

第5章 集合から見た世界

*In Normal form, each attribute/field of an entity/record should depend on
the entity key, the whole key, and nothing but the key, so help me Codd.*

** anonymous*

5.1. はじめに

LINGO の一番の特色は、巨大なシステムをモデル化する能力です。これを可能にするのが、類似したオブジェクトを集合化するというコンセプトです。現実的な例をモデル化してみると、類似したオブジェクトが複数あることに気付きます。例えば、工場、製品、期間、顧客、乗り物、要員などです。LINGO は、このような類似したオブジェクトを一つのグループにまとめ、集合とします。集合ができると、一つのステートメントで集合に含まれるメンバーそれぞれに適用できます。

LINGO で作られる大きなシステムは、①SETS 節、②DATA 節、③モデルの方程式、から成ります。SETS 節では、ある種の問題を解く時にデータの構造を説明し、DATA 節は、データが定義されます。モデルの方程式部分では、各種データと決定した事項の関係が定義されます。

5.1.1. なぜ集合を使うのか

大規模なモデルでは、非常によく似た計算や制約をいくつも作る必要があります。LINGO は集合を使い、これらの方程式や制約を効率よく表現できます。例えば、100 ある倉庫の輸送モデルを考える場合、それぞれの制約式(例：「倉庫 1 は今ある在庫以上のものを輸送できない、倉庫 2 は今ある在庫以上のものを輸送できない、倉庫 3 は今ある在庫以上のものを輸送できない...」)を書くのは大変面倒な作業です。「各倉庫は今ある在庫以上のものを輸送できない」と一つの式で表す方が便利です。

5.1.2. 集合とは？

集合とは、類似したオブジェクトの群です。集合は、製品やトラックのリストであったり、要員であったりします。集合に含まれるメンバーには、それぞれ少なくとも一つの関連した特徴（例：重量、価格/単位、収入）があります。これらの特徴を「属性 (attribute)」と呼びます。同じ集合に含まれるメンバーは、同じ属性を持ちます。LINGO がそのモデルを解くためには、属性の値が前もって分かっても未知であっても構いません。属性の例として、製品には「価格」があり、トラックには運搬できる「許容量」があります。そして、雇用する要員には、「給料」や「誕生日」といった属性があります。

5.1.3. 集合のタイプ

LINGO には、原始と派生という 2 種類の集合があります。原始集合は、これ以上減らせないオブジェクトから成る集合です。派生集合は、2 種類あり、a) 部分集合の選択、b) デカルト積（クロス乗積）の両方もしくはどちらか一方を用い、1 つ以上の集合から成っています。派生集合に含まれるメンバーは既存の他の集合メンバーでもあります。例えば、「WAREHOUSE(倉庫)」と「Customer(顧客)」という二つの原始集合があるとします。そこには、「SHIPLINK(輸送経路)」という派生集合があります。SHIPLINK は、WAREHOUSE と CUSTOMER という二つの原始集合のあらゆる組み合わせを表しますが、これは他の派生集合から作ることも可能です。

5.2. 集合節 (SETS Section)

大抵の場合、集合を基盤とした LINGO のモデルでは最初の節が集合節となります。集合節は、「SETS:」（コロンを含む）というキーワードで始まり「ENDSETS」で終わります。しかし、モデルに必ず Sets 節が無くても構いません。逆に、複数の Sets 節を持つことも可能です。そして Sets 節は、モデルのどこに置いても問題はありませんが、重要な規定としてモデル内の制約を参照する前に必ず集合とそのアトリビュート(属性)を定義しなければなりません。

5.2.1. 原始集合の定義

Sets 節に原始集合を定義するためには：

- ・ 集合の名前
- ・ メンバーの属性

を指定する。書式は次のようになります。

集合名:[属性リスト];

「集合名」とはユーザーが集合につける名前のことです。覚えやすいように説明的な名前のほうが良いでしょう。次のように名前をつけます。英字 (A-Z) で始まり、その後は英数字 (1-9) かアンダースコア (_) もつかえます。そして 31 字以内にしてください。LINGO は、大文字と小文字の違いを認識しません。

例：

```
SETS:  
WAREHOUSE: CAPACITY;  
ENDSETS
```

これは、複数の WAREHOUSE(倉庫)があり、それぞれ CAPACITY (許容量)があることを示します。集合内のメンバーには、ゼロから複数の属性があり、それらを「属性リスト」に指定します。属性の名前は LINGO の命名規則に則り、各属性間は、カンマ(,)で区切ります。

例えば、WAREHOUSE に場所と船積ドックの数という属性が追加したとします。これらの属性は、次のように加えることができます。

```
WAREHOUSE: CAPACITY, LOCATION, DOCKS;
```

5.2.2. 派生集合の定義

派生集合を定義するには：

- ・ 集合の名前
- ・ 親集合
- ・ メンバーの属性（ある場合）

を指定します。書式は次のようになります。

集合名 (親集合のリスト) [メンバーシップフィルター] [:属性リスト];

「集合名」は、ユーザーが集合につける名前です。「メンバーシップフィルター」とは、オプションで集合に含まれるメンバーにつける一般条件です。「親集合リスト」はあらかじめ定義した集合で、カンマ(,) で区切られています。LINGO は各親集合から全てのメンバーの組み合わせで、派生集合のメンバーを作ります。例えば次の Sets 節を考えてみましょう。

```
SETS:  
PRODUCT ;  
MACHINE ;  
WEEK ;  
ALLOWED( PRODUCT, MACHINE, WEEK) : VOLUME ;  
ENDSETS
```

PRODUCT、MACHINE、WEEK は原始集合で、ALLOWED は PRODUCT、原始集合 MACHINE、WEEK から
の派生集合です。特に定めがない限り、ALLOWED には、上記三つの原始集合のメンバー全ての組み合わせが
含まれます。属性である VOLUME を使うことにより、各週どの機械からどの製品を生産するかを特定できます。メ
ンバー全ての組み合わせを含む派生集合を「密な集合(dense set)」と呼びます。メンバーシップフィルターかそ
の派生集合のメンバーが明確に DATA 節に含まれる場合、その集合は「疎な集合(sparse set)」と呼びます。

要は、派生集合のメンバーは下記のどれかから成ります。

- ・ DATA 節の明確なメンバー・リスト
- ・ メンバーシップフィルター
- ・ 派生集合のメンバーについて触れない場合は、黙示的に密である

DATA 節に含まれる派生集合の明確なメンバーシップリストの仕様は、次の項で説明します。

モデルが大きな疎な集合の場合、全てのメンバーをリストするのは困難です。幸いにも、多くの疎な集合では、
全てのメンバーがメンバーでないものと差別化できるような条件を満たします。この条件を明確にできれば、入力
に伴う労力を大幅に減らすことができます。これがまさにメンバーシップフィルターの役目です。メンバーシップフ
ィルターを使って派生集合のメンバーリストを定義する方法は、できうる集合それぞれに含まれるメンバーは、そ
れに含まれるために満たさなければならぬ論理条件があります。いわゆる、この論理条件が基準に達していないメ
ンバーをフィルターにかける役目をします。

例えば、TRUCKS という集合を定義し、トラックそれぞれには CAPACITY という属性があるとします。そこで、
TRUCKS から許容運搬量が大いものだけをまとめて派生集合を作りたい場合、条件に合うトラックを明確なメンバ
ーリストにしても良いですが、メンバーシップフィルターを使うと次のような簡単な式になります。

```
HEAVY_DUTY( TRUCKS) | CAPACITY( &1) #GT# 50000;
```

この集合を HEAVY_DUTY とし、親集合の TRUCKS からの派生集合とします。縦棒 (|) を使い、メンバーシップフィルターの開始ラインとします。このメンバーシップフィルターでは、運搬許容量を (CAPACITY(&1)) 以上を (#GT#) で表し、運搬許容量が 50,000 以上のトラックのみを HEAVY_DUTY に入れます。フィルターに含まれる「&1」という記号は、集合インデックスと呼ばれます。メンバーシップフィルターを使って派生集合を作る時、LINGO は親集合から全ての組み合わせを作ります。これら全ての組み合わせをフィルターにかけ、条件に達するかどうかを確認します。最初の親集合の値が「&1」に入力され、二番目が「&2」に入力され、三番、四番と同様に入力されます。今回の例では、親集合が一つしかないため、「&2」以降は必要ない。「#GT#」は論理演算子です。LINGO で認識される論理演算子は以下のとおりです。

