Leichterer Zugang zu Operations Research : die Modelliersprache LPL

Autor(en): **Hürlimann, Tony**

Objekttyp: **Article**

Zeitschrift: **Bulletin des Schweizerischen Elektrotechnischen Vereins, des Verbandes Schweizerischer Elektrizitätsunternehmen = Bulletin de l'Association Suisse des Electriciens, de l'Association des Entreprises électriques suisses**

Band (Jahr): **83 (1992)**

Heft 17

PDF erstellt am: **11.09.2024**

Persistenter Link: https://doi.org/10.5169/seals-902861

Leichterer Zugang zu Operations Research

Die Modelliersprache LPL

Tony Hürlimann

Operations Research, ein etabliertes mathematisches Instrumentarium zur Lösung von Optimierungsaufgaben, leistet auf vielen Gebieten hervorragende Dienste. Es könnte allerdings noch wesentlich breiteren Einsatz finden, wenn nicht mathematische Hürden vielen Praktikern den Zugang zu den Methoden des Operations Research versperren würden. Der vorliegende Artikel beschreibt eine an der Universität Freiburg entwickelte Modelliersprache, mit welcher der Zugang erheblich vereinfacht wird.

Operations Research, un instrument mathématique établi pour résoudre des tâches d'optimisation, opère de remarquables performances dans beaucoup de domaines. Il trouverait cependant une utilisation notablement plus vaste si des obstacles mathématiques ne bloquaient pas à beaucoup de practiciens l'accès aux méthodes d'Operations Research. L'article décrit un langage de modélisation développé à l'Université de Fribourg, qui simplifie grandement l'accès.

Adresse des Autors

Tony Hürlimann, Dr.lic.rer.pol.,
Institut for Automation and Operations Research,
Université de Fribourg, 1700 Freiburg.

In der Praxis muss man in der Lage sein, komplexe Entscheidungen rasch zu treffen. Quantitative Aspekte sind dabei oft ebenso wichtig wie die qualitativen. Das Operations Research hat Methoden entwickelt, welche Entscheidungsprozesse bewältigen helfen, und dank Computern können heute grosse mathematische Modelle in einigen Minuten gerechnet werden. Die Operations Research-Forschung hat sich in den letzten Jahrzehnten hauptsächlich darauf konzentriert, mathematische Lösungsmethoden und -verfahren für bestimmte Modelle zu entwickeln. Diese Arbeiten erlauben – zusammen mit immer leistungsfähigeren Computern –, auch grössere Modelle zu lösen. Doch wurden Fragen des Modellmanagements (Modellformulierung, Modellbau und Modellwartung) langezeit stiefmütterlich behandelt und sind immer noch relativ unterentwickelt. Vor allem in der praktischen Anwendung des OR, wo der Einsatz grösserer Modelle für komplexe Entscheidungsprozesse unumgänglich ist, sind Modellmanagement-Instrumente nicht nur nützlich, sondern unerlässlich. Wenn die Lösung eines Modells auf einem Grosscomputer höchstens Minuten dauert, die Modellformulierung und -wartung sich aber über Wochen, ja Monate erstreckt, was durchaus nicht unüblich ist, so besteht hier ein krasses Missverhältnis, und der Einsatz von gut entwickelten Methoden lohnt sich nicht. Der Einsatz von OR-Methoden in der Praxis hängt direkt von der Investition ab, die es braucht, um solche Modelle zu erstellen und zu warten. Leider aber gibt es noch keine Instrumente, mit denen man grosse Modelle auch bequem manipulieren kann. Deswegen sind die finanziellen Mittel, die man benötigt, um grosse Modelle aufzustellen, zu warten und zu verändern, immer noch gewaltig. Die hohen Kosten sind einer der Hauptgründe, wieso die Methoden des Operations Research nur zögernd in die Praxis Eingang finden.

Die Einsicht, dass es für die praxisorientierte Anwendung des Operations Research mehr braucht als nur effiziente Lösungsalgorithmen, und dass ein Modell nicht von Anfang an in einer Form vorliegt, welche der Lösungsalgorithmus verlangt, hat sich in der letzten Zeit, wenn auch zögernd, durchgesetzt. Zwar wurden schon in den siebziger Jahren effiziente und mächtige Matrix-Generatoren und Report-Generatoren geschaffen [1...4], doch typischerweise arbeitet man mit diesen Instrumenten so, dass der Modellierer ein Computerprogramm schreibt, welches das Modell in eine vom Matrix-Generator lesbare Form bringt. Wird das Modell geändert, muss auch dieses Übersetzungsprogramm neu geschrieben oder abgeändert werden. Ein solches Vorgehen verlangt vom Modellierer nicht nur Kenntnisse in der Programmierung, sondern ist auch mit einem erheblichen Aufwand verbunden. Zudem liegt das Modell in einer Form vor – nämlich als Fortran-Programm –, das eher vom Programmierer als vom Modellierer interpretiert und manipuliert werden kann. Dieses Vorgehen ist auch heute noch verbreitet.

