STICHTING
MATHEMATISCH CENTRUM
2e BOERHAAVESTRAAT 49
AMSTERDAM

AFDELING MATHEMATISCHE STATISTIEK

S 371

AN INPUT SYSTEM FOR LINEAR PROGRAMMING PROBLEMS

part 1

formal description of L.P. problems

by

Jac.M. Anthonisse

preliminary 4
21-1-69

The Mathematical Centre at Amsterdam, founded the 11th of February, 1946, is a non-profit institution aiming at the promotion of pure mathematics and its applications, and is sponsored by the Netherlands Government through the Netherlands Organization for Pure Research (Z.W.O.) and the Central National Council for Applied Scientific Research in the Netherlands (T.N.O.), by the Municipality of Amsterdam and by several industries.

Summary

The formal system presented in this report gives a syntactical definition of linear programming problems.
This system can be used to construct a computer-program which accepts the mathematical model of a linear programming problem as input.

Introduction

Any linear programming problem can be put into the following form:

$$\text{maximize} \quad \sum_{j=1}^{n} c_j x_j \qquad (1)$$

subject to the constraints

$$\sum_{j=1}^{n} a_{ij} x_j = b_i \qquad (i=1, \ldots, m) \qquad (2)$$

and

$$x_j \geq 0 \ (j=1, \ldots, n) \qquad (3).$$

In these formulas n, m, c_j, b_i and a_{ij} represent known numerical values. The problem is to find optimal numerical values for the variables x_j.

The formulation of an actual problem as a linear programming problem seldom leads directly to formulas that are as simple as (1), (2) and (3).

Some examples of more complex constructions will be given now, the symbols x, y and z denote variables.

$$\sum_{j=1}^{n} x_j + \sum_{j=1}^{m} y_j \qquad (4)$$

$$\sum_{i=1}^{m} \sum_{j=1}^{n} a_{ij} x_{ij} \qquad (5)$$

$$\sum_{i=1}^{m} \sum_{j=1}^{i} a_{ij} x_{ij} \qquad (6)$$

$$\sum_{i=1}^{m} \sum_{j=l_i}^{n_i} (a_{ij} x_{ij} + z_j + y_i) \qquad (7)$$

$$\sum_{i=1}^{m} \sum_{j=i}^{8} (a_{i-j,j} \, x_{i,j+5} + z_j + y_i) \qquad (8)$$

$$\sum_{i=1}^{m} x_{k_i+i} \qquad (9)$$

$$\sum_{i=1}^{j} \frac{a_{kj}}{b_l} x_i + y_l = c_l \quad (j=1, \ldots, m;\ k=j, \ldots, m;\ l=1, \ldots, u_{jk}) \tag{10}$$

If a model contains complex constructions it may be a tedious task to put the model into the standard form (1), (2), (3). This task must be performed because the techniques for solving linear programming problems require as "input" the columns of the standardized matrix of constraints.

Often, sequences of L.P. problems are solved which are all similar, in the sense that the matrices of constraints have a common "structure". In such cases a computer program can be written which generates columns of the matrix of constraints (all columns, the "next" column, or a specified column). The preparation of input for the program that solves the L.P. problem is usually simplified by such a "matrix generator". Another advantage is the decrease of storage requirements.

For the solution of a sequence of L.P. problems without a common "structure" a "general matrix generator" might be usefull. The input for such a generator should contain a description of the "structure" of the matrix. Without defining the meaning of the term "structure" it may be stated that the mathematical model of a linear programming problem contains a complete description of the matrix of constraints. Thus the following input should be sufficient for a "general matrix generator":

α) a mathematical model of the L.P. problem,

β) numerical values for the coefficients occurring in the model.

The present report is part of a project to construct a computer program that approximates a "general matrix generator". In sections 1-6 a formal system is presented, giving a syntactical definition of the concept "linear programming problem", including a method to describe the numerical values for coefficients in the model.

In section 7 some examples are given.

1. Basic Concepts, Syntax

(1.1) <letter>:=
a|b|c|d|e|f|g|h|i|j|k|l|m|n|o|p|q|r|s|t|u|v|w|x|y|z

(1.2) <digit>:= 0|1|2|3|4|5|6|7|8|9

(1.3) <identifier>:= <letter>|<identifier><letter>|<identifier><digit>

(1.4) <natural>:= <digit>|<natural><digit>

(1.5) <decimal fraction>:=.<natural>

(1.6) <exponent part>:= $_{10}$<natural>|$_{10}$+<natural>|$_{10}$-<natural>

(1.7) <decimal number>:= <natural>|<decimal fraction>|<natural><decimal fraction>

(1.8) <unsigned number>:= <decimal number>|<exponent part>|
<decimal number><exponent part>

(1.9) <number>:= +<unsigned number>|-<unsigned number>

(1.10) <symbol>:= <letter>|<digit>|<other symbol>

(1.11) <sequence>:= <symbol>|<sequence><symbol>

(1.12) <string>:= ‹<sequence>›

(1.13) <identification>:= <string>

Basic Concepts, Semantics

The notation that is used to define the formal system is best explained by an example. In definition (1.4) a "natural" is defined to be either a "digit" or a "natural" followed by a "digit".
With the help of definition (1.2) the following examples of "natural" are found.