- ・ #EQ# = 同等
- ・ #NE# ≠ 不等
- ・ #GE# ≥ 以上
- ・ #LT# < 未満
- ・ #LE# ≤ 以下

5.2.3. まとめ

LINGO は、原始と派生という 2 種類の集合を認識できます。原始集合とは、モデルをそれ以上細かくできない基礎的なオブジェクトを指します。逆に、派生集合は他の集合から取り出して部分集合を構成します。元となる集合を派生集合の親と呼び、原始集合または派生集合から構成されています。派生集合は、疎または密に分かれます。密な集合は親集合の組み合わせが全て含まれる（デカルト積または親集合との交差とも呼ばれます）。疎な集合は交差する親集合の一部を部分集合として含み、明示的なリストまたはメンバーシップフィルターのどちらかで定義できます。明示的なリストの場合、疎な集合のメンバーをデータ節にリストします。メンバーシップフィルターは、全てのメンバーが満たさなければならない論理的な条件を定め、メンバーをすっきりとリストします。下記の図は、集合の種類を示しています。

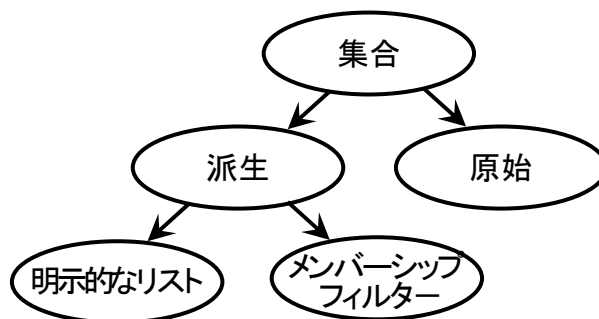


図 5.1 集合の種類

5.3. Data 節（データ節）

集合節では、ある種類の問題のデータ構造を説明しました。データ節は、このような問題の特定な事例を作る為に、データを提供します。データ節は、モデルから変更が生じやすいデータを分離します。データを分離することで、モデルの保守が簡単になり、モデルを拡大・縮小するのが容易になります。

大規模なモデルは、a) 集合節、b)Data 節、c)方程式の三つの部分に区分することが望ましいです。モデルを作る人は、この三つの節を全て把握します。しかし、モデル作成者がきちんと区分していれば、ユーザーは Data 節だけを理解すれば良い訳です。

集合節と同様、Data 節は「DATA:」で始まり、「ENDDATA」で終了します。この Data 節に文を入力して定義したい集合の属性やメンバーを初期化することができます。書式はは下記の通りです。

属性リスト=値のリスト；

もしくは以下のようになります。

集合の名前 = メンバーリスト；

「属性リスト」には、初期化したい属性の名前(カンマで分けてもよい)を入力します。もし、複数の属性の名前が文の左側にある場合、全ての属性は同じ集合に関連していなければなりません。「値のリスト」には、属性リストにある属性に割り当てる値(カンマで分けてもよい)を入力します。次の例を見てみましょう。

```
SETS:  
SET1: X, Y;  
ENDSETS  
DATA:  
SET1 = M1, M2, M3;  
X = 1 2 3;  
Y = 4 5 6;  
ENDDATA
```

ここには、二つの属性 X と Y が SET1 で定義されています。X の値は、1、2、3、Y の値は、4、5、6 です。同じモデルを次のように表現することもできます。

```
SETS:  
SET1: X, Y;  
ENDSETS  
DATA:  
SET1 X Y =  
M1 1 4  
M2 2 5  
M3 3 6;  
ENDDATA
```

これを見ると、X が 1、4、2 のように見えますが、LINGO は Data 節の値を「行」でなく「列」形式で読むので、X の値は1、2、3なります。Data 節は、派生集合のメンバーを特定するのにも使われます。次の例では、どのように Data 節に集合のメンバーを特定するか、またどのように疎な派生集合を特定するかを説明します。必要はないが、この例では VOLUME 属性の値も指定します。

```
SETS:
PRODUCT ;
MACHINE ;
WEEK ;
ALLOWED( PRODUCT, MACHINE, WEEK): VOLUME;
ENDSETS
DATA:
PRODUCT = A B;
MACHINE = M N;
WEEK = 1..2;
ALLOWED, VOLUME =
A M 1 20.5
A N 2 31.3
B N 1 15.8;
ENDDATA
```

ALLOWED には、8 つのメンバー全ては含まれていません。代わりに、(A,M,1)、(A,N,2)と(B,N,1)の 3 つのみが含まれる疎な集合になっています。

LINGO は、多数の標準的な集合を認識します。例えば、Data 節に、

```
PRODUCT = 1..5;
```

と指示すると、PRODUCT のメンバーは、1、2、3、4、5 になります。もしくは、

```
PERIOD = Feb..May;
```

と、指示すると、PERIOD のメンバーは、Feb、Mar、Apr、May になります。他の例の集合には、mon..sun や thing1..thing12 があります。

Data 節に属性が指定されていない場合、デフォルトで決定変数となります。LINGO は、モデルの方程式の節にある文と一致する値を属性とします。

ここでは、Data 節の使い方について簡単に紹介しました、ここで挙げた例のようにデータを Data 節に置かなくてもかまいません。Data 節は、Excel の OLE リンク、データベースの ODBC リンクそしてテキストベースのデータファイルを読み込むことができます。その例は、後ほど紹介します。

LINGO が派生集合を作る時、一番早く増加するのは一番右側の親集合です。

5.4. 集合ループ関数

モデルの方程式の節は、色々な属性の関係を示します。集合節や Data 節にない文は、デフォルトで方程式節にあります。集合をベースとしたモデル構築の強みは、一つの文を使って、集合内の全てのメンバーに作業を適用する能力にあります。LINGO で、これを可能にする関数を集合ループ関数と呼びます。この機能を使わないのは、労力の損失です。集合ループ関数を用いると、ある演算を行うため全メンバーに対して反復します。LINGO には現在 4 つのループ関数があります。

| 関数 | 用途 |
|------|-----------------------|
| @FOR | 集合に含まれる全てのメンバーに制約を課す。 |
| @SUM | 全ての集合メンバーに対して和の計算を行う。 |
| @MIN | 全ての集合メンバー上で最小値を計算する |
| @MAX | 全ての集合メンバー上で最大値を計算する。 |

集合ループ関数の書式は次の通りである。

@ループ関数(集合名 [(集合索引リスト) [|条件文の限定詞]] :
式のリスト);

「@ループ関数」には、上記の表から一つを適切なものを選んで入力します。「集合名」とは、ループさせたい集合を入力します。「集合索引リスト」は、オプションで集合名に指定した親集合である原始集合に対応する索引リストです。LINGO は、「集合名」に指定した集合のメンバー間をループしつつ、作られた索引に「集合名」で指定した集合の現メンバーに対応する値を設定します。「条件文の限定詞」もオプションで使えるフィルターで、集合ループ関数の範囲を限定するのに使えます。LINGO が「集合名」に指定した集合のメンバーをループする時、条件文の限定詞をチェックします。結果が true の場合、「式のリスト」がそれぞれのメンバーに対して実行されます。そうでなければ、省略されます。「式のリスト」とは、「集合名」に指定した集合に含まれるメンバーに適用される方程式のリストを指します。@FOR 関数を使う時には、この式のリストにセミコロン(;)で区切られた複数の式が含まれます。これらの式は、モデルの制約式として追加されます。この他の集合ループ関数(@SUM, @MAX, @MIN)を使う時には、式のリストに含まれる式が一つでなければなりません。もし、集合索引リストを省略した場合、索引リストを参照する全ての属性を「集合名」に指定する集合の中に定義します。

5.4.1. @SUM 集合ループ関数

この例では集合ループ関数の一般的な機能と @SUM 関数の説明をするために、@SUM 関数を使って総和を作ります。

次のモデルを見てみましょう:

```
SETS:
SET_A : X;
ENDSETS
DATA:
SET_A = A1 A2 A3 A4 A5;
X = 5 1 3 4 6;
ENDDATA
X_SUM = @SUM( SET_A( J) : X( J));
```

LINGO は、先ず内部のアクキュレーターを 0 に初期化します。次に、SET_A のメンバー間をループします。「索引変数 J」は、先ず SET_A の最初のメンバー (例えば A1) を設定し、次に X(A1) をアクキュレーターに追加します。次に、J を 2 番目の要素に設定し、この過程を X の値全てがアクキュレーターに追加されるまで続けます。合計の値は、X_SUM に保存されます。

式のリストにある全ての属性 (この例の場合、X が式のリストにある) が索引集合 (SET_A) に定義されています。このような場合、総和を次のように書くこともできます。

```
X_SUM = @SUM( SET_A: X );
```

この場合、余計な索引集合リストと X の索引がなくなっています。使われる式がこれだけ簡単明瞭な場合、索引リストは「暗示」されているといえます。このような索引リストは、式のリストにある属性が、違う親集合から参照されている場合は使えません。

次に、属性 X にある要素の最初の三つの和を求めたい場合を想定します。集合索引の条件文の限定詞を使って、次のように遂行します。

```
X3_SUM = @SUM( SET_A( J ) | J #LE# 3: X( J) );
```

