを用いて診断を行う．
3. 発見的：データマイニングに代表されるように，データから何らかの知識を発見する．
4. 予測的：機械学習に代表されるように，データから未来を予測する．
5. 指示的：最適化を用いて将来に対する行動を指し示す．

記述的・診断的の境は曖昧だが，そこにはデータ集計や可視化のほかに，シミュレーション，待ち行列理論，在庫モデル，PERT (Program Evaluation and Review Technique) などが含まれる．同様に，発見的・予測的の境も曖昧であるが，そこには回帰分析，時系列分析，判別分析などの統計分析や連想分析，データマイニング，機械学習などが含まれる．これらの分析は主に戦略を評価するためのものであり，戦略を生成する指示的アナリティクスとは明確な違いがある．指示的（規範的とも訳される）アナリティクスには，最適化の諸技法が含まれる．つまり，アナリティクスは OR の解析手法そのものであり，INFORMS が（OR からアナリティクスに）看板を付け替えたのも頷ける．

以下の構成は次のとおりである．

2 節では，本稿の理解に必要なサプライ・チェインの基礎的な概念について述べる．

3 節では，サプライ・チェインのための Pandas データフレームの設計について述べる．

4 節では，可視化ツールの選択法と使い方について述べる．

5 節では，需要予測の方法と注意点について述べる．

6 節では，生産と輸送を統合したモダンな多期間ロジスティック・ネットワーク設計モデルについて述べる．

7 節では，工場内での最適化であるスケジューリングモデルと，その適用事例について述べる．

8 節では，まとめと今後の課題について述べる．

2. サプライ・チェインの基本概念

サプライ・チェインでは，原料から消費までのモノの流れを情報を用いて最適化することを目標とする．サプライ・チェインの解析を行うためには，以下で述べる処理的・解析的情報技術の違い，意思決定レベル，集約・非集約の概念を理解しておくことが重要である．

処理的情報技術 (IT) とは，人間でいうと神経網に相当し，日々の業務のための情報伝達を行うための IT を指す．通常，企業のデータベースに保管されているのは処理的 IT のためのデータであるので，これを処理的データと呼ぶ．一方，解析的 IT は，人間でいうと頭脳に相当し，意思決定を行うための IT を指す．アナリティクスとは解析のことであり，アナリティクスのための技術は解析的 IT の一部である．解析的 IT に必要なデータは解析的データと呼ばれ，処理的データや外部データから生成する必要がある．具体的には，企

くぼ　みきお
東京海洋大学
〒 135–8533 東京都江東区越中島 2–1–6
kubo@kaiyodai.ac.jp

業内のデータベース（処理的データ）や，Excel や BI (Business Intelligence) ツールなどにばらばらに保管されているデータから，解析に必要なデータだけを抽出し，さらにデータベースで管理していない情報（たとえば地点の緯度・経度や天気予報）を追加したものが解析的データになる．これを適切な形式（たとえば次節で述べる Pandas のデータフレーム）に保管して，企業全体で共有できるようにしておくのである．

意思決定レベルは，通常，ストラテジック，タクティカル，オペレーショナルに分類され，順に，長期，中期，短期と（カタカナから和語にという意味で）意訳される．長期，中期，短期の区別は明確なものではなく，最近では複数の意思決定レベルにまたがるモデルが必要とされるようになってきている．たとえば，従来は長期レベルの意思決定モデルであったロジスティック・ネットワーク設計は，月次の生産・輸送計画にも用いられ，さらには短期のロットサイズ決定やスケジューリングと融合する必要性も出てきている（実際に求解可能かどうかは別問題だが）．

集約 (aggregation：ドリルアップ) は，データをまとめることを表し，非集約 (disaggregation：ドリルダウン）はその反対の操作を表す．これは，旧来は OLAP (OnLine Analytical Process) と呼ばれていた技術にほかならない．OLAP はデータベース上に格納されたデータを経営幹部に見せるための技術であったが，現在ではデータを可視化するための基本技術となっている．一般に，意思決定レベルが高いほど集約されたデータを扱い，低くなるにつれて非集約されたデータを扱う．つまり，短期レベルになるにつれてデータの粒度が細かくなるのである．

3. データフレーム

データフレームは Python のデータサイエンス用の定番モジュールである Pandas の基本データ構造である．簡単に言うと，Excel のシートのように，行（インデックス）と列から構成されるデータであり，データ分析に便利なさまざまな機能を有する．