0
01
35

Some examples of other concepts:

(1.3)	i	variable	x1
(1.8)	5.6	$_{10}2$	1.1$_{10}{-3}$
(1.12)	{Example}		

The concept "other symbol", used in (1.10), is not defined in this report, except for the provision that "}" is not an "other symbol". An "other symbol" can occur in a "string" only, and may be a capital letter or any symbol that can be reproduced in an implementation of the formal system.
A "string" does not influence the computations but is used to identify the problem and the constraints, and may describe their physical meaning. These "strings" can be reproduced in the solution of an actual problem.

2. Formulas and Linear Forms, Syntax

(2.1) <lower bound>:= <real formula>

(2.2) <upper bound>:= <real formula>

(2.3) <sum>:= SUM(<index>,<lower bound>,<upper bound>,<real formula>)

(2.4) <real primary>:= <unsigned number>|<index>|<integer>|<real>|
<formula> | <sum>|(<real formula>)

(2.5) <real factor>:= <real primary>|<real factor>↑<real primary>

(2.6) <real term>:= <real factor>|<real term>×<real factor>|
<real term>/<real factor>

(2.7) <real formula>:= <real term>|+<real term>|-<real term>|
<real formula>+<real term>|<real formula>-<real term>

(2.8) <simple term>:= <variable>|<real term>×<variable>

(2.9) <simple linear form>:= <simple term>|+<simple term>|-<simple term>|
<simple linear form>+<simple term>|<simple linear form>-<simple term>

(2.10) <final sigma>:= S(<index>,<lower bound>,<upper bound>,<simple linear form>)

(2.11) <sigma>:= <final sigma>|S(<index>,<lower bound>,<upper bound>,<sigma>)

(2.12) <term>:= <simple term>|<sigma>| <real term>×<sigma>

(2.13) <linear form>:= <term>|+<term>|-<term>|<linear form>+<term>|
<linear form>-<term>

Formulas and Linear Forms, Semantics

A "real formula" is a rule to compute a numerical value. That numerical value is found by the application of operators to operands. An operand may be given in the form of a "real formula".
Apart from the operators addition (+), subtraction (-), multiplication (×), division (/) and exponentiation (↑) the system contains the operator SUM. The value of a "sum" (2.3) is

$$\sum(f \mid l \leq i \leq u)$$

in the usual mathematical notation, where f,l,i and u denote a "real formula", "lower bound","index" and "upper bound" respectively.

The concept of "linear form" is essential for the definition of a linear programming problem.
It occurs as objective function and as left hand side in a constraint.
A homogeneous linear form which, in the usual mathematical notation, can be written neither using the ∑-symbol nor the "and so on" dots, is equivalent with a "simple linear form" (2.9).
In the formal system, the ∑-symbol and the dots are replaced by a "sigma" (2.11). The meaning of a "sigma" is

$$\sum(f \mid l \leq i \leq u)$$

in the usual mathematical notation, where l, i and u denote a "lower bound", "index", "upper bound" respectively and f denotes either a "simple linear form" or a "sigma".
The facts that f does not denote a "linear form" and that each "linear form" is homogeneous put some restrictions on the notation of a model.

In both a "sum" and a "sigma" nesting may occur, the "index" of a "sum" may not be used, explicitly or implicitly, outside the "real formula" of that sum.
Similarly, the "index" of a "sigma" may not be used, explicitly or implicitly, outside the lower-level-sigma.