#LE# は論理演算子で、左側の集合索引 (J) を右側のもの (3) と比べ、左側の集合索引が右側より以下の場合 (真)、それ以外の場合「偽」を返します。そして LINGO が和を求める場合、集合の索引変数 J を条件文の限定詞である J#LE#3 に埋め込みます。もし真であれば、X(J) が和に加えられます。

結果的に、LINGO は X の最初の 3 つの項を足し、4 番目と 5 番目の項を省略し、答えは 9 となります。

この例を終える前に、ちょっとした注意点があります。それは、最後の集合の索引 J が返した値です。J #LE# 3 を使って、索引変数と 3 を比べます。これが意味をもつためには、J は数値でなくてはなりません。集合の索引は、集合のメンバー間をループするのに使われるため、集合索引はただの現集合メンバーの代替物であると考えられます。ある意味そうなのであるが、集合索引が返すのは現集合メンバーの親原始集合での「索引」です。言い換えると、最初のメンバーに対する索引は 1 で、2 番目は 2... という値が返されます。このように、集合の索引が数値を返すということを考えれば、他の変数と共に算術式に使うことができます。

5.4.2. @MIN と @MAX ループ関数

@MIN と @MAX 関数は、ある集合のメンバーの最小値と最大値を求める時に使います。次のモデルを見てみましょう。

```
SETS:
SET_A : X;
ENDSETS
DATA:
SET_A = A1 A2 A3 A4 A5;
X = 5 1 3 4 6;
ENDDATA
```

X の最小値と最大値を求めるには、次の二つの式を加えるだけである。

```
THE_MIN_OF_X = @MIN( SET_A( J ): X( J) );
THE_MAX_OF_X = @MAX( SET_A( J ): X( J) );
```

上記の例と同様に、索引集合に属性が設定されているため、暗示的な索引リストを使うことができます。暗示的な索引を使うと、式を再計算できます。

```
THE_MIN_OF_X = @MIN( SET_A: X );
THE_MAX_OF_X = @MAX( SET_A: X );
```

どちらにしても、このモデルを解くために、LINGO は X の期待した最小と最大値を返します。

| 変数 | 値 |
|--------------|----------|
| THE_MIN_OF_X | 1.000000 |
| THE_MAX_OF_X | 6.000000 |

説明のため、X の最初の三つの要素の最小と最大の値を求めると仮定しましょう。@SUM の例の時のように、条件文の限定詞である J #LE# 3 を加えるだけです。

```
THE_MIN_OF_X_3 = @MIN( SET_A( J ) | J #LE# 3: X( J) );
THE_MAX_OF_X_3 = @MAX( SET_A( J ) | J #LE# 3: X( J) );
```

解は次にまとめました。

| 変数 | 値 |
|----------------|----------|
| THE_MIN_OF_X_3 | 1.000000 |
| THE_MAX_OF_X_3 | 5.000000 |

5.4.3. @FOR 集合ループ関数

@FOR 関数は、特定の集合のメンバーに制約を付ける場合に使われる。スカラーをベースとしたモデル言語は、各制約をそれぞれ明確に入力するのに対し、@FOR 関数は制約を一度入力し、後は LINGO がメンバーそれぞれに対応する制約を生成します。@FOR は集合をベースとしたモデルの強力なツールです。

次の例を見てください。

```
SETS:
TRUCKS : HAUL;
ENDSETS
DATA:
TRUCKS = MAC, PETERBILT, FORD, DODGE;
ENDDATA
```

明確に言うと、HAUL と名づけられた属性と原始集合として四台のトラックがります。もし、属性 HAUL がトラックが運べる量を意味するならば、次のように @FOR を使って運べる量を 2,500 ポンドに制約できます。

```
@FOR( TRUCKS( T): HAUL( T) <= 2500 );
```

この場合、LINGO が作った制約を見ても有益です。Windows の場合、LINGO|Generate コマンドを使います。他のプラットフォームの場合は GENERATE コマンドを使います。このコマンドで、LINGO が次の四つの制約を作るのが判ります。

```
HAUL( MAC) <= 2500;
HAUL( PETERBILT) <= 2500;
HAUL( FORD) <= 2500;
HAUL( DODGE) <= 2500;
```

予想した通り、各トラック毎に一つの制約を作りました。

@FOR (太字部分) を使って、属性である GPM にある五つの数字の逆数を求めるモデルは次のようになります。

```
SETS:
OBJECT: GPM, MPG;
ENDSETS
DATA:
OBJECT = A B C D E;
GPM = .0303 .03571 .04545 .07142 .10;
ENDDATA
@FOR( OBJECT( I ):
MPG( I) = 1 / GPM( I)
);
```

このモデルを解くと、逆数として次の値が得られます。

| 変数 | 値 |
|--------|----------|
| MPG(A) | 33.00330 |
| MPG(B) | 28.00336 |
| MPG(C) | 22.00220 |
| MPG(D) | 14.00168 |
| MPG(E) | 10.00000 |

ゼロの逆数は定義されていないため、0 に出会ったら、その値をとばすように条件文の限定詞を @FOR に入力します。そのために @FOR 文は次のようになります。

```
@FOR( OBJECT( I) | GPM( I) #NE# 0:
MPG( I) = 1 / GPM( I)
);
```

条件文の限定詞(太字部分)は、GPM が 0 と同等でない(#NE#)ことを確かめます。もし 0 でないならば、計算を続けます。

以上は、@FOR の簡単な説明ですが、この後の節で色々な例題を示します。

5.4.4. ネスト化した集合ループ関数

前節で紹介した簡単なモデルは、@FOR を一つの集合に対応させていました。大きなモデルの場合、他の集合ループ関数にある集合ループ関数を対応させなくてはならないかもしれません。一つの集合ループ関数が他のものの範囲にある場合、それをネスティングと呼びます。LINGO では、ネスティングすることができます。

次の例は、@SUM ループが @FOR にネスティングした例です。

```
! The demand constraints;
@FOR( VENDORS( J ):
@SUM( WAREHOUSES( I ): VOLUME( I, J) ) = DEMAND( J ););
```

各ベンダー(VENDERS)の需要量と、倉庫(WAREHOUSE)からベンダーに輸送される量の和を等しくします。

@SUM、@MAX、@MINは、どの集合ループ関数にもネスティングできます。しかし、@FORは@FORにのみネスティングが可能です。

5.5. 集合をベースにしたモデルの例

LINGO で作られる四つの集合を思い出して下さい。

- ・ 原始集合
- ・ 密な派生集合
- ・ 疎な派生集合－明示的なリスト
- ・ 疎な派生集合－membership filter

この節では上記四種類のモデルの構築と説明を通して、集合ベースのモデリングの向上を目指します。

5.5.1. 原始集合の例

この要員配置モデルでは、原始集合の使い方を説明します。このモデルは、LINGO のメインディレクトリにあります。SAMPLES というサブディレクトリに STAFFDEMO.LNG というファイル名前で保存されています。

問題

“Pluto Dogs” という 7 日間毎日営業する人気ホットドッグ店を経営すると仮定します。要員は、5 日間労働し、必ず 2 日間連続で休日を与えるという条件で雇用します。各要員の週給は同じとします。忙しい曜日とそうでない曜日があり、過去の経験からどの日により多くの労働力が必要かはわかっています。予想では、下記の人数が各曜日に必要です。

| 曜日 | 月 | 火 | 水 | 木 | 金 | 土 | 日 |
|-------|----|----|----|----|----|----|----|
| 必要要員数 | 20 | 16 | 13 | 16 | 19 | 14 | 12 |

必要な人数を下回ることなく労働力を最低限に抑えるために、最初に働く各曜日の初めに何人必要かを決める必要があります。

定式化

集合ベースのモデルを作るためにまず考えなければいけないことは、「どれが関連する集合で、その属性は何か」ということです。このモデルには、「曜日(DAYS)」という一つの原始集合しかありません。DAYS には、二つの属性があります。一つ目は、各曜日に必要な要員の人数(REQUIRED)で、もう一つの決定変数は、各曜日の開始時の要員の人数(START)です。これで集合と Data 節を書き始める事ができます。

```
SETS:
  DAYS : REQUIRED, START;
ENDSETS
DATA:
  DAYS = MON TUE WED THU FRI SAT SUN;
  REQUIRED = 20 16 13 16 19 14 12;
ENDDATA
```

モデルの数学的な関係（目的と制約）を入力するところまでできました。目的関数を書くところから始めましょう。ここでは、要員の人数を最小化することが目的となります。これを数学的に表記すると次のようになります。

$$\text{Minimize : } \sum_i \text{START } i$$

LINGO の文に置き換えると、数学的な表記に似ていることがわかります。「MIN=」で「Minimize」を置き換え、「 \sum_i 」を「@SUM(DAYS(I):)」で置き換えるだけです。LINGO の式を書いてみましょう。

$$\text{MIN} = \text{@SUM(DAYS(I): START(I));}$$

これで、残りは制約のみです。このモデルには、一つの制約しかありません。それは、毎日必要な要員数を下まわってはいけないということです。書き換えてみると、次のようになります。