注意：本稿では，Anaconda[1] を用いて基本的なモジュールがインストールされていると仮定してコードを記述する．基本モジュールの解説とインストール法については拙著 [1] を参照されたい．

サプライ・チェイン・データにおける基本軸として，時間，地点（工場，倉庫，顧客など），製品（最終製品，部品，原材料など）の三つが挙げられる．各軸には集約・非集約関係（親子関係）を定義したさまざまな粒度のデータを保管し，それを適宜切り出して解析を行う．これらのサプライ・チェインの基本データは，Pandas のデータフレームで保持しておくと解析が楽になる．

時間データに対しては，Python では `Datetime` 型が標準モジュールとして準備されており，これを用いて時刻と日付を合わせたデータとして取り扱う．この際，データフレームのインデックスに `Datetime` 型をもつデータを入れておくと `DatetimeIndex` という特別なインデックスになり，任意の期（日次，週次，月次，年次など）への集約や，特定の期間の切り出しを容易に行うことが可能になる．

地点データは，地球上の位置情報（緯度・経度）をもち，地理学的データベースと併せて解析を行う必要がある．地点の可視化については後述するが，高速にデータを加工するためには計算幾何学のアルゴリズムが必要になる．基本的な計算幾何学のデータ構造は，SciPy の spatial サブモジュールが利用できる．たとえば，緯度・経度から地点の集約や近接点の解析を行うには，spatial に含まれる K-次元木や Voronoi 図を使えば，大規模データでも高速な処理が可能になる [1].

製品データに対しては，集約・非集約の親子関係からなる木構造を作成しておく必要があり，そのためには部品展開表 (BOM) の解析が必要になる．ネットワークを扱うためには，NetworkX モジュールが便利であり，大規模データを対話型に描画するためには（後述する）Plotly がよい．

上述したように，行には日付を表す `DatetimeIndex` を入れるが，列には製品，製品群，地点，地点種別，地点群などからなる階層型インデックス (MultiIndex) をとるとよい．この方式の利点は Pandas の処理だけでさまざまなデータ変換ができるので，Python に精通していない人でもある程度の作業ができることであるが，弱点はデータ量が（若干ではあるが）増大することである．

例として，日付 `Period`，顧客 `Customer`，製品 `Product`，需要量 `Demand` の列をもつデータフレーム `demand_df` があるとしよう．ここから，上で述べた型のデータフレームを生成するためには，Pandas の `pivot_table` 関数を用いればよい．

```
import pandas as pd
demand_pivot = pd.pivot_table(demand_df,
    index = "Period", columns =["Customer"
    , "Product"], values="Demand" )
```

[1] https://anaconda.org/

これによって以下のようなデータフレームが生成される．

```
Customer Cust0... Cust3 Cust4
Product Prod0 Prod1 Prod2 Prod3 Prod...
Period
2017-01-01 NaN 44768.0 14193.0 31269.0 NaN
2017-01-02 17587.0 46371.0 NaN NaN NaN
2017-01-03 8155.0 NaN NaN 8700.0 NaN
2017-01-05 5959.0 NaN 51663.0 33063.0 NaN...
```

ここで NaN は数値が入っていないことを表す．行（インデックス）は日付であり，列は顧客と製品の階層である．

注文のない日付を埋めるためには，日付時刻の集約を行う resample メソッドを用いるとよい．引数の"1d" は 1 日単位で集約することを表す．さらに fillna メソッドで NaN を 0 に置換しておく．

```
demand_pivot = demand_pivot.resample("1d"
    ).sum().fillna(0)
```

このデータフレームを用いれば，顧客や製品に対する切り出しが簡単にできる．たとえば，ある顧客 (Cust2) に対する全製品の需要を合計し，それを 1 週間単位に集約したデータの累積量を計算するには，以下のようにすればよい．

```
demand_pivot.xs("Cust2", level="Customer"
    , axis=1).sum(axis=1).resample("1w").
    mean().cumsum()
```

ここで，xs は階層型インデックスの切り出しを行うためのメソッドであり，axis は軸（1 は列）を表す．また，cumsum は累積量を計算するためのメソッドである．

データ集約には groupby メソッドを用いる．以下では，製品ごとに需要を集約し，月次の合計を計算している．

```
demand_pivot.groupby(level="Product", axis
    =1).sum().resample("1m").sum()
```