3. Declarations, Syntax

(3.1) <simple domain>:= <lower bound>≤<index>≤<upper bound>

(3.2) <list of simple domains>:= <simple domain>|
<list of simple domains>,<simple domain>

(3.3) <domain>:= (<list of simple domains>)

(3.4) <list of indices>:= <index>|<list of indices>,<index>

(3.5) <simple declarant>:= <identifier>

(3.6) <indexed declarant>:= <identifier>[<list of indices>]<domain>

(3.7) <declarant>:= <simple declarant>|<indexed declarant>

(3.8) <list of declarants>:= <declarant>|<list of declarants>,<declarant>

(3.9) <formula description>:= <simple declarant> = <real formula>

(3.10) <list of formula descriptions>:=
<formula description>|<list of formula descriptions>,<formula description>

(3.11) <declaration of indices>:= < >|index <list of indices>;

(3.12) <declaration of integers>:= < >|integer <list of declarants>;

(3.13) <declaration of reals>:= < >|real <list of declarants>;

(3.14) <declaration of formulas> := < >|formula <list of formula descriptions>;

(3.15) <declaration of discrete variables>:= < >|discrete <list of declarants>;

(3.16) <declaration of continuous variables>:= < >|continuous <list of declarants>;

(3.17) <declaration part>:= <declaration of indices><declaration of integers>
<declaration of reals> <declaration of formulas>
<declaration of discrete variables><declaration of continuous variables>

(3.18) <subscript>:= <real formula>

(3.19) <list of subscripts>:= <subscript>|<list of subscripts>,<subscript>

(3.20) <subscripted>:= <identifier>[<list of subscripts>]

(3.21) <simple>:= <identifier>

(3.22) <index>:= <simple>

(3.23) <integer>:= <simple>|<subscripted>

(3.24) <real>:= <simple>|<subscripted>

(3.25) <discrete>:= <simple>|<subscripted>

(3.26) <continuous>:= <simple>|<subscripted>

(3.27) <formula>:= <simple>

(3.28) <variable>:= <discrete>|<continuous>

Declarations, Semantics

The meaning of all identifiers used in the description of the linear programming problem must be specified in the "declaration part" (3.17). Each identifier has a unique type, determined by the declaration of that identifier.
By the "declaration of indices" each identifier occurring as an "index" in the "list of indices" is defined to be of type "index".
By the "declaration of integers" (3.12), "declaration of reals" (3.13), "declaration of discrete variables" (3.15) and the "declaration of continuous variables" (3.16) each identifier occurring either as a "simple declarant" (3.5) or as an "indexed declarant" (3.6) is defined to be of type "integer", "real", "discrete" or "continuous" respectively.
By the "declaration of formulas" (3.14), each identifier occurring as a "simple declarant" in a "formula description" (3.9) is defined to be of type "formula".

In the "domain" of an "indexed declarant" (3.6) the "lower bound" and "upper bound" of each index of that "indexed declarant" is defined. In the domain the indices must be given in the same order as in the list of indices.
If a "lower bound" or an "upper bound" depends on an index that index should precede these bounds in the domain.

Identifiers of the types "integer" and "real" correspond with known numerical values in the mathematical model, belonging to the set of integer and real numbers respectively.
Identifiers of the types "discrete" and "continuous" correspond with variables in the mathematical model. A "discrete" variable must have an integer value in the solution of the linear programming problem.
Identifiers of the type "formula" identify a "real formula" and represent numerical values. The numerical value of a "formula" is found by evaluating the corresponding "real formula".

A "simple" (3.21) and a "subscripted" (3.20) correspond with a "simple declarant" and an "indexed declarant" respectively. The number of subscripts must be equal to the number of indices.

4. Objective Function and Constraints, Syntax

(4.1) <objective function>:= <linear form>

(4.2) <objective part>:= MAXIMIZE : <objective function>|
MINIMIZE : <objective function>

(4.3) <relation>:= ≤ | ≥ | =

(4.4) <constraint> := <linear form><relation><real formula>|
<linear form><relation><real formula><domain>

(4.5) <constraint description>:= <identification><constraint>

(4.6) <list of constraint descriptions>:= <constraint description>|
<list of constraint descriptions><constraint description>

(4.7) <constraints part>:= <list of constraint descriptions>

(4.8) <proper model>:= <objective part><constraints part>

(4.9) <structural part>:= <identification><declaration part><proper model>

Objective Function and Constraints, Semantics

A linear programming problem consists of an objective function and a number of constraints. Each index used in the objective function must have a "lower bound" and an "upper bound" defined within the objective function.
In a "constraint" the "lower bound" and "upper bound" of an index are either defined within the "linear form" or "real formula" or in the "domain". In the latter case each value of the index corresponds with a set of rows in the standardizedmatrix of constraints.
If, in a domain, a "lower bound" or an "upper bound" depends on an index that index should precede these bounds in the domain.