各曜日： 本日出勤の要員の数 \geq 一日に必要な人数

この式の右側にある必要な要員数はわかっています。それは単に REQUIRED (I) です。左側の「本日の出勤の要員数」を割り出すのは多少困難です。要員は「5 日間働いて 2 日間連続して休日をとる」ため、それを計算する式は次のようになります。

$$\begin{aligned} \text{本日の出勤人数} = & \text{今日から働き始める人数} + \text{昨日から働き始めた要員数} + \\ & \text{2 日前に働き始めた要員数} + \text{3 日前に働き始めた要員数} + \\ & \text{4 日前に働き始めた要員数} \end{aligned}$$

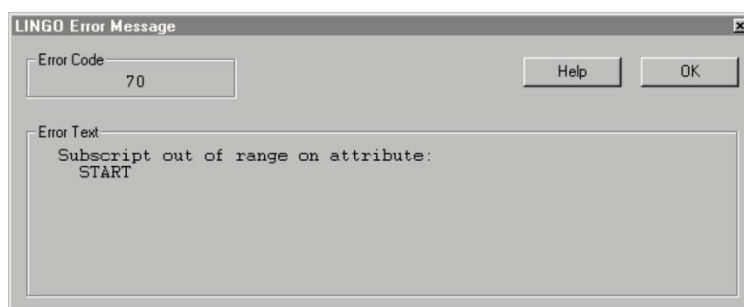
今日働いている人数を計算するには、今日から働き始める人数と4日前まで働いていた人数を足した数を計算します。5日前から働き始めた人数と6日前から働き始めた人数は、休日にあたるため数えません。数学的な表現をすると下記のようにになります。

$$\sum_{i=j-4}^j \text{START } i \geq \text{REQUIRED } j, \quad \text{for } j \in \text{DAYS}$$

これを LINGO 形式に直すと、次のようになります。

$$\begin{aligned} & \text{@FOR(DAYS(J) :} \\ & \text{@SUM(DAYS(I) | I \#LE\# 5: START(J - I + 1))} \\ & \text{>= REQUIRED(J));} \end{aligned}$$

これを言葉で言い表すと、各曜日 5 日間に働く人数として、4 日前から今日までに働き始める要員数が必要最低人数と同じ、もしくはそれ以上になるという意味になります。LINGO の文は、4 日前に働き始めた要員から当日までの 5 日間の合計が、当日に要求されている要員数より多いか等しいということなので、一見正しいように見えますが、このモデルをこの制約で解析しようするとエラーメッセージが表示されてしまいます。



なぜこのエラーメッセージが出るのかを理解するために、木曜日に何が起きているかを考えてみましょう。木曜日は、DAYS の索引で「4」とされています。要員を配置する上で木曜日の制約は次のように表されます。

$$\begin{aligned} & \text{START(4 - 1 + 1) + START(4 - 2 + 1) +} \\ & \text{START(4 - 3 + 1) + START(4 - 4 + 1) +} \\ & \text{START(4 - 5 + 1) >= REQUIRED(4);} \end{aligned}$$

簡略化すると次のようになります。

```
START ( 4 ) + START ( 3 ) +
START ( 2 ) + START ( 1 ) +
START ( 0 ) >= REQUIRED ( 4 ) ;
```

START(0)が問題です。START は曜日 1 から7なので、START(0)は存在しません。よって、0 という索引は「範囲外」と認識されます。

インデックスが 0 以下の場合、週の最後に回します。特に 0 は日曜日(7)、-1は土曜日(6)・・・となります。

LINGO に、これを行う@WRAP 関数があります。

@WRAPS は、INDEX と LIMIT という二つの引数を必要とします。正式には、@WRAP は $J=INDEX-K*LIMIT$ のようになります。ここで、K は J が区間[1,LIMIT] に含まれるような整数です。略して言えば、@WARP は、INDEX に LIMIT を足す、もしくは引くかを、解が1から LIMIT の間になるまで続けます。よって、それは多重周期計画モデルの索引に対応できます。

@WRAP を取り入れると、要員の制約は最終的に次のように修正されます。

```
@FOR ( DAYS ( J ) :
@SUM ( DAYS ( I ) | I #LE# 5 :
START ( @WRAP ( J - I + 1, 7 ) ) ) >= REQUIRED ( J )
);
```

下記が要員配置モデルの全体像です。

```
SETS:
  DAYS : REQUIRED, START;
ENDSETS
DATA:
  DAYS = MON TUE WED THU FRI SAT SUN;
  REQUIRED = 20 16 13 16 19 14 12;
ENDDATA
MIN = @SUM ( DAYS ( I ) : START ( I ) );
@FOR ( DAYS ( J ) :
  @SUM ( DAYS ( I ) | I #LE# 5 :
    START ( @WRAP ( J - I + 1, 7 ) ) ) >= REQUIRED ( J )
);
```

このモデルを解くと、解のレポートが表示されます。

```

Optimal solution found at step:    8
Objective value:                    22.00000
Variable                            Value          Reduced Cost
REQUIRED ( MON)                    20.00000        0.0000000
REQUIRED ( TUE)                    16.00000        0.0000000
REQUIRED ( WED)                    13.00000        0.0000000
REQUIRED ( THU)                    16.00000        0.0000000
REQUIRED ( FRI)                    19.00000        0.0000000
REQUIRED ( SAT)                    14.00000        0.0000000
REQUIRED ( SUN)                    12.00000        0.0000000
START ( MON)                       8.00000        0.0000000
START ( TUE)                       2.00000        0.0000000
START ( WED)                       0.00000        0.0000000
START ( THU)                       6.00000        0.0000000
START ( FRI)                       3.00000        0.0000000
START ( SAT)                       3.00000        0.0000000
START ( SUN)                       0.00000        0.0000000
Row   Slack or Surplus              Dual Price
  1                22.00000             1.000000
  2                 0.0000000            -0.200000
  3                 0.0000000            -0.200000
  4                 0.0000000            -0.200000
  5                 0.0000000            -0.200000
  6                 0.0000000            -0.200000
  7                 0.0000000            -0.200000
  8                 0.0000000            -0.200000
    
```

目的値は 22 で、22 名の要員を雇用しなければならないことを示しています。
スケジュールに沿って要員を配置します。

| | 月 | 火 | 水 | 木 | 金 | 土 | 日 |
|----|---|---|---|---|---|---|---|
| 開始 | 8 | 2 | 0 | 6 | 3 | 3 | 0 |

必須要員数を示す行（行 2-7）の Slack or Surplus を見てみると、全ての曜日において 0 となっている。これは、余分な要員は配置されていないということを意味します。

5.5.2. 密な派生集合の例

この例では、混合モデルにおける密な派生集合の使い方を説明します。この例は、LINGO のメインディレクトリにある SAMPLES サブディレクトリにある CHESS.LNG というファイル名で保存されています。

問題

Chess Snackfoods(CS)社は *Pawn*、*Knight*、*Bishop*、*King* という4タイプのナッツブレンドを取り扱っています。各製品には特定の割合でピーナッツとカシューナッツが混ぜられています。各ナッツの割合と 1 袋ごとの価格が下記の表にまとめられています。

| | Pawn | Knight | Bishop | King | |
|---------------|------|--------|--------|------|----|
| ピーナッツ (oz.) | 15 | 10 | 6 | 2 | |
| カシューナッツ (oz.) | | 1 | 6 | 10 | 14 |
| 価格 (ドル/ポンド) | | 2 | 3 | 4 | 5 |

CS 社は、業者から一日に 750 ポンドのピーナッツと 250 ポンドのカシューナッツを受け取る契約をする。今回の問題は、手持ちのナッツの量を超さずに収益を最大化するには、どのブレンドを何ポンドずつ生産すればよいかということになる。

定式化

このモデルの原始集合は、ナッツの種類とブレンドの種類である。NUTS 集合には、SUPPLY という一つの属性がある。これはナッツの毎日の供給量をポンドで示したものである。BRANDS 集合には、PRICE と PRODUCE という二つの属性がある。PRICE はブランドの売値、PRODUCE は一日に何ポンドのブランドを作るかを定める決定変数である。

ブランドの式を入力するには、もう一つ集合が必要となる。ナッツのタイプとブランドを定義する 2 次元の表が必要となる。これを作るためには、NUTS と BRANDS 集合から派生集合を作らなければいけない。この派生集合を付け加えると集合節が完了する。

SETS:

NUTS : SUPPLY;

BRANDS : PRICE, PRODUCE;

FORMULA(NUTS, BRANDS): OUNCES;