Eine Modelliersprache für Operations Research

Eine Modelliersprache als Instrument einer Modellformulierung könnte diese Nachteile umgehen, da sie eine Formulierung erlaubt, die dem Modellbauer näher steht. Die Idee einer Modelliersprache ist nicht neu, und verschiedene Sprachen wie Gams [5], AMPL [6], Lingo [7] sind bereits

Modelliersprache für Operations Research

Am Institut für Automation und Operations Research an der Universität Freiburg werden seit einiger Zeit Instrumente des Modellmanagements entwickelt. Eines davon ist die in diesem Beitrag vorgestellte Modelliersprache LPL (Linear Programming Language). Sie erlaubt auch grosse, mathematische Modelle in der bekannten, algebraischen Notation zu schreiben. Die LPL-Sprache wurde zum Erstellen von MPS Input-Dateien und Resultate-Tabellen grösserer LP-Modelle erfolgreich eingesetzt. Der LPL-Compiler übersetzt ein LPL-Programm, das ein vollständiges Modell repräsentiert, in den Eingabecode eines LP/MIP-Lösungsprogramms, ruft den Lösungsalgorithmus auf und liest die Lösung; ein integrierter Tabellengenerator gibt vom Benützer definierte Resultate-Tabellen aus. Ein Dateneingabe-Generator erlaubt zudem, die Daten in verschiedenen Formaten zu lesen.

entwickelt worden und verfügbar. Erwähnt werden soll auch der Ansatz SML [8], bei dem nicht nur eine Modelliersprache, sondern eine ganze Modellierumgebung im Zentrum steht. Eine Modelliersprache ist sicher nur ein, wenn auch wichtiger, Bestandteil einer Modellumgebung. Dazu gehören nämlich auch Instrumente der Modell-Datenverwaltung [9; 10], der Modelverifikation [11], der Modelldokumentation [12; 8] sowie Instrumente, die den gesamten Lebenszyklus und die Evolution eines Modells unterstützen.

Dieser Artikel stellt eine Modelliersprache, genannt LPL (Linear Programming Language), vor, mit der man auch grössere mathematische Modelle formulieren und bequem warten kann. Die Formulierung eines Modells in der LPL-Form kann direkt dazu verwendet werden, das Modell automatisch in den vom Lösungsalgorithmus verlangten Kode zu übersetzen. LPL unterscheidet sich von den oben genannten Spachen folgendermassen: Die Sprache besitzt eine offene Schnittstelle zu den meisten Lösungsalgorithmen. Der Dateneingabe-Generator ist in keiner andern Sprache vorhanden und nur Gams besitzt auch einen Resultate-Generator. Die Syntax von LPL ist flexibler. Entitäten können mehrmals definiert und zugewiesen werden wie in Gams, aber im Gegensatz zu Gams kann in LPL die Modelstruktur unabhängig von den Daten verarbeitet werden. LPL besitzt zudem als einzige Sprache die Möglichkeit, Masseinheiten selber zu definieren. Die wichtigsten Punkte seien kurz zusammengefasst:

- Ein LPL-Modell besteht aus einer einfachen Syntax – ähnlich der mathematischen Notation mit indexierten Ausdrücken.
- Es können kleine und grosse Modelle definiert werden, dank der Aufspaltung des Modells in Modellstruktur und Modelldaten.
- Ein mächtiger Indexmechanismus ist eingebaut, mit dem die Modellstruktur sehr flexibel gestaltet und das Modell «aufgeblasen» werden kann.
- Ein innovativer Tabellen- und Dateneingabe-Generator ist eingebaut.
- Beliebige indexierte Ausdrücke sind formulier- und sofort auswertbar.
- Die Variablen- und Restriktionennamen können automatisch oder vom Benützer gesteuert generiert werden.
- Mit Hilfe von verschiedenen Instrumenten kann das Modell auf Fehler untersucht werden.
- Ein eingebauter Texteditor erlaubt ein Modell zu formulieren und zu ändern.
- Für LP/MIP-Modelle wird der MPS-Standard-Kode schnell produziert.
- Die Schnittstelle zu einem LP/MIP-Lösungsalgorithmus ist offen und kann vom Benützer definiert werden.

Anhand eines einfachen Modells aus der Elektrizitätswirtschaft sollen im weiteren die Grundideen von LPL kurz vorgestellt werden.