5. Initialization and Modification, Syntax

(5.1) <list of numbers>:= <number>|<list of numbers><number>

(5.2) <simple piece determination>:= <natural>|+<natural>|-<natural>|<index>

(5.3) <piece determination>:= <simple piece determination>|
<piece determination>,<simple piece determination>

(5.4) <order>:= < >|(<list of indices>)

(5.5) <piece>:= <identifier>|<identifier>[<piece determination>]<order>

(5.6) <portion>:= <piece><list of numbers>

(5.7) <list of portions>:= <portion>|<list of portions><portion>

(5.8) <initialization>:= INIT <identification><list of portions>

(5.9) <post optimization>:= POST <identification><list of portions>

(5.10) <range>:= RANGE <number><number>

(5.11) <parametrization>:= PARA <identification><range><list of portions>

(5.12) <modification>:= <initialization>|<post optimization>|<parametrization>

(5.13) <list of modifications>:= < >|<modification>|
<list of modifications><modification>

(5.14) <numerical part>:= <initialization><list of modifications>|< >

(5.15) <linear programming problem>:= OPEN <structural part>
<numerical part> CLOSE

Initialization and Modification, Semantics

In an "initialization" (5.8) and a "post optimization" (5.9) numerical values are assigned to "integers" and "reals", and an actual linear programming problem, determined by these values, is to be solved.
The values are given in "portions" (5.6), the "piece determination" (5.3) and the "order" (5.4) specify the correspondence between a "subscripted" and the "list of numbers". This is best explained by an example and some Algol-60.
Let a be declared as

$$\underline{\text{integer}}\quad a[i,j,k]\ (1 \le i \le 3,\ 1 \le j \le 2,\ 1 \le k \le j);$$

The piece a[i,j,k]
defines the correspondence:

```
for i: = 1,2,3    do
for j: = 1,2      do
for k: = 1 step 1 until j do
    a[i,j,k]: = next number;
```

The piece a[i,j,k] (j,i,k)
defines the correspondence:

```
for j: = 1,2      do
for i: = 1,2,3    do
for k: = 1 step 1 until j do
    a[i,j,k]: = next number;
```

The piece a[i,2,k] (k,i)
defines the correspondence:

```
for k: = 1,2      do
for i: = 1,2,3    do
    a[i,2,k]: = next number;
```

The terms "post optimization" and "initialization" indicate that the computations to solve the new problem should or should not be based upon the solution of the previous problem.
In a "parametrization" (5.11) the "list of portions" is preceded by a "range" (5.10) defining an interval of real numbers. Each real number in this interval corresponds with a linear programming problem.

Let q be declared as

$$\underline{\text{real}}\ q;$$

and let q have, by an initialization or a modification, the value 3.
Then

RANGE + 5 + 8 q + 4

means
that for each value of p $(+5 \leq p \leq +8)$ the linear programming problem that is found by substituting for q the expression

$$3 + 4 \times p$$

is to be solved.
Parametrization is similarly defined for subscripted reals and integers.

6. Alphabetic list of concepts

constraint	(4.4)
constraint description	(4.5)
constraints part	(4.7)
continuous	(3.26)
decimal fraction	(1.5)
decimal number	(1.7)
declarant	(3.7)
declaration of continuous variables	(3.16)
declaration of discrete variables	(3.15)
declaration of formulas	(3.14)
declaration of indices	(3.11)
declaration of integers	(3.12)
declaration of reals	(3.13)
declaration part	(3.17)
digit	(1.2)
discrete	(3.25)
domain	(3.3)
exponent part	(1.6)
final sigma	(2.10)
formula	(3.27)
formula description	(3.9)
identification	(1.13)
identifier	(1.3)
index	(3.22)
indexed declarant	(3.6)
initialization	(5.8)
integer	(3.23)
letter	(1.1)
linear form	(2.13)
linear programming problem	(5.15)
list of constraint descriptions	(4.6)
list of declarants	(3.8)
list of formula descriptions	(3.10)
list of indices	(3.4)
list of modifications	(5.13)
list of numbers	(5.1)
list of portions	(5.7)
list of simple domains	(3.2)

list of subscripts	(3.19)
lower bound	(2.1)
modification	(5.12)
natural	(1.4)
number	(1.9)
numerical part	(5.14)
objective function	(4.1)
objective part	(4.2)
order	(5.4)
other symbol	(1.10)
parametrization	(5.11)
piece	(5.5)
piece determination	(5.3)
portion	(5.6)
post optimization	(5.9)
proper model	(4.8)
range	(5.10)
real	(3.24)
real factor	(2.5)
real formula	(2.7)
real primary	(2.4)
real term	(2.6)
relation	(4.3)
sequence	(1.11)
sigma	(2.11)
simple	(3.21)
simple declarant	(3.5)
simple domain	(3.1)
simple linear form	(2.9)
simple piece determination	(5.2)
simple term	(2.8)
string	(1.12)
structural part	(4.9)
subscript	(3.18)
subscripted	(3.20)
sum	(2.3)
symbol	(1.10)
term	(2.12)
unsigned number	(1.8)
upper bound	(2.2)
variable	(3.28)

7. Examples

This section contains some linear programming problems, formulated in accordance with the formal system developed above.