ENDSETS

この派生集合を FORMULA と名づけた。FORMULA には OUNCES という属性があり、ブランドにどのナッツが何オンス使われるかを保存する。しかし、この派生集合のメンバーはまだ指定されていないため、LINGO は全てのナッツのペアとブランドを含む密な集合を作る。

集合の定義ができたので、Data 節に移る。まず、次のように SUPPLY, PRICE, OUNCES を初期化する。

DATA:

NUTS = PEANUTS, CASHEWS;

SUPPLY = 750 250;

BRANDS = PAWN, KNIGHT, BISHOP, KING;

PRICE = 2 3 4 5;

OUNCES = 15 10 6 2 !(Peanuts);

1 6 10 14; !(Cashews);

ENDDATA

データと集合の用意ができたので、目的関数と制約に入る。利益を最大化する目的関数は、比較的簡単である。

MAX = @SUM(BRANDS(I): PRICE(I) * PRODUCE(I));

制約は、一日に供給されるナッツの量以上生産してはならないということのみである。言い変えると、

ナッツの種類 i ごとに、使用されるナッツ i の量(ポンド)は、ナッツ i の供給量以下でなくてはならない。

LINGO 形式で表現すると、次のようになる。

@FOR(NUTS(I):

@SUM(BRANDS(J):

OUNCES(I, J) * PRODUCE(J) / 16) <= SUPPLY(I);

オンスをポンドに変換するために左辺の総和を 16 で割る。完成したモデルは下記のようなになる。

SETS:

NUTS : SUPPLY;

BRANDS : PRICE, PRODUCE;

FORMULA(NUTS, BRANDS): OUNCES;

ENDSETS

DATA:

NUTS = PEANUTS, CASHEWS;

SUPPLY = 750 250;

BRANDS = PAWN, KNIGHT, BISHOP, KING;

PRICE = 2 3 4 5;

OUNCES = 15 10 6 2 !(Peanuts);

1 6 10 14; !(Cashews);

ENDDATA

MAX = @SUM(BRANDS(I):

PRICE(I) * PRODUCE(I));

@FOR(NUTS(I):

@SUM(BRANDS(J):

OUNCES(I, J) * PRODUCE(J)/16) <= SUPPLY(I));

簡略化した解のレポートは、以下のように表現される。

Optimal solution found at step: 0

Objective value: 2692.308

| Variable | Value | Reduced Cost |
|------------------|-----------|---------------|
| PRODUCE(PAWN) | 769.2308 | 0.0000000 |
| PRODUCE(KNIGHT) | 0.0000000 | 0.1538461 |
| PRODUCE(BISHOP) | 0.0000000 | 0.7692297E-01 |
| PRODUCE(KING) | 230.7692 | 0.0000000 |

Row Slack or Surplus Dual Price

| | | |
|---|-----------|----------|
| 1 | 2692.308 | 1.000000 |
| 2 | 0.0000000 | 1.769231 |
| 3 | 0.0000000 | 5.461538 |

解のレポートを見てみると、CS 社は \$ 2692.30 の利益を得るには、769.2 ポンドの Pawn と 230.8 ポンドの King を製造するのがよいことがわかる。さらに、Dual Price(双対価格)列を見てみると、CS 社

は 1 ポンドのピーナッツなら \$ 1.77 まで、カシューナッツなら \$ 5.46 までなら追加でナッツを購入してもよいという結果が出ている。マーケティング的な理由により、少なくともある程度の Knight と Bishop を生産しなければならない場合、減少費用の列を見てみると、Knight を最初の 1 ポンド生産すると \$ 15.4、Bishop を最初の 1 ポンド生産すると \$ 7.69 利益が減少することがわかる。

5.5.3. 疎な派生集合—明示的なリストの例

この例では、疎な派生集合の明示的なリストの使い方を説明する。この方法を使って疎な集合を定義する場合、集合に属する全てのメンバーを直接入力する。それらは大抵、親集合のデカルト積全部から作られる小さくて密な部分集合である。

例として、新製品を展開するプロジェクトに関わるタスクのクリティカルパスを決定する PERT (Project Evaluation and Review Technique) モデルを考えてみよう。PERT とは、1950 年代に開発された大きなプロジェクトの進行具合を管理し、責任者達をサポートするための簡単で強力な技法である。公式に適用された初めてのアプリケーションは、「ポラリス・プロジェクト」という弾道ミサイル積載潜水艦プロジェクトでした。Craven (2001)によると、PERT はポラリス・プロジェクトのキーパーソンであった Admiral William F. Raborn という人に名づけられました。Raborn には、PERT というニックネームの新妻がいました。その新妻に敬意を表して、ポラリス・プロジェクトをモニターする管理システムを PERT としたのである。実際、ポラリス・プロジェクトは予定より 18 ヶ月も早く完了した。PERT は特に遅れたら全体の進行に響いてしまうような重要な仕事の確認に便利である。この全体に響いてしまうようなタスクをクリティカルパスと呼ぶ。長期に渡る大プロジェクトの動きを理解することは、プロジェクトが目的から脱線したり、遅延の可能性を大幅に減らすことに繋がる。PERT や PERT に似た CPM(クリティカルパス法)は、今後も多様なプロジェクトの成功に役立つでしょう。この例は、LINGO メインディレクトリの SAMPLES フォルダに PERT.LNG というファイル名で保存されている。

問題

Wireless Widgets (WW) 社が Solar Widget という新製品を売り出そうとしている。発売予定日に遅れがでないよう、WW 社はその予定日までのタスクを PERT で分析したいと考えている。その分析により、時間内にしておかなければ遅延が発生する仕事—クリティカルパス—を確認することができ、Solar Widget を予定通りに売り出すことが出来る。それらの仕事は、発表するまえに完了していなければいけない。それらの予定完了日は、下記の通りである。

| 仕事 | 週 |
|----------|----|
| デザインの仕上げ | 10 |
| 需要の予測 | 14 |
| 競合の調査 | 3 |
| 価格設定 | 3 |
| 生産工程の計画 | 7 |
| 費用の見積 | 4 |
| 販売員の訓練 | 10 |

特定のタスクは、他のタスクが始まる前に終わらせなければいけない。下記に、優先順位を表す関係図がある。

図 5.1 製品発売に向けての優先順位関係

例えば、「需要の予測」から出ている2つの矢印は「生産工程の計画」と「価格設定」より前に「需要の予測」を終わらせなければならないことを示す。

目的は、Solar Widgets の発売に向けて PERT モデルを使い、タスクのクリティカルパスを特定することである。

定式化

このプロジェクトを表現するには、原始集合が必要である。TASKS という集合に四つの属性を結び付ける。

TIME : タスクを完了させる時間 (既知)

ES : タスクを開始する可能な限り早い時間 (計算が必要)

LS : タスクを開始する最遅の時間 (計算が必要)

SLACK : そのタスクの LS と ES の差 (計算が必要)

もし、タスクの SLACK が 0 ならば、そのタスクは決められた日時に開始しないとプロジェクト全体が遅れることを意味する。このように SLACK が 0 のタスクがクリティカルパスである。

タスクの開始時を計算するには優先関係を考える必要があるため、モデルに優先順位の関係を入力する。この優先順位関係をタスクの順序対のリストとする。例えば、DESIGN が FORECAST の前に終わっていなければならないという関係を表す場合、(DESIGN,FORECAST) となる。TASKS 集合に 2 次元の派生集合を作る事によって、優先順位の関係を入力できる。よって、Data 節は次のようになる。

DATA:

TASKS : TIME, ES, LS, SLACK;

PRED(TASKS, TASKS);

PRED には、属性がないことに留意して下さい。目的は、タスク間の優先順位の関係を示すだけである。次に、タスクの時間と順序対を Data 節に入力すると、次のようになる。

DATA:

TASKS= DESIGN, FORECAST, SURVEY, PRICE, SCHEDULE, COSTOUT, TRAIN;

TIME = 10, 14, 3, 3, 7, 4, 10;

PRED =

DESIGN,FORECAST,

DESIGN,SURVEY,

FORECAST,PRICE,

FORECAST,SCHEDULE,

```
SURVEY,PRICE,
SCHEDULE,COSTOUT,
PRICE,TRAIN,
COSTOUT,TRAIN;
ENDDATA
```

PRED の最初のメンバーは、順序対 (DESIGN, FORECAST)であり、DESIGN という一つのタスクではないということを忘れないで下さい。よって、この集合には、合計で八つのメンバーが入る。それぞれは、優先順位の関係を示す図のアーキに対応する。

この例から学んでおかなければならないのは、PRED が明示的なリストを使った疎な派生集合であるということである。この集合は、TASKS と PRED からできたサブセットである。疎であるのは、49 あるメンバーの内、八つのみでできた集合であり、明示的なリストであるのは、その集合に入りたいメンバーをリストしたからである。明示的に疎な集合のメンバーのリストを作るのは、数千ものメンバーから選ばなくてはならない場合、便利ではない。しかし、メンバーの集合条件がはっきりと定義されておらず、密なものよりも疎な集合の規模が小さい場合に意味がある。