結果は以下のようになる．

```
Product Prod0 Prod1 Prod2 Prod3 Prod4
Period
2017-01-31 283937 851852 620741 575210 642923
2017-02-28 587202 679136 471629 713835 362774
2017-03-31 9546 37546 211487 NaN 412...
```

ほかにも顧客，製品のマスター，地点・製品ごとの在庫量，部品展開表などもデータフレームとして保持しておき，必要に応じて切り出し，集約を行い，以下で述べる可視化を行う．

4. 可視化

サプライ・チェインの可視化（visualization：日本の業界用語では「見える化」）のためには，企業体内にあるデータ群を，理解しやすいように変換する必要がある．通常は，人間が認知しやすいように，データを加工し，可視化のための方法を決め，軸を決め，画面上にデータを写像する．人間が理解しやすい順として，位置（x, y 座標），大きさ，色調などがあるが，それらのうち，どれを選択するかは可視化デザイナーのセンスによる．

Python を用いて可視化を行うためのモジュールはたくさんある．単なる静的な画像を得るには matplotlib や seaborn を使えば容易にでき，動的で対話型の可視化するためには Bokeh や Plotly を使えばよい．サプライ・チェインは複雑なので，可能ならば，単なる静的な画像としてではなく，ユーザーが対話可能な動的環境として提供することが望ましい．さらには，企業内のさまざまな部署でデータを共有するためには，ブラウザで稼働し，データをクラウドで共有できるものが望ましい．これらの条件を満たすモダンな可視化モジュールとして Plotly/Dash がある．Plotly[2] とは，対話型の図を描画するための Python パッケージであり，クラウドを用いたデータ共有もサポートしている．（Plotly 自身はオープンソースであるが，クラウド上のデータをパブリックにしないためには有料のサービスに申し込む必要がある．）Dash[3] とは，Plotly を基礎として Web アプリを開発するための枠組みであり，こちらもオープンソースである．

サプライ・チェインの可視化には，三つの軸（時間，地点，製品）を，人間が容易に認知可能な 2 次元の情報に，どのように落とし込むかが重要になる．（Plotly などモダンな可視化ツールは 3 次元にも対応しているが，科学技術計算の可視化以外の実務では 2 次元を推奨する．）

時間軸を横軸にとって縦軸に需要量や在庫量をとると時系列データになるので，線グラフで描画する．問題は，ほかの 2 軸（地点と製品）であるが，色調か線種で区別して同じグラフに描画することも可能であるが，種類が多いと人間の目では識別できなくなる．Plotly を使えばマウスオーバーで情報を動的に表示させることもできるが，より詳細な分析のためには，並べ替え・選択・切り出し・集約の機能を図に追加して，対話的

[2] https://plot.ly/
[3] https://plot.ly/products/dash/

な可視化を目指すべきである．

地点データは緯度・経度の情報を平面上に射影して地図上に描画することによって可視化できる．この際，地理学的情報システム (GIS) を用いてもよいが，より柔軟に解析を行うためには Plotly の地図描画を用いるほうがよい．在庫や需要などの可視化と地図の描画を連動して行うことができるので，地図上で在庫の多い倉庫を大きくプロットしたり，クリックすると在庫の変化量を別のグラフに表示したりすることが容易にできる．Plotly は無料の地図のほかに，Mapbox[4]（商用サイトでの利用は有料）経由で OpenStreetMap[5]を用いることもできる．日本の詳細な地図を表示するには，こちらを推奨する．なお，ジオコーディング（住所と緯度・経度の変換）には，geocoder モジュール[6] を用いることを推奨する．

需要量，在庫量，輸送量，生産量などの数値情報は，地図上ではサイズ（点の大きさや線の太さ），色調，透過度で表すことができる．製品軸は切り出しや集約の機能を図に追加する．また地図上への描画では，時間軸は考慮できないので，適当な期間に集約して表示する必要があるが，Plotly ではスライダーやアニメーションが追加できるので，時間とともに変化する図が描画できる．

同様に，散布図，ヒストグラムや棒グラフなどが必要な場合でも，スライダーで（集約した）時間軸が追加できる．たとえば，月ごとの需要と返品の関係（散布図）や線形回帰の決定係数の分布（ヒストグラム）や各種費用と KPI (Key Performance Indicator)（棒グラフ）を時間軸のスライダー付きで描画することによってさまざまな洞察が得られる．

可視化はそれ自身が目的ではない．実務家が意思決定を行うため，そのためのヒントになる洞察を得るためや，最適化の結果の妥当性の検証などのために行うのである．単に目新しい可視化手法を用いたり，たくさんの機能を付加するのではなく，ユーザーが見やすく，必要な情報を容易に取り出せるような対話型可視化システムを目指すべきである．

5. 需要予測

Python ではさまざまな予測手法がすでにそろっている．たとえば予測の古典的手法である移動平均法は，Pandas モジュールのデータフレームのメソッドである `rolling` [7] を用いると簡単にできる．

`rolling` は指定した長さでデータを集計するので，結果に `mean` 関数を使えば移動平均，`std` 関数を使えば移動標準偏差が計算できる．たとえば，時系列データを表すシリーズ（1 列のデータフレーム）`ts` に対する 12 期の移動平均と移動標準偏差は，以下のように求めることができる．

```
rolmean = ts.rolling(window=12).mean()
rolstd = ts.rolling(window=12).std()
```

もう一つの古典的手法である指数平滑法は，同じく Pandas データフレームの `ewm` (exponentially weighted method) [8] を用いればよい．引数は `alpha` であり，これは減衰係数を指定する．

```
ts.ewm(alpha=.5).mean()
```

季節変動や傾向変動がある場合には，それを除去してから分析するのが常套手段である．Holt–Winter による指数平滑法の拡張もある．指数平滑法は実装も容易であり，バリエーションが豊富なので，公開されているコードを参考に自分で実装することを推奨する．自分で書くのが面倒な場合には，`statsmodels.tsa.seasonal` にある `seasonal_decompose` 関数[9] を使う方法もある．

```
from statsmodels.tsa.seasonal import
    seasonal_decompose
decomposition = seasonal_decompose(ts)
trend = decomposition.trend
seasonal = decomposition.seasonal
residual = decomposition.resid
```

これだけで，時系列データ `ts` を，傾向変動 `trend`，季節変動 `seasonal` と残差 `residual` に分解することができる．

指数平滑法のかわりに，より一般的な自己回帰和分移動平均 (ARIMA) 過程 `statsmodels.tsa.arima_model` も使うことができるが，サプライ・チェインにおける需要予測ではあまり使わないので省略する．予測手法としては回帰分析が定番であるが，statsmodels か scikit-learn モジュールを使えば容易にできる [1]．たとえば 3 節の需要データフレーム `demand_pivot` のある顧客 (`Cust2`) の累積需要量に対

[4] https://www.mapbox.com/
[5] https://www.openstreetmap.org/
[6] https://pypi.python.org/pypi/geocoder/
[7] https://pandas.pydata.org/pandas-docs/stable/generated/pandas.DataFrame.rolling.html
[8] https://pandas.pydata.org/pandas-docs/stable/generated/pandas.DataFrame.ewm.html
[9] http://www.statsmodels.org/dev/generated/statsmodels.tsa.seasonal.seasonal_decompose.html

して，scikit-learn を用いて線形回帰を行うには，以下のようにする．

```
from sklearn.linear_model import
    LinearRegression
import numpy as np
cum_demand = demand_pivot.xs( "Cust2",
    level= "Customer", axis=1).sum(axis
    =1).cumsum ()
reg = LinearRegression () # 線形回帰インス
    タンスの生成
x = np.array(range(len(cum_demand))).
    reshape(-1, 1)
y = cum_demand
reg.fit(x, y) # 誤差の最小化
yhat = reg.predict(x) # 予測
```

流行物としては深層学習（多層ニューラルネット）もあるが，サプライ・チェインではあまり必要ない．それよりも，簡単な回帰を行い，誤差の要因を見つけ，現実的な対処法を考えたほうがよい場合が多い．たとえば，メーカーの需要データは，卸までしか得られないことが多いため，小売店での最終需要とは異なり（いわゆる鞭効果のため）大きく変動する．特に，需要が 0 の期が続いた後に，非常に大きな需要が発生する傾向があるため，通常の回帰分析やニューラルネットでは予測がしにくい．需要分布も理論では当然のように仮定される（対数）正規分布や Poisson (Erlang) 分布とはかけ離れたものになる．理由はケースバイケースであるが，たとえば輸送効率のための車立て割引，返品制度やプロモーションなどが挙げられる．通常，末端の需要は定常に消費されているはずなので，累積需要をもとに簡単な（低次の多項式項を追加した）線形回帰，もしくは Holt–Winter の指数平滑法などで予測を行い，誤差の要因を割引，返品，プロモーションの情報をもとに分析することが推奨される．

需要データだけでなくほかのデータも加味して考えないと予測ができないことも多い．気温や天気などは当然だが，在庫量も重要である．在庫理論では，在庫がない場合には需要はバックオーダーもしくは品切れ (lost sales) として扱うが，実務では何の履歴も残さないため，たまたま誰も購入しなかったのか，それとも在庫がなくて売れなかったのか，意図的に品切れ状態にしたのか，誰にもわからないケースもある．これを解決するためには，現場へのヒヤリングと需要と在庫（できれば生産）を連動させた可視化を行うことが推奨される．さらには，在庫がなくて売れなかった場合には，在庫があった場合に発生したであろう需要を加味して予測を行う必要がある．また，頻繁なモデルチェンジを行う製品の場合には，販売開始と終了の時期なども重要なデータとなる．

しばしば，予測はコンペの対象として扱われ，なるべく予測が当たる手法を探索することに興味を奪われがちであるが，実務において重要なことは予測を当てることではなく，誤差を管理し，異常事態が発生したら適切な行動をとれるように準備しておくことなのだ．可視化と同様に，需要予測自身が目的ではなく，在庫管理や生産計画などの意思決定のため（最終的には費用削減のため）に行うことを忘れてはならない．予測精度をほんの少し上げるために高価な予測システムを導入することは本末転倒である．

6. 多期間ロジスティック・ネットワーク設計の拡張

旧来のロジスティック・ネットワーク設計モデルでは，倉庫や工場などの施設の開設・閉鎖を意思決定変数としたストラテジックレベルのものが主流であったが，実際には施設の開設・閉鎖の意思決定は稀であるため，近年ではより普段使いのシステムが必要となってきている．言い換えれば，最適化の頻度を上げて，頻繁にサプライ・チェイン全体の見直しを行うことが要求されているのだ．そこでは，工場での生産，輸送，在庫など，日々の運用のための最適化が必要になる．意思決定レベルとしては，中期（タクティカル）から短期（オペレーショナル）が重要になり，その中で長期的な意思決定も含めて（動的な環境の中で徐々に変えていく項目として）考えるのである．

中・短期の意思決定では，サプライ・チェイン全体での情報共有が必要になる．長期レベルのように本社の中枢部だけで意思決定を行うのではなく，日々の運用の中で会社全体で生産・在庫・輸送の情報を可視化し，フィードバックができるようにシステムを設計しておくことが重要になる．工場内の生産計画に対しては，古典的な生産計画のように段取り替え費用を無視したものではいけない．近年では，多品種少量生産が余儀なくされているため，段取り替えの意思決定が重要になってきているからだ．中期レベルでは，期の単位は月次の多期間ロジスティック・ネットワーク設計に工場内の段取りを考慮したモデルを推奨する．短期レベルでは，日次のロットサイズ決定モデルで大まかな段取り計画を立て，スケジューリングモデルで詳細なスケジュール計画を立てるのが望ましい．

中期レベルの最適化には数理最適化ソルバーを用いる．Python による定式化は比較的容易であり，商用の Gurobi[10] と無料の PuLP（ソルバーは CBC: Coin Branch and Cut）[11] は，筆者らが作成した mypulp モ

ジュール [1] を用いれば，ほぼ同じ文法で実装できる．特に注意すべきことは，規模が大きすぎる場合には，適切なデータの集約を行うことである．サプライ・チェインには時間，地点，製品の 3 軸があるが，どのデータにどのような集約を行うかはケースバイケースであり，アートである．ほぼ同一の地点にある顧客や倉庫は集約すべきであるが，単に似ているからといって製品を集約するのは危険である．細かいデータでの最適化を要求する顧客のいいなりになるのではなく，うまく説得し，気軽に求解できる範囲で最適化を行うのがコツである．

需要や供給の不確実性を考慮する必要がある場合には，モデル化は複雑になる．供給途絶に関しては，問題依存の特注モデルが必要になる．需要の不確実性を考慮したモデルにおいては，安全在庫の取り扱いが最も困難な部分になる．単純に確率最適化モデルを作ったところで，現実規模の問題例の求解は難しい．段取りを行う期が変数となるので，生産リード時間も変数になり，そのため各地点の安全在庫量も変数として扱わなければならない．数理最適化ソルバーを用いて安全在庫を近似的に考慮したアルゴリズムを作成する必要があるが，どういった解法がよいかは問題依存になる．