Only the "structural part" of each problem is given.
Although an implementation of the formal system might contain the convention "all variables have lower bound = 0 unless specified otherwise", all lower bounds are given explicitly.

```
OPEN ⊰ Example 1, Curve fitting with minimum deviations,
see: Walter D. Fisher,
     A note on curve fitting with minimum deviations by linear programming,
     Journal of the American Statistical Association 56 (1961) 359-362⊱

index        i,j;
integer      n,k;
real         x[i,j](1 ≤ i ≤ n, 1 ≤ j ≤ k);
continuous   u[i](1 ≤ i ≤ n), v[i](1 ≤ i ≤ n),
             y[i](1 ≤ i ≤ k), z[i](1 ≤ i ≤ k);

MINIMIZE:  S(i,1,n,u[i] + v[i])

⊰equations⊱

     u[i] - v[i] + y[1] - z[1] +

     S(j,2,k,x[i,j] × y[j] - x[i,j] × z[j]) = x[i,1](1 ≤ i ≤ n)

⊰non-neg u⊱        u[i] ≥ 0        (1 ≤ i ≤ n)
⊰non-neg v⊱        v[i] ≥ 0        (1 ≤ i ≤ n)
⊰non-neg y⊱        y[j] ≥ 0        (1 ≤ j ≤ k)
⊰non-neg z⊱        z[j] ≥ 0        (1 ≤ j ≤ k)
```

```
OPEN < Example 2, Optimal production,
see: W. Dinkelbach und F. Steffens,
     Gemischt ganzzahlige lineare Programme zur Loesung gewisser
     Entscheidungsprobleme,
     Unternehmensforschung 5 (1961) 3-14,
     Model III>

index       i,j,r;
integer     n,nsub[i](1 ≤ i ≤ n),k,m;
real        g[i,r](1 ≤ i ≤ n, 1 ≤ r ≤ nsub[i]),
            c[i,r](1 ≤ i ≤ n, 1 ≤ r ≤ nsub[i]),
            a[j,i,r](1 ≤ j ≤ m, 1 ≤ i ≤ n, 1 ≤ r ≤ nsub[i]),
            b[j](1 ≤ j ≤ m);
discrete    u[i,r](1 ≤ i ≤ n, 1 ≤ r ≤ nsub[i]);
continuous  x[i,r](1 ≤ i ≤ n, 1 ≤ r ≤ nsub[i]);

MAXIMIZE:   S(i,1,n,S(r,1,nsub[i], c[i,r] × x[i,r]))

<process>
   S(i,1,n,S(r,1,nsub[i], a[j,i,r] × x[i,r])) ≤ b[j](1 ≤ j ≤ m)

<bounds>
   x[i,r] - g[i,r] × u[i,r] ≤ 0 (1 ≤ i ≤ n, 1 ≤ r ≤ nsub[i])

<at most one partial process>
   S(r,1,nsub[i], u[i,r]) ≤ 1 (1 ≤ i ≤ n)

<at most k processes>
   S(i,1,n,S(r,1,nsub[i], u[i,r])) ≤ k

<non-neg u>
   u[i,r] ≥ 0 (1 ≤ i ≤ n, 1 ≤ r ≤ nsub[i])

<non-neg x>
   x[i,r] ≥ 0 (1 ≤ i ≤ n, 1 ≤ r ≤ nsub[i])
```

```
OPEN ⊀ Example 3, Optimal production,
see: example 2≻

index       i,j,r;
integer     n,m,k,nsub[i](1 ≤ i ≤ n);
real        a[j,i,r](1 ≤ j ≤ m, 1 ≤ i ≤ n, 1 ≤ r ≤ nsub[i]),
            b[j](1 ≤ j ≤ m),
            c[i,r](1 ≤ i ≤ n, 1 ≤ r ≤ nsub[i]),
            beta[i](1 ≤ i ≤ n);
formula     g = r × beta[i]/nsub[i];
discrete    u[i,r](1 ≤ i ≤ n, 1 ≤ r ≤ nsub[i]);
continuous  x[i,r](1 ≤ i ≤ n, 1 ≤ r ≤ nsub[i]);


MAXIMIZE:   S(i,1,n,S(r,1,nsub[i], c[i,r] × x[i,r]))

⊀process≻
   S(i,1,n,S(r,1,nsub[i], a[j,i,r] × x[i,r])) ≤ b[j](1 ≤ j ≤ m)

⊀bounds≻
   x[i,r] - g × u[i,r] ≤ 0 (1 ≤ i ≤ n, 1 ≤ r ≤ nsub[i])

⊀at most one partial process≻
   S(r,1,nsub[i], u[i,r]) ≤ 1 (1 ≤ i ≤ n)

⊀at most k processes≻
   S(i,1,n,S(r,1,nsub[i], u[i,r])) ≤ k

⊀non-neg u≻
   u[i,r] ≥ 0 (1 ≤ i ≤ n, 1 ≤ r ≤ nsub[i])

⊀non-neg x≻
   x[i,r] ≥ 0 (1 ≤ i ≤ n, 1 ≤ r ≤ nsub[i])