Beispiel aus der Elektrizitätswirtschaft

Ein vereinfachtes Modellbeispiel aus der Elektrizitätswirtschaft soll Ausgangspunkt für die Beschreibung der Modeliersprache LPL sein [13]. In einem Elektrizitätswerk müssen die Stromgeneratoren ständig ein- und ausgeschaltet werden, um die stündlichen Schwankungen des Stromverbrauchs zu decken. Angenommen, ein Tag werde in 5 Zeitzonen eingeteilt, in welchen wir einen konstanten, geschätzten Stromverbrauch annehmen (Tab. I). Nehmen wir weiter vereinfachend an, dass im Elektrizitätswerk drei verschiedene Generatorentypen installiert sind, und zwar zwölf vom Typ 1, zehn vom Typ 2 und fünf vom Typ 3. Jeder Generatortyp, einmal in Betrieb, liefert eine minimale Strommenge, kann aber ein bestimmtes Maximum nicht übersteigen. Die Betriebskosten über dem Minimum sind höher als auf dem Minimum. Zudem müssen Anschaltkosten in Betracht gezogen werden. Die Daten sind in der Tabelle II zusammengestellt. Um die geschätzte Nachfrage jederzeit decken zu können, ist zudem vorgesehen, dass eine plötzliche Zunahme von 15% der Nachfrage abgefangen werden kann, ohne dass neue Generatoren in Betrieb genommen werden müssen.

Tageszeit-Zone	Stromverbrauch
Mitternacht bis 6 Uhr	15 000 Megawatt
6–9 Uhr	30 000 Megawatt
9–15 Uhr	25 000 Megawatt
15–18 Uhr	40 000 Megawatt
18 Uhr bis Mitternacht	27 000 Megawatt

Tabelle I

Die Frage ist nun, welche Generatoren müssen zu jeder Tageszeit in Betrieb sein oder in Betrieb genommen werden, wenn die Betriebskosten zu minimieren sind. Ausserdem interessiert uns die Frage, welches die marginalen Produktionskosten des Stroms zu jeder Tageszeit ist. Dieses Problem kann in ein lineares Optimierungsmodell umgewandelt werden, welches in Bild 1 aufgelistet ist. Da die Zeitzonen zyklisch definiert sind, d.h. die Vorperiode der ersten Zeitzone ist die letzte Zeitzone (des Vortages) und die nachfolgende Periode der letzten Zeitzone ist die erste Zeitzone (des nächsten Tages), ist die Restriktion $s_{it} \geq n_{it}-n_{it-1}$, auch für $t = 1$ definiert als $s_{i1} \geq n_{i1}-n_{iT}$. Die Struktur dieses Modells kann direkt in der LPL-Sprache wiedergegeben werden. Das entsprechende LPL-Programm ist in Bild 2 aufgelistet.

Das Modell in Bild 1 gibt die algebraische Formulierung des vereinfachten Modells wieder. Es besteht aus einer Liste von Deklarationen: Indexmengen, Datentabellen, Unbekannte, eine minimierende Funktion und lineare Restriktionen. Keine Daten sind definiert. Die Beschreibung ist dimensionsunabhängig und repräsentiert die Modellstruktur. Die LPL-Formulierung in Bild 2 ist ähnlich aufgebaut. Die verschiedenen Deklarationen werden jeweils durch ein reser-

	Typ 1	Typ 2	Typ 3
minimale Kapazität (GW)	850	1'250	1'500
maximale Kapazität (GW)	2'000	1'750	4'000
minimale Betriebskosten (SFr/Std)	1'000	2'600	3'000
zusätzliche Betriebskosten pro GW			
über dem Minimum (SFr / MW)	2.--	1.30	3.--
Anschaltkosten	2'000	1'000	500
Anzahl Generatoren	**12**	**10**	**5**

Tabelle II

viertes Wort SET, UNIT, COEF VAR, MODEL usw. eingeleitet und durch einen Strichpunkt abgeschlossen. Es ist wichtig zu sehen, dass diese Formulierung ebenfalls ganz unabhängig von den Daten ist. Die Daten sind definiert in den beiden Dateien Strom1.dat und Strom2.dat (s. Tab. III und IV für unser konkretes Beispiel). Die Grösse des Modells ist einzig abhängig von der Grösse dieser Datentabellen, die ausserhalb der Modellstruktur definiert sind. Die LPL-Formulierung zeigt noch einige weitere Differenzen gegenüber der algebraischen Formulierung von Bild 1: Kommentare stehen in Hochkommata oder in geschweiften Klammern, die Indexe sind geklammert statt tiefgestellt und Zeichen wie Σ sind durch reservierte Wörter wie SUM ersetzt worden. Zudem wird die LPL-Formