全22章シリーズ ・ 第4章

AMPL: 数理計画のためのモデリング言語

より大きなモデルの構築

多品種輸送・多期間生産・生産と輸送の統合 — 小さな*LP*を結び合わせて大きなモデルを組み立てる

4.1 多品種輸送

4.2 多期間生産

4.3 生産と輸送

原著: Robert Fourer, David M. Gay, Brian W. Kernighan

AGENDA

本章の内容

4.1 多品種輸送モデル

前章の輸送モデルを拡張し、複数の製品を同時に発地から着地へ輸送する。発地・着地間の総輸送量を制限するMulti制約も導入する。

4.2 多期間生産モデル

生産計画を1週間ではなくT週間に拡張する。在庫変数Invを使い、Balance制約で各週の生産・販売・在庫をつなぐ。

4.3 生産と輸送のモデル

圧延所ごとの生産モデルと製品ごとの輸送モデルを1つに統合し、生産と輸送の両方の費用を最小化する。

INTRODUCTION

大きな線形計画問題はどう生まれるか

経路A：個々のモデルを精緻化する

添字集合を細かく分ければLPは大きくなる。だがある点を超えるとデータ維持も解の理解も難しくなり、細かくする利益は薄れる。

経路B：小さなモデルを結び合わせる

（本章のテーマ）

同種の単純なLPが多数生まれる場面で、それらを結ぶ変数や制約を加えると、全体は1つの大きなLPになる。

本章で扱う3つの結合パターン

- 製品ごとの輸送問題
- 週ごとの生産問題
- 生産×輸送の組合せ

製品の集合 PROD を追加する

- 前章の輸送モデルは単一商品のみを扱っていた
- 新しい集合 PROD（製品）を定義し、supply・demand・cost・Trans すべての添字に PROD を加える
- 発地 i から輸送される製品 p の量：supply[i, p]
- i から j へ輸送される製品 p の量：Trans[i, j, p]
- check 文も PROD で添字づけ、製品ごとに「供給の合計 = 需要の合計」を検証する



AMPLモデル：multi.mod（図4-1）

前半

```
set ORIG;   # origins
set DEST;   # destinations
set PROD;   # products

param supply {ORIG,PROD} >= 0;
param demand {DEST,PROD} >= 0;

   check {p in PROD}:
      sum {i in ORIG} supply[i,p] =
         sum {j in DEST} demand[j,p];

param limit {ORIG,DEST} >= 0;

param cost {ORIG,DEST,PROD} >= 0;
var Trans {ORIG,DEST,PROD} >= 0;
```

後半

```
minimize Total_Cost:
   sum {i in ORIG, j in DEST, p in PROD}
      cost[i,j,p] * Trans[i,j,p];

subject to Supply {i in ORIG, p in PROD}:
   sum {j in DEST} Trans[i,j,p] = supply[i,p];

subject to Demand {j in DEST, p in PROD}:
   sum {i in ORIG} Trans[i,j,p] = demand[j,p];

subject to Multi {i in ORIG, j in DEST}:
   sum {p in PROD} Trans[i,j,p] <= limit[i,j];
```

Multi制約：輸送能力を共有する

- 発地・着地の組合せで添字づけたパラメータ limit[i, j] を追加する
- Multi制約：各 (i, j) について、全製品 p の輸送量の合計が limit[i, j] を超えない
- この制約があると、ある製品の輸送量は他の製品の輸送量に左右されるため、製品ごとに独立して解くことはできなくなる

AMPL

```
param limit {ORIG,DEST} >= 0;

subject to Multi {i in ORIG, j in DEST}:
   sum {p in PROD} Trans[i,j,p] <= limit[i,j];
```

規模の例

発地 5 ・ 着地 20 ・ 製品 10 のとき

1,000
変数の数
（発地×着地×製品）

250
制約の数
（発地+着地）×製品

見かけほど大きくない

limit制約がなければ、製品の輸送は互いに独立なので、製品ごとに小さな輸送問題を別々に解く方が効率的である。limit制約が加わって初めて、1つの大きなLPを解く必要が生じる。

3次元データを2次元の表に切り分けて表示

ampl

```
ampl: model multi.mod; data multi.dat; solve;
CPLEX 8.0.0: optimal solution; objective 199500

ampl: display {p in PROD}: {i in ORIG, j in DEST} Trans[i,j,p];
Trans[i,j,'bands'] [*,*] (tr)
:    CLEV GARY PITT   :=
DET     0     0   300
FRA   225     0    75
FRE     0     0   225
LAF   225     0    25
LAN     0     0   100
STL   250   400     0
WIN     0     0    75
;
```

切り分け表示の考え方

ORIG × DEST × PROD の3次元データは、1つの添字（ここでは製品）に沿って切り分け、2次元の表の集まりとして表示する。display コマンドは切り分ける添字を自動推測するが、明示的な添字式を書けば表示方法を指定できる。

端数輸送の悩み

PITT→LAFへのバンド輸送はわずか25トン。「0か50トン以上」という条件は線形制約では表せない ―― 第20章の整数計画法で扱う。

coils・*plate* の表も同様に表示される（紙面の都合により省略）

週の集合 1..T を追加する

- 図1-4aの生産モデルを、1週間ではなくT週間分の計画に拡張する
- 1からTまでの整数の集合を表す略記 1..T を使う
- rate 以外の全パラメータ・変数をこの集合上に複製する
- 週ごとに制約 Time ができ、目的関数は製品と週の両方について合計する

図4-3

```
set PROD;                          # products
param T > 0;                       # number of weeks

param rate {PROD} > 0;             # tons/hour produced
param avail {1..T} >= 0;           # hours available
param profit {PROD,1..T};          # profit per ton
param market {PROD,1..T} >= 0; # limit on tons sold

var Make {p in PROD, t in 1..T}
     >= 0, <= market[p,t];

maximize Total_Profit:
   sum {p in PROD, t in 1..T}
      profit[p,t] * Make[p,t];

subject to Time {t in 1..T}:
   sum {p in PROD} (1/rate[p])
      * Make[p,t] <= avail[t];
```

在庫で週をつなぐ：Make・Inv・Sell

- Make[p, t] は生産量のみを表し、必ずしも販売量とは一致しなくなる
- 新しい変数 Inv[p, t]（在庫）と Sell[p, t]（販売量、上限 market[p,t]）を追加する
- 初期在庫パラメータ inv0[p] を、制約 Init_Inv によって Inv[p, 0] に固定する

Balance制約（週をつなぐ鍵）

生産 Make[p,t] + **前週末の在庫 Inv[p,t-1]** = **販売 Sell[p,t]** + **今週末の在庫 Inv[p,t]**

意味：生産量+繰り越された在庫 = 販売量+*繰り越す在庫（第1週では Inv[p,0] が初期在庫 inv0[p] になる）*

解の解釈：在庫を使って高値の週に売る

Make（生産量, トン）

週	bands	coils
1	5,990	1,407
2	6,000	1,400
3	1,400	3,500
4	2,000	4,200

Inv（在庫量, トン）

週	bands	coils
0	10	0
1	0	1,100
2	0	0
3	0	0
4	0	0

Sell（販売量, トン）

週	bands	coils
1	6,000	307
2	6,000	2,500
3	1,400	3,500
4	2,000	4,200

第1週に生産したコイル1,100トンは、より高値で売れる第2週まで在庫として持ち越されている。第2〜4週はコイルが上限まで販売され、バンドが残りの生産能力を埋めている。

生産モデルと輸送モデルを統合する

生産モデル
（圧延所 ORIG ごとに複製）
Make[i,p] ・ 制約 Time

輸送モデル
（製品 PROD ごとに複製）
Trans[i,j,p] ・ 制約 Supply/Demand

```
Make[i, p]  =  supply[i, p]
```

- ORIG（発地＝圧延所）と PROD（製品）の集合は両モデルで共通
- 輸送モデルの supply[i,p]（発地で利用可能な量）は、生産モデルの Make[i,p]（発地で生産される量）と実は同じもの
- supply の定義を落とし、Supply制約を「Σ(j∈DEST) Trans[i,j,p] = Make[i,p]」に書き換えて2つのモデルを1つに統合する

AMPLモデル：steelP.mod（図4-6）

前半

```
set ORIG;   # origins (mills)
set DEST;   # destinations
set PROD;   # products

param rate {ORIG,PROD} > 0;
param avail {ORIG} >= 0;
param demand {DEST,PROD} >= 0;

param make_cost {ORIG,PROD} >= 0;
param trans_cost {ORIG,DEST,PROD}
   >= 0;

var Make {ORIG,PROD} >= 0;
var Trans {ORIG,DEST,PROD} >= 0;
```

後半

```
minimize Total_Cost:
   sum {i in ORIG, p in PROD}
      make_cost[i,p] * Make[i,p] +
   sum {i in ORIG, j in DEST, p in PROD}
      trans_cost[i,j,p] * Trans[i,j,p];

subject to Time {i in ORIG}:
   sum {p in PROD} (1/rate[i,p])
      * Make[i,p] <= avail[i];

subject to Supply {i in ORIG, p in PROD}:
   sum {j in DEST} Trans[i,j,p] = Make[i,p];

subject to Demand {j in DEST, p in PROD}:
   sum {i in ORIG} Trans[i,j,p] = demand[j,p];
```

解の解釈：費用の内訳と双対値

1,392,175
総費用（objective）

1,215,250
生産費用（約87%）

176,925
輸送費用（約13%）

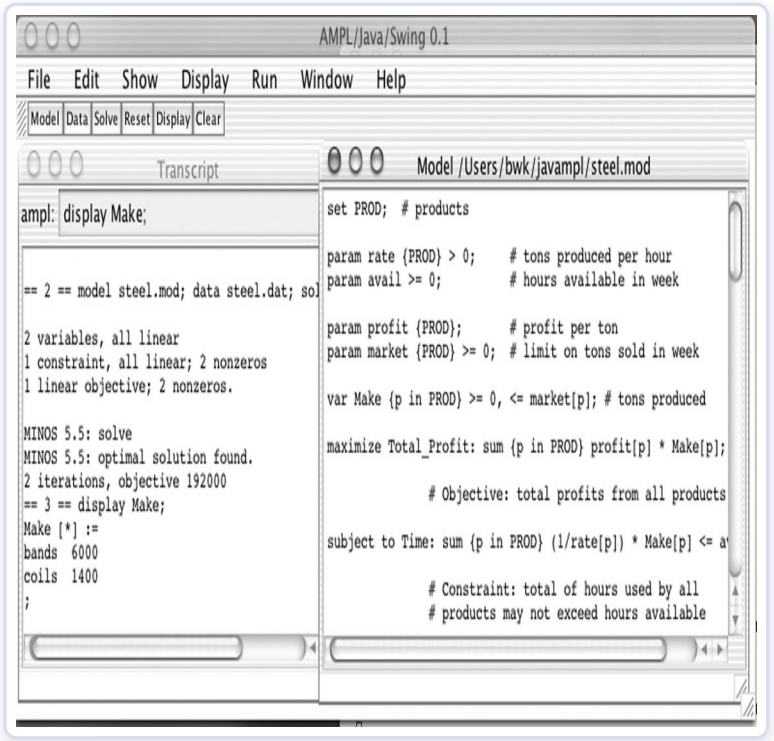
- 生産費用が総費用の大半を占め、輸送費用は相対的に小さい
- AMPLは任意の正しい式（合計など）を評価・表示できる ―― display sum{...} で費用内訳を直接確認できる

Time制約の双対値（限界価値）：GARY -2800 ／ CLEV -1300 ／ PITT 0
能力を追加するならGARYへの投資が総費用に最も大きく効き、PITTに追加しても効果はない。