```

```
OPEN ⊰ Example 4, Capacity planning,
see: Robert B. Fetter,
     A linear programming model for long range capacity planning,
     Management Science 7 (1961) 372-378
     log[t] = natural logarithm of 1 + ρ_t ⊱

index       i,j,t;
integer     n,r,m,
            capt[j](1 ≤ j ≤ r),
            k[j](r+1 ≤ j ≤ m-1);
real        a[i,j](2 ≤ i ≤ n, 1 ≤ j ≤ m-1),
            b[i,j](2 ≤ i ≤ n + 1, 1 ≤ j ≤ r),
            c[i,j](1 ≤ i ≤ n, 1 ≤ j ≤ m),
            s[i,j](1 ≤ i ≤ n, 1 ≤ j ≤ m-1),
            log[t](1 ≤ t ≤ n),
            capd[i](1 ≤ i ≤ n);
formula     dis = 2.71828 ↑ (- SUM(t,1,i,log[t]));
continuous  cap x[i,j](1 ≤ i ≤ n, 1 ≤ j ≤ m),
            cap a[i,j](1 ≤ i ≤ n, 1 ≤ j ≤ m-1),
            cap b[i,j](2 ≤ i ≤ n+1, 1 ≤ j ≤ r),
            cap s[i,j](1 ≤ i ≤ n, 1 ≤ j ≤ m-1);

MINIMIZE:
    S(i,1,n,S(j,1,m,dis × c[i,j] × cap x[i,j]))
  + S(i,1,n,S(j,1,m-1, dis × a[i,j] × cap a[i,j]))
  + S(i,1,n,S(j,1,m-1, dis × s[i,j] × cap s[i,j]))
  - S(i,2,n+1,S(j,1,r, dis × b[i,j] × cap b[i,j]))

⊰demands⊱

    S(j,1,m,cap x[i,j]) = cap d[i](1 ≤ i ≤ n)

⊰owned 1⊱

    capx[1, j] - capa[1, j] + caps[1, j]= 0 (1 ≤ j ≤ r)

⊰owned 2⊱

    cap x[i,j] - S(t,1,i,cap a[t,j] - cap b[t,j]) + cap s[i,j] = 0
                         (1 ≤ j ≤ r, 2 ≤ i ≤ n)
```

```
<leased>
   cap x[i,j] - S(t,i - k[j] + 1,i,cap a[t,j]) + cap s[i,j] = 0
                 (r+1 ≤ j ≤ m-1, 1 ≤ i ≤ n)

<terms of service>
   S(i,2,cap t[j] + t, cap b[i,j])
 - S(i,1,t,cap a[i,j]) ≥ 0

                (1 ≤ j ≤ r, 1 ≤ t ≤ n - cap t[j] + 1)

<non neg cap x>
     cap x[i,j] ≥ 0 (1 ≤ i ≤ n, 1 ≤ j ≤ m)

<non neg cap a>
     cap a[i,j] ≥ 0 (1 ≤ i ≤ n, 1 ≤ j ≤ m-1)

<non neg cap b>
     cap b[i,j] ≥ 0 (2 ≤ i ≤ n+1, 1 ≤ j ≤ r)

<non neg cap s>
     cap s[i,j] ≥ 0 (1 ≤ i ≤ n, 1 ≤ j ≤ m-1)
```

```
OPEN ⊀ Example 5, Traffic signals,
see: John D.C. Little,
     The synchronization of traffic signals by mixed-integer linear
     programming,
     Operations Research 14 (1966) 568-594,
     LP2⊁

index       i;
real        k,cap t1,cap t2,n,
            d[i](1 ≤ i ≤ n-1),
            r[i](1 ≤ i ≤ n),
            f[i](1 ≤ i ≤ n-1),
            e[i](1 ≤ i ≤ n-1),
            f bar[i](1 ≤ i ≤ n-1),
            e bar[i](1 ≤ i ≤ n-1),
            h[i](1 ≤ i ≤ n-2),
            g[i](1 ≤ i ≤ n-2),
            h bar[i](1 ≤ i ≤ n-2),
            g bar[i](1 ≤ i ≤ n-2);
discrete    m[i](1 ≤ i ≤ n-1);
continuous  b,bbar,z,
            w[i](1 ≤ i ≤ n),
            w bar[i](1 ≤ i ≤ n),
            t[i](1 ≤ i ≤ n-1),
            t bar[i](1 ≤ i ≤ n-1);

MAXIMIZE:   b + bbar

⊀11⊁        bbar - k × b = 0

⊀12a⊁       z ≥ 1/cap t2

⊀12b⊁       z ≤ 1/cap t1

⊀13a⊁       w[i] + b ≤ 1-r[i] (1 ≤ i ≤ n)

⊀13b⊁       w bar[i] + bbar ≤ 1-r[i](1 ≤ i ≤ n)

⊀14⊁        w[i] + w bar[i] - w[i+1] - w bar[i+1]
          + t[i] + t bar[i] - m[i] = -r[i] + r[i+1](1 ≤ i ≤ n-1)
```

<15aa> d[i]/f[i] × z -t[i] ≤ 0 (1 ≤ i ≤ n-1)

<15ab> t[i] - d[i]/e[i] × z ≤ 0 (1 ≤ i ≤ n-1)

<15ba> d[i]/f bar[i] × z - t bar[i] ≤ 0 (1 ≤ i ≤ n-1)

<15bb> t bar[i] - d[i]/e bar[i] × z ≤ 0 (1 ≤ i ≤ n-1)

<16aa> d[i]/h[i] × z -d[i]/d[i+1] × t[i+1]
+ t[i] ≤ 0 (1 ≤ i ≤ n-2)

<16ab> d[i]/d[i+1] × t[i + 1] - t[i]
- d[i]/g[i] × z ≤ 0 (1 ≤ i ≤ n-2)

<16ba> d[i]/h bar[i] × z - d[i]/d[i+1] × t bar[i+1]
+ t bar[i] ≤ 0 (1 ≤ i ≤ n-2)

<16bb> d[i]/d[i+1] × t bar[i+1] - t bar[i]
-d[i]/g bar[i] × z ≤ 0 (1 ≤ i ≤ n-2)

<non neg> b ≥ 0
<non neg> b bar ≥ 0
<non neg> w[i] ≥ 0 (1 ≤ i ≤ n)
<non neg> w bar[i] ≥ 0 (1 ≤ i ≤ n)

```
OPEN ⊀ Example 6, Fermentation,
see: E. Koenigsberg,
     Some Industrial Applications of
     Linear Programming,
     Operations Research Quarterly 12 (1961) 105-114⊁

index       i,j,n;
integer     capn,capm,tm;
real        cm,cf,cp,cb,cw,mu,phi,pi,beta,chi,
            hm[i](1 ≤ i ≤ capn),   hpm[i](1 ≤ i ≤ capn),
            hf[i](1 ≤ i ≤ capn),   hpf[i](1 ≤ i ≤ capn),
            hp[i](1 ≤ i ≤ capn),   hpp[i](1 ≤ i ≤ capn),
            hb[i](1 ≤ i ≤ capn),   hpb[i](1 ≤ i ≤ capn),
            hw[i](1 ≤ i ≤ capn),   hpw[i](1 ≤ i ≤ capn),
            plantcap,warecap,capf0,capp0,capw0,d[i](1 ≤ i ≤ capn);
continuous  m[i](1 ≤ i ≤ capn),    mp[i](1 ≤ i ≤ capn),
            f[i](1 ≤ i ≤ capn),    fp[i](1 ≤ i ≤ capn),
            p[i](1 ≤ i ≤ capn),    pp[i](1 ≤ i ≤ capn),
            b[i](1 ≤ i ≤ capn),    bp[i](1 ≤ i ≤ capn),
            w[i](1 ≤ i ≤ capn),    wp[i](1 ≤ i ≤ capn),
            capf[j](1 ≤ j ≤ capn), capp[j] (1 ≤ j ≤ capn),
            capw[j](1 ≤ j ≤ capn);

MINIMIZE:   S(i,1,capn,mu × cm × mp[i] +
                      phi × cf × fp[i] +
                       pi × cp × pp[i] +
                     beta × cb × bp[i] +
                      chi × cw × wp[i] )

⊀3⊁         f[i] + fp[i] - m[i-tm] - mp[i-tm] = 0
                         (tm+1 ≤ i ≤ capn)

⊀4⊁         -capf[j] + S(i,1,j,
                 f[i] + fp[i] - p[i] - pp[i]) =  -capf0    (1 ≤ j ≤ capn)
```

⟨5⟩	S(n,0,9,f[j-n] + fp[j-n]) - capf[j] ≤ 0	(10 ≤ j ≤ capn)
⟨5b⟩	capf[j] - S(n,0,29,f[j-n] + fp[j-n]) ≤ 0	(30 ≤ j ≤ capn)
⟨6⟩	-capp[j] + S(i,1,j, p[i] + pp[i] - b[i] - bp[i]) = -capp0	(1 ≤ j ≤ capn)
⟨7a⟩	S(n,0,1,p[j-n] + pp[j-n]) - capp[j] ≤ 0	(2 ≤ j ≤ capn)
⟨7b⟩	capp[j] - S(n,0,4,p[j-n] + pp[j-n]) ≤ 0	(5 ≤ j ≤ capn)
⟨8a⟩	b[i] ≤ plantcap	(1 ≤ i ≤ capn)
⟨8b⟩	bp[i] ≤ plantcap	(1 ≤ i ≤ capn)
⟨9a⟩	-capw[j] + S(i,1,j,w[i] + wp[i])= SUM(i,1,j,d[i]) - capw0	(1 ≤ j ≤ capn)
⟨9b⟩	capw[j] ≤ warecap	(1 ≤ j ≤ capn)
⟨10⟩	capw[j] S(n,0,capm,w[j-n] + wp[j-n]) ≤ 0	(capm + 1 ≤ j ≤ capn)
⟨11a⟩	mu × m[i] ≤ hm[i]	(1 ≤ i ≤ capn)
⟨11b⟩	mu × mp[i] ≤ hpm[i]	(1 ≤ i ≤ capn)
⟨11c⟩	phi × f[i] ≤ hf[i]	(1 ≤ i ≤ capn)
⟨11d⟩	phi × fp[i] ≤ hpf[i]	(1 ≤ i ≤ capn)
⟨11e⟩	pi × p[i] ≤ hp[i]	(1 ≤ i ≤ capn)
⟨11f⟩	pi × pp[i] ≤ hpp[i]	(1 ≤ i ≤ capn)
⟨11g⟩	beta × b[i] ≤ hb[i]	(1 ≤ i ≤ capn)
⟨11h⟩	beta × bp[i] ≤ hpb[i]	(1 ≤ i ≤ capn)
⟨11i⟩	chi × w[i] ≤ hw[i]	(1 ≤ i ≤ capn)

<11j> $chi \times wp[i] \leq hpw[i] \quad (1 \leq i \leq capn)$

<12> $m[i] \geq 0 \ (1 \leq i \leq capn)$

<13> $f[i] \geq 0 \ (1 \leq i \leq capn)$

<14> $p[i] \geq 0 \ (1 \leq i \leq capn)$

<15> $b[i] \geq 0 \ (1 \leq i \leq capn)$

<16> $w[i] \geq 0 \ (1 \leq i \leq capn)$

<17> $mp[i] \geq 0 \ (1 \leq i \leq capn)$

<18> $fp[i] \geq 0 \ (1 \leq i \leq capn)$

<19> $pp[i] \geq 0 \ (1 \leq i \leq capn)$

<20> $bp[i] \geq 0 \ (1 \leq i \leq capn)$

<21> $wp[i] \geq 0 \ (1 \leq i \leq capn)$

<22> $cap\ f[i] \geq 0 \ (1 \leq i \leq capn)$

<23> $cap\ p[i] \geq 0 \ (1 \leq i \leq capn)$

<24> $cap\ w[i] \geq 0 \ (1 \leq i \leq capn)$

```
OPEN ⊰ Example 7, Chromatic number,
The graph G contains n vertices and m edges.
Edge j connects vertex s[j] with vertex t[j]
z = chromatic number of G⊱

index       i,j;
integer     s[j](1 ≤ j ≤ m), t[j](1 ≤ j ≤ m),m,n;
discrete    x[j](1 ≤ j ≤ n), d[i](1 ≤ i ≤ m),z;

MINIMIZE:   z

⊰z bounds⊱ z - x[j] ≥ 0                        (1 ≤ j ≤ n)

⊰conflicts 1 ⊱

     x[s[i]] - x[t[i]] + n × d[i] ≥ 1      (1 ≤ i ≤ m)

⊰conflicts 2⊱

     x[s[i]] - x[t[i]] + n × d[i] ≤ n-1    (1 ≤ i ≤ m)

⊰zero-one d⊱
           d[i] ≤ 1                         (1 ≤ i ≤ m)

⊰non-neg d⊱
           d[i] ≥ 0                         (1 ≤ i ≤ m)

⊰natural x⊱
           x[j] ≥ 1                         (1 ≤ j ≤ n)
```