7. 工場内の最適化

工場内の生産計画を最適化するためのロットサイズ決定モデルやスケジューリングモデルは現場に依存して設計すべきものである．同じ企業でも工場が異なると生産の仕方が異なり，違うモデリングを行うことが必要な場合も多々ある．一般には，工程，品目（製品，半製品，部品，原材料などの総称），部品展開表（レシピ），作業員，機械などの実務における情報を，スケジューリングモデルで必要な情報（作業，作業モード，資源，時間制約，状態など）に変換する必要がある．

この変換の一例ではあるが，筆者が最近依頼を受けたスケジューリングの実際問題を，スケジューリングソルバー OptSeq[12] に変換したときの手順を示そう．

いま，製品には納期があり，その製品を製造するためには，部品展開表にしたがって必要な品目を順に生産していく必要があるものとする．顧客は製品の数量と納期を指定するが，それらの製品をばらばらに生産すべきか，一度にまとめて生産するかによってモデルが変わってくる．このようなロットまとめの意思決定は，ロットサイズ決定モデルを用いて最適化する場合もあるが，ここでは，ばらばらに生産するものと仮定する．その際には，最終製品 1 単位を製造することを表す作業を，必要な数量分だけ準備し，納期を同じ時刻に設定する．

部品展開表から必要な品目を列挙し，子品目から親品目を生産するための工程を作業として表す．ここで，複数の工程が必要な場合でも，途中に中断を入れながら連続して生産される場合には，一つの作業として表現する．作業には複数のモード（生産の仕方）を追加でき，さらにモードの途中での資源の使用量も，時間とともに変化するように設定できる．複数工程の間には，作業が中断可能であると定義する．この際，中断中も作業場を占有する場合には，中断中に資源を使用すると定義しておく．

工程間で発生する品目の在庫の表現法も重要である．OptSeq では在庫は明示的には扱うことができない．（在庫を陽的に考慮したモデルはロットサイズ決定モデルであり，スケジューリングモデルでは作業の開始時刻と資源への割り当てを最適化する．）在庫は，先行する作業が開始すると同時に発生し，後続する作業が終了するまで，工程間に存在すると考えられる．これをスケジューリングモデルで表現するためには，一つのダミー作業を準備し，先行作業の開始と同時刻に開始し，後続作業の終了時に終了するように設定する．作業時間は 0 で，作業開始直後に中断可能とし，中断中も在庫置き場を表す資源を使用すると設定することによって，直接的には表現できなかった在庫が表現できる．連続生産を行う場合の中間在庫量の上限も，（前後の工程の作業速度から計算される適切な）作業中断時間の上限を設定することによって表現できる．

上で示したのはモデル化の一例であるが，実際には現場ごとにモデルが異なってくる．ある工場で非常にうまく稼働したシステムが，同じ企業のほかの工場では（例え似たような製品を製造していても）全く機能しなかったというのは，稀なケースではなく，当然のことなのである．ましてや，ディスパッチング・ルール（これは古典的な貪欲ヒューリスティクスであり，処理的 IT に相当する）だけを搭載した汎用スケジューラーが現場できちんと動くことは稀であり，導入時にチューニングされて偶然動いていた（動いているように見えた）だけで，すぐに使われなくなることがほとんどなのである．上で示したようなダミーを利用したモデル化や典型モデルへの帰着は，OR の真骨頂だと思われるが，そういったことをできるセンスをもった

[10] http://www.gurobi.com/
[11] https://pythonhosted.org/PuLP/index.html
[12] http://logopt.com/OptSeq/OptSeq.htm

人材が圧倒的に不足しているのが，今日の（少なくとも日本国内の）OR の問題点だと感じる．

8. おわりに

ここでは，サプライ・チェインにおけるアナリティクスの使用法や注意点について述べた．今後の課題としては，サプライ・チェインの解析を行うための，ライブラリ群の整備や可視化のためのクラウドシステムの開発・提供が挙げられる．紙面の都合で十分には書ききれなかったが，機会があれば，それらを含めて続きを書いて出版したいと考えている．

謝辞　本研究は一部グローバル ATC (No.1005304) の助成を受けたものである

参考文献

[1] 久保幹雄，小林和博，斉藤努，並木誠，橋本英樹，『Python 言語によるビジネスアナリティクス―実務家のための最適化・統計解析・機械学習―』，近代科学社，2016.