集合とデータができあがった。次は式を作ってみよう。まず、一番早い開始時 (ES)、一番遅い開始時 (LS)、その時間の差 (SLAC)を計算しなければならない。これらの時間が分かれば、SLACK はその差を計算するだけであるので、問題は ES と LS の計算である。まずは、ES を計算するための式から始めてみよう。タスクは、それ以前に完了していなければならないタスクが全て終わっていなければ始められない。ということは、このタスク以前のタスク全ての一番遅い完了時間が分かれば、このタスクの一番早い開始時間が分かる。よって、簡単に言うと、タスク t の一番早い開始時間は、t 以前のタスク全ての一番早い開始時間にその完了時間を足したものの和になる。LINGO の表現で表すとこうなる。

```
@FOR( TASKS( J) | J #GT# 1:
```

```
ES( J) = @MAX( PRED( I, J): ES( I) + TIME( I));
```

最初のタスクには、それより前のタスクがないため、J#GT#を加えてタスク 1 の計算を省く。

LS の計算は、手順を逆に考えるという以外は ES と似ている。要は、タスク t の一番遅い開始時間が最小となり、その後のタスク j で、j の一番遅い開始時間から t を完了するのにかかる時間を引いた時間となる。これを LINGO で表現するとこうなる。

```
@FOR( TASKS( I) | I #LT# LTASK:
```

```
LS( I) = @MIN( PRED( I, J): LS( J) - TIME( I));
```

最後のタスク以後にタスクはないので、最後のタスクの計算を省く。

SLACK は、LS と ES の差なので、次のように書くことができる。

```
@FOR( TASKS( I): SLACK( I) = LS( I) - ES( I));
```

タスク 1 の開始時間を任意の値に設定する。今回は 0 とする。

```
ES( 1) = 0;
```

最後のタスクの一番遅い開始時間を除いて全ての変数の値を計算する式ができた。最後のプロジェクトが、その一番早い開始時間よりも遅く開始するとプロジェクト全体が遅れることがわかる。ということは、当然一番最後のプロジェクトは一番早い開始時間と一番遅い開始時間が同じでなければいけない。LINGO の式で表すと次のようになる。

LS(7) = ES(7);

これでもいいが、これは関係を示す一般的な方法ではない。いくつかのタスクをモデルに追加するとすると、この式の「7」をタスクの数を表す新しい数字と置き換えなければいけない。LINGO の集合ベースのモデリング言語は、式に影響なくデータを変更できるというのが目的なので、これではデータの独立性を犯す。よって、次のように書き換えてみました。

LTASK = @SIZE(TASKS);

LS(LTASK) = ES(LTASK);

@SIZE 関数は集合の大きさを返してくれる。この場合、予想通り 7 という解になる。しかし、タスク数を変更した場合、@SIZE は新しい正確な値を返す。このように、モデルの構造のデータ独立性を保つことができる。

PERT 全体と解を下記に記載した。

SETS:

TASKS : TIME, ES, LS, SLACK;

PRED(TASKS, TASKS);

ENDSETS

DATA:

TASKS= DESIGN, FORECAST, SURVEY, PRICE, SCHEDULE, COSTOUT, TRAIN;

TIME = 10, 14, 3, 3, 7, 4, 10;

PRED =

DESIGN,FORECAST,

DESIGN,SURVEY,

FORECAST,PRICE,

FORECAST,SCHEDULE,

SURVEY,PRICE,

SCHEDULE,COSTOUT,

PRICE,TRAIN,

COSTOUT,TRAIN;

ENDDATA

@FOR(TASKS(J) | J #GT# 1:

ES(J) = @MAX(PRED(I, J): ES(I) + TIME(I))

);

@FOR(TASKS(I) | I #LT# LTASK:

LS(I) = @MIN(PRED(I, J): LS(J) - TIME(I));

);

@FOR(TASKS(I): SLACK(I) = LS(I) - ES(I));

ES(1) = 0;

LTASK = @SIZE(TASKS);

LS(LTASK) = ES(LTASK);

この解の興味深いところは次の点である。

| Variable | Value |
|-----------------|-----------|
| LTASK | 7.000000 |
| ES(DESIGN) | 0.000000 |
| ES(FORECAST) | 10.000000 |
| ES(SURVEY) | 10.000000 |
| ES(PRICE) | 24.000000 |
| ES(SCHEDULE) | 24.000000 |
| ES(COSTOUT) | 31.000000 |
| ES(TRAIN) | 35.000000 |
| LS(DESIGN) | 0.000000 |
| LS(FORECAST) | 10.000000 |
| LS(SURVEY) | 29.000000 |
| LS(PRICE) | 32.000000 |
| LS(SCHEDULE) | 24.000000 |
| LS(COSTOUT) | 31.000000 |
| LS(TRAIN) | 35.000000 |
| SLACK(DESIGN) | 0.000000 |
| SLACK(FORECAST) | 0.000000 |
| SLACK(SURVEY) | 19.000000 |
| SLACK(PRICE) | 8.000000 |
| SLACK(SCHEDULE) | 0.000000 |
| SLACK(COSTOUT) | 0.000000 |
| SLACK(TRAIN) | 0.000000 |

タスクの Slack を見てみると、SURVEY と PRICE はそれぞれ 19 と 8 になっている。SURVEY と PRICE の開始時に遅れがでて、この Slack の分だけの遅れならばプロジェクト全体の完了予定日を遅らせることはない。DESIGN、FORECAST、SCHEDULE、COSTOUT と TRAIN は、SURVEY や PRICE は、SURVEY と PRICE と違い、Slack が 0 である。これらがプロジェクトのクリティカルパスであり、開始が遅れるとプロジェクトの完了日に遅れが発生する。プロジェクト責任者は、こういったクリティカルパスが、決められた日時に開始し、予定した時間内で完了するよう気を配らなければいけない。最後に、ES(TRAIN)の値が 35 であるということは、この新 Solar Widget プロジェクトの開始まで 45 週間かかることを示す。35 週間ではなく、45 週間なのは、要員のトレーニング開始まで 35 週間かかり、トレーニング自体が 10 週間なので、合計 45 週間かかるということを示す。

5.5.4. メンバーシップフィルターを使った疎な派生集合

ここでは、メンバーシップフィルターを使った疎な派生集合を説明する。メンバーシップフィルターとは、派生集合を定義する 3 つ目の方法である。この方法を使って集合を定義する場合、各メンバーが満たさなければならない論理条件を指定する。この条件は、それを満たさないメンバーを除外する時に

使う。

マッチング問題を使って説明しよう。最低限の費用で組み合わせたい N 個のオブジェクトがある。これは、ルームメイトの選択問題とも呼ばれている。この問題は、新入生を寮に割り当てるために、大学が毎年度始めに直面する問題である。ペアである (I, J) は、 (J, I) と区別される。よって、任意で I は J よりも少ないとする。正式には、 I と J が順序対でなければいけない。言い換えれば、 J よりも少ない I 以外、不必要な対は作りたくない。この「 J より小さい I 」というのがメンバーシップフィルターになる。

この例は、LINGO メインディレクトリの SAMPLES フォルダにファイル名 MATCHD.LNG で保存されている。

問題

ある企業の戦略計画部を任せられ、その部には 8 人のアナリストがおり、部はもうすぐ新しいオフィスに移動するとする。そのオフィスには合計 4 つの部屋があるので、8 人のアナリストを 4 組に分けることにした。観察の結果、誰と誰を組ませると効率がよいかかわっている。部の和が乱れないよう、相性の悪いアナリストを同室にするのは最小限に抑えなければいけない。これを実現するために不一致度の採点システムを編み出した。採点は 1 から 10 段階あり、組み合わせの点が 1 の場合は相性がすばらしく良い、10 の場合は論争が起こることが考えられるため、オフィスから鋭利なものを片付けなければいけない。下記が点数の一覧表である。

| Analysts | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 |
|----------|---|---|---|---|---|---|---|---|
| 1 | - | 9 | 3 | 4 | 2 | 1 | 5 | 6 |
| 2 | - | - | 1 | 7 | 3 | 5 | 2 | 1 |
| 3 | - | - | - | 4 | 4 | 2 | 9 | 2 |
| 4 | - | - | - | - | 1 | 5 | 5 | 2 |
| 5 | - | - | - | - | - | 8 | 7 | 6 |
| 6 | - | - | - | - | - | - | 2 | 3 |
| 7 | - | - | - | - | - | - | - | 4 |

アナリスト「 I と J 」、 $(J$ と $I)$ の組み合わせは区別できないため、表の対角線より上部のみを表示した。この問題は、不一致度が最も低いアナリスト同士の組み合わせを探すことである。

定式化

まずは、8 人のアナリストを集合にする。これは原始集合で、下記のように簡単に書き表せます。

ANALYSTS;

最後に作りたい集合は、可能な限り全ての組み合わせを含む集合である。これは ANALYST 同士が交差する派生集合である。始めのパスとして密な派生集合を作る。

PAIRS(ANALYSTS, ANALYSTS);

しかし、この集合には $PAIRS(I, J)$ と $PAIRS(J, I)$ が含まれている。同じペアは 2 つも要らない。さらに、この集合だと同じアナリスト同士 (I, I) など) も含んでいる。個人で部屋を持つことがベストであるが、これは不可能である。よって、派生集合にメンバーシップフィルターをかけ、各 (I, J) ペアを最終集合に入れるため、 J は I より大きいという条件を課さなければいけない。これを次のように表す。

PAIRS(ANALYSTS, ANALYSTS) | #GT# &1;

メンバーシップフィルターは縦線 (|) で始まる。Filter 内の「&1 と &2」は索引の代わりである。

これは、メンバーシップフィルター内のみでしか使えない。LINGO がそのまま PAIRS 集合を作ると、ANALYSTS の集合同士が重なる全ての組み合わせを作り出すので、そこにメンバーシップフィルターを付け、条件をクリアすかどうかを試す。特に、ANALYSTS 同士の集合に含まれる各ペア (I, J) は、I を &I と置き換え、J を &2 とし、フィルターにかけられる。結果が真ならば PAIRS 集合に追加される。表で見ると、対角線より上部の (I, J) の組み合わせのみが表示されている。

PAIRS の二つの属性についても考えなければいけない。まずは、ペアの不一致度と関連のある属性が必要となる。そして、アナリスト J と I がペアになったことを示す属性が必要である。これらの属性を RATING と MATCH と呼ぶことにする。それらの属性を次のように PAIRS 集合につけてみましょう。

PAIRS(ANALYSTS, ANALYSTS) | &2 #GT# &1: RATING, MATCH;

Data 節を使い、上記の不一致度の RATING 属性を初期化した。

DATA:

ANALYSTS = 1..8;

RATING =

9 3 4 2 1 5 6

1 7 3 5 2 1

4 4 2 9 2

1 5 5 2

8 7 6

2 3

4;

ENDDATA

アナリストの組み合わせが I と J になった場合 MATCH(I, J) を 1、それ以外は 0 を出力する。こうすることで、MATCH 属性はこのモデルの決定変数となる。

目的関数は、最終的な組み合わせの中で不一致度を最小化することである。これは、RATING と MATCH 属性の内積で、次のように式に表すことができる。

MIN = @SUM(PAIRS(I, J):

RATING(I, J) * MATCH(I, J));

このモデルには 1 つの制約があり、言葉で表すと「アナリストは、各自別のアナリストとペアを組まなければならない」となる。この制約を LINGO の式にすると、次のようになる。

@FOR(ANALYSTS(I):

@SUM(PAIRS(J, K) | J #EQ# I #OR# K #EQ# I:

MATCH(J, K) = 1

);

この制約で興味深いのは、@SUM の conditional qualifier である J # EQ # I # OR # K # EQ # I である。全てのアナリスト I に対して、I を含む全ての MATCH 変数を合計し、それを 1 にする。その際、I と組むアナリストがもう 1 人の別のアナリストであることを確定できる。この conditional qualifier は、I を含む MATCH 変数のみを合計するように指定する。

このモデルに必要な機能がもう 1 つある。J と I が組み合わせられる場合、MATCH(I, J) が 1 になるように、そしてそれ以外は 0 となるように設定されているが、LINGO は変数の範囲を制限しないと範囲が 0 から無限大の任意の値になりえる。MATCH を 0 か 1 に制限したいため、このモデルにもう一つ機能を加えなければいけない。それは、MATCH 属性に @BIN 変数範囲関数を適用することである。変数範囲関数は、変数の値を制限するのに使う。制約とは違い、変数範囲関数はモデルに式を加える必要がない。@BIN 関数が変数を 2 進法 (例えば 0 や 1) に絞る。モデルに 2 値変数が含まれる場合、そのモデルは IP モデルという。IP モデルは連続型変数のみを含むモデルよりも解くのがかなり困難である。適当に作られた IP モデル (数百以上の整数変数を含む) は、永遠に解を得ることができません。よって、できるだけ 2 値変数の使用は避けた方がよいでしょう。MATCH に属する全ての変数に @BIN を適用するには、次のように @FOR 関数を追加する。

```
@FOR( PAIRS( I, J): @BIN( MATCH( I, J)));
```

この例の式全体と解の一部を下記に記した。

SETS:

ANALYSTS;

PAIRS(ANALYSTS, ANALYSTS) | &2 #GT# &1:

RATING, MATCH;

ENDSETS

DATA:

ANALYSTS = 1..8;

RATING =

9 3 4 2 1 5 6

1 7 3 5 2 1

4 4 2 9 2

1 5 5 2

8 7 6

2 3

4;

ENDDATA

MIN = @SUM(PAIRS(I, J):

RATING(I, J) * MATCH(I, J));

@FOR(ANALYSTS(I):

@SUM(PAIRS(J, K) | J #EQ# I #OR# K #EQ# I:

MATCH(J, K) = 1

);

@FOR(PAIRS(I, J): @BIN(MATCH(I, J)));

解は次のようになった。

| Variable | Value |
|--------------|-----------|
| MATCH(1, 2) | 0.0000000 |

```
MATCH( 1, 3) 0.0000000  
MATCH( 1, 4) 0.0000000  
MATCH( 1, 5) 0.0000000  
MATCH( 1, 6) 1.0000000  
MATCH( 1, 7) 0.0000000  
MATCH( 1, 8) 0.0000000  
MATCH( 2, 3) 0.0000000  
MATCH( 2, 4) 0.0000000  
MATCH( 2, 5) 0.0000000  
MATCH( 2, 6) 0.0000000  
MATCH( 2, 7) 1.0000000  
MATCH( 2, 8) 0.0000000  
MATCH( 3, 4) 0.0000000  
MATCH( 3, 5) 0.0000000  
MATCH( 3, 6) 0.0000000  
MATCH( 3, 7) 0.0000000  
MATCH( 3, 8) 1.0000000  
MATCH( 4, 5) 1.0000000  
MATCH( 4, 6) 0.0000000  
MATCH( 4, 7) 0.0000000  
MATCH( 4, 8) 0.0000000  
MATCH( 5, 6) 0.0000000  
MATCH( 5, 7) 0.0000000  
MATCH( 5, 8) 0.0000000  
MATCH( 6, 7) 0.0000000  
MATCH( 6, 8) 0.0000000  
MATCH( 7, 8) 0.0000000
```

目的値を見てみると、最適なペアリングの不一致度の点数の合計が 6 であることがわかりました。Value 列の「1」を見てみると (1,6)、(2,7)、(3,8)、(4,5)が最適な組み合わせであることがわかる。

5.6. 変数の定義域設定関数

変数の定義域設定関数は、この章のマッチング問題で@BIN を使った時に簡単に説明した。定義域設定関数を使うことで、決定変数になりえる値を制約することができる。下に四つの例を挙げてみました。

@BIN (Y);

@GIN (X);

@BND (100, DELIVER, 250);

@FREE (PROFIT);

@BIN(Y)は、変数 Y を 2 値に制限する。これで、変数 Y は、0 か 1 という値しかなくなる。

@GIN(X)は、変数 X を整数値に限定する。これで X を、0、1、2、...というような値のみに縛ることができる。

@BND()は、単に上限と下限を指定する。例えば、@BND(100, DELIVER, 250)の場合、DELIVER は、[100, 250] の間の数でなければいけない。これには他の表現方法もある。

DELIVER >= 100;

DELIVER <= 250;

LINGO はデフォルトで、全ての決定変数の下限を 0 に設定する。@FREE(PROFIT)のように下限は書き換えることができる。この場合、 $(-\infty, \infty)$ になる。定義域設定関数は、他の制約と同様に@FOR ループで使うことができる。

5.7. LINGO とスプレッドシート

この章では、大規模な問題を解く時に LINGO がどれほど便利かを説明した。モデルを作るのに、一番よく使われるのは、疑うことなくスプレッドシートモデルだと言えるでしょう。いつ、どの方法を使うのが適切なのでしょうか。