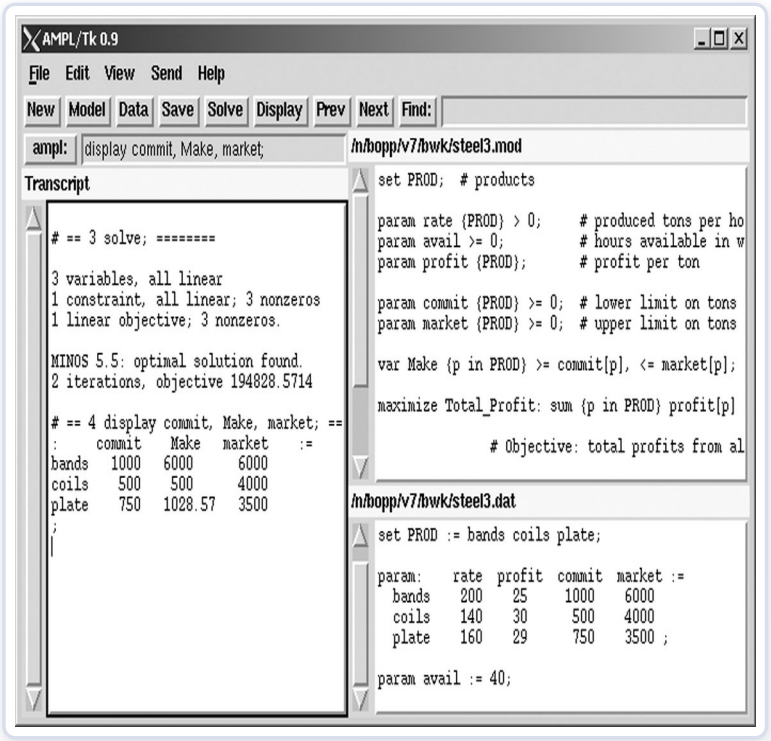
REFERENCE

AMPLの対話的モデリング環境（画面例）

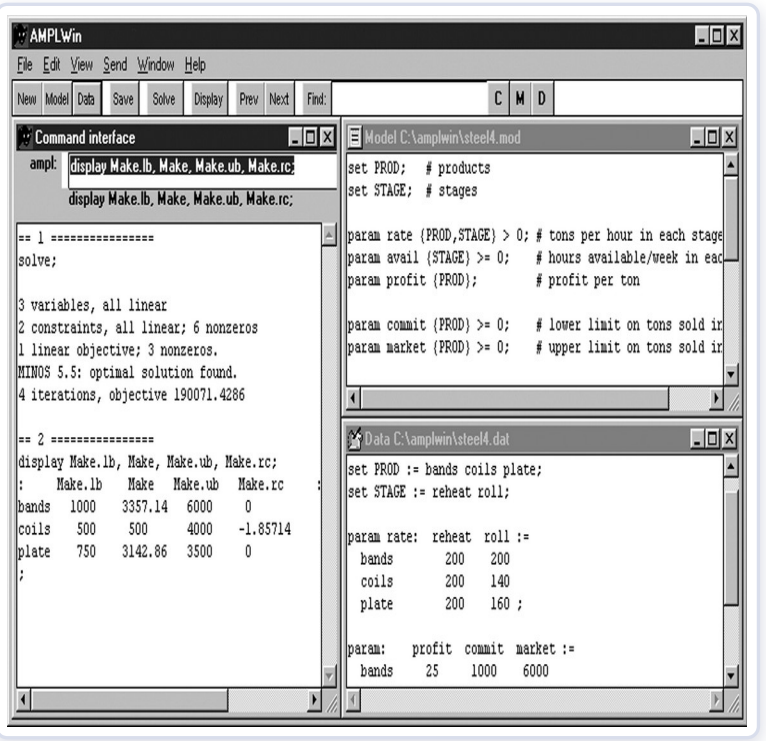
さまざまなAMPL統合環境（IDE）で、モデルの編集・solveの実行・display結果の確認を対話的に行っている様子



AMPL/Java/Swing — モデルの編集・solveの実行・displayによる結果確認という基本操作の例



AMPL/Tk — 変数に上限・下限（commit / market）を持たせたモデルの実行例



AMPLWin — 複数の添字集合（PROD × STAGE）でパラメータを添字づけたモデルの実行例

SUMMARY

本章のまとめ

- 大きなLPの多くは、小さな単純なLPを結び合わせることで生まれる
- 多品種輸送モデルはPROD集合を加えて添字を拡張し、Multi制約で発地・着地間の総輸送量を制限できる
- 多期間生産モデルは 1..T 集合で週を複製し、Balance制約（生産＋繰越在庫＝販売＋繰越在庫）で週をつなぐ
- 生産と輸送のモデルは、Make[i,p] = supply[i,p] という対応に着目して2つのモデルを1つに統合する
- AMPLは集合・添字を活用することで、モデルの本質的な論理を保ったまま規模を拡張できる