スプレッドシートを使つてのモデリングの主な利点を挙げてみました。

- ・優れた報告書の書式設定機能
- ・幅広い層の人々にも理解される分かりやすさ
- ・ワードなどの他の良くできたインターフェース能力

LINGO でモデル構築を行う主な利点も挙げてみました。

- ・様々な意味での柔軟である。
 - ・ 拡張性：集合の大きさを、式のコピーや修正することなく変更が可能（例：事項列、製品、顧客、仕入先、輸送モデルを増やすなど）である。スプレッドシートのように列の数が上限 255 だとか、行の上限が 65536 であるというような上限がないし、式に使える最大文字数が 900 以下でなくてはならないというような制限もない。
 - ・ 疎な集合も表現できる。
 - ・ 監査能力と可視性：LINGO では、完全かつ包括的な形で式を見ることができる。複雑なスプレッドシート上で、モデル式を本当に理解しようとすることは、探偵がするような労力がある。
 - ・ 多次元を簡単に表現できる。スプレッドシートは、2次元は扱えるが、3次元以上は難しい。

ある人は、LINGO とスプレッドシートを合わせて最大の利益を得るでしょう。LINGO にリンクをつけ、自動的にスプレッドシート、データベースや普通のファイルからデータを得たり送ったりできる。Windows の場合、OLE (Object Linking and Embedding)や ODBC (Open Database Connectivity) インターフェースがすでに備わっている。OLE 機能で Excel のスプレッドシートを LINGO に繋ぐには二つのステップがある。

a) スプレッドシートでは、出力先、入力元であるデータ域をそれぞれ LINGO に指定する。これは、スプレッドシート内の指定域をマウスでハイライトし、

[挿入|名前|定義]コマンドで名前を設定し、LINGO に組み込むだけでできる。一番わかりやすい名前は、LINGO のモデル内の参照されるデータ名と同じにすることである。

b) LINGO のモデル内で使われるスプレッドシートから検索される各属性（ベクトル）（例：工場設備能力）は、LINGO の Data 節に次のように書きます。

```
CAPACITY = @OLE('C:\MYDATA.XLS');
```

スプレッドシートに送られるそれぞれの属性（例：輸送量）は、LINGO の Data 節に次のような形で示さなければいけない。

```
@OLE('C:\MYDATA.XLS') = AMT_SHIPPED;
```

Excel のスプレッドシートが一つだけしか開いていない場合、次のように簡略できる。

```
CAPACITY = @OLE();
```

LINGO は、開いているスプレッドシートのみから CAPACITY という範囲を探す。同じモデルを違うスプレッドシートのデータに用いたい場合、スプレッドシートが特定されていないのは、とても便利である。

このスプレッドシートとの連結は、LINGO のモデルを関連を持たせたスプレッドシートの中に埋め込むということで、更に進化させることができる。スプレッドシートを開くと、LINGO のモデルがすぐに使える状態となっているため非常に便利である。下記のスクリーンショットを見てください。これが、スプレッドシートに埋め込まれた輸送モデルの例である。カジュアルユーザーには、これは普通のスプレッドシートと解析ボタンに見えるでしょう。

このデータと結果は、スプレッドシートファイルの最初のタブ/シートに仕分けされている。一見して分からないかもしれないが、LINGO モデルが同じスプレッドシートの他のタブに保存されている。下記を見て下さい。完全に隠されているのは、VBA のプログラムである。このプログラムが、最初のタブにある解析ボタンが押される度に二番目のタブにある LINGO モデルに解析をさせているのである。この例は、xlingtran.xls というファイル名で保存されている。

@OLE()が LINGO のモデルとスプレッドシートを結ぶのに使われているように、@ODBC()は LINGO のモデルと SQL インターフェースをサポートするデータベースを繋ぎ、@TEXT()がシンプルなテキストファイルと LINGO のモデルを繋ぎます。例えば、「myfile.out」というファイルに属性 X の値を送るには、次のようにする。

```
DATA:
```

```
@TEXT('MYFILE.OUT') = X;
```

```
ENDDATA
```

次の文で、X の値を説明文と共にスクリーンに出力する。

```
@TEXT() = 'The value of X=', X;
```

もう一つの LINGO を他のアプリケーション合体させる方法は、サブルーチンコールである。一般的なコンピュータープログラミング、例えば C/C++や Visual Basic では、LINGO の DLL (Dynamic Link Library)を呼び出すことができる。モデルは文字列変数として LINGO の DLL に通される。詳しくは、

LINGO のマニュアルを参照して下さい。

5.8. LINGO とプログラミング

LINGO10 の注目すべき新機能は、プログラミング（ある意味コンピュータープログラミング）かルーピング機能である。この機能の主な利点は、a) モデルで使われるデータの前処理（例：複雑な利益の寄与係数の計算）ができるということと、b) LINGO の標準書式ではなく、解をカスタマイズした形で出力するよう後処理ができる、c) 一度に二つもしくはそれ以上のモデルを解析できることである。この複数のモデルを一度のクリックで解析できる機能は、次の場合に便利である。

1. ある臨界パラメータの効用により利益にどのような違いがでるかを見るパラメータ解析を行う。
2. 目標がヒエラルキーの形をとっている場合に目標計画法問題を解く。
3. 反復しつつ、列生成しながら変数や制約を追加させていくという、追加的なモデル生成を行う。

5.8.1 プログラミングの部分構成

実行可能ステートメントは CALC 節で扱う。

CALC:

! Executable statements;

ENDCALC

計算式は、四つの制御文 (@IFC、@FOR、@WHILE、@BREAK) に遭遇しない限り、CALC 節の上から下に連続的に計算される。

下記は、「IF 条件」の書式である。

@IFC(condition:

! Executable Statements;

@ELSE

! Executable Statements;

@ENDIF

ここには、二つのループの制御文がある。@FOR が既知のサイズをループするようにしてあるのと、

@FOR(set | condition:

! Executable Statements;

);

初期に未知数回ループするように@WHILE がある。

@WHILE(condition:

! Executable Statements;

);

次のようにループを中止することもできる。

@BREAK

出力の文字列を次のような形で書くこともできる。

@WRITE(output list);

「output list」は明示的な文字列、変数、特定の書式の変数、@FORMAT()もしくは行末文字、@

NEWLINE(n)になる。

下記は、@FORMAT 関数のシンタックスである。

@FORMAT(math_expression, field_description).

CALC 節で、参照して解きたい問題は、SUBMODEL として指定されている。

例：

```
SUBMODEL mymodel:
```

```
! Model Statements;
```

```
ENDSUBMODEL
```

@SOLVE を使って、前に CALC 節に定義したサブモデルを解くことができる。

例：

```
@SOLVE( mymodel);
```

Astro-Cosmo 問題の有効フロンティアの計算を使って説明する。

```
! Model to compute efficient frontier;
```

```
SUBMODEL ASTROCOSMO:
```

```
MAX = OBJ;
```

```
OBJ= 20*A + 30*C;
```

```
A <= 60;
```

```
C <= 50;
```

```
A + 2*C <= LABORAV;
```

```
ENDSUBMODEL
```

```
DATA:
```

```
! Number of points to compute in efficient frontier;
```

```
NPTS = 11;
```

```
! Upper limit on labor(lower limit is 0);
```

```
UPLIM = 200;
```

```
ENDDATA
```

```
CALC:
```

```
! Set output level to super terse;
```

```
@SET( 'TERSEO', 2);
```

```
@WRITE(' Labor Profit',@NEWLINE(1));
```

```
! Loop over points on efficient frontier;
```

```
i = 1;
```

```
@WHILE( i #LE# NPTS:
```

```
LABORAV = UPLIM*(i-1)/(NPTS-1);
```

```
! Solve model with new labor availability;
```

```
@SOLVE(ASTROCOSMO);
```

```
! Write the objective value, OBJ, in a field of
```

```
8 characters with 2 digits to the right of decimal point;  
@WRITE(" ", @FORMAT(LABORAV, "8.0f"),  
' ', @FORMAT(OBJ,"8.2f"), @NEWLINE(1));  
i = i + 1;  
); ! End @WHILE loop;  
ENDCALC
```

これで下記が出力される。

Labor Profit

| | |
|-----|---------|
| 0 | 0.00 |
| 20 | 400.00 |
| 40 | 800.00 |
| 60 | 1200.00 |
| 80 | 1500.00 |
| 100 | 1800.00 |
| 120 | 2100.00 |
| 140 | 2400.00 |
| 160 | 2700.00 |
| 180 | 2700.00 |
| 200 | 2700.00 |

LINGO でのプログラミングについて詳しくは、オンラインマニュアル(英語版)を参照してください。

5.9. 練習問題

1. ある学術機関の特定の学期での現状を評価してみようと考えている。評価されるのは、教える教員とそのコース登録した生徒である。そのために、誰がどのコースを教えるか、どの生徒がどのコースに登録したか、また既知の生徒がどのコースを取っているかを把握しておかなければいけない。それぞれのコースにつき教員が一人の場合、どの集合を勧めますか？
2. 教員側を各コース一人ではなく、いくつかのコースには複数の教員を使うことにより、この件を複雑にした場合はどうでしょうか。問1の回答をどのように変更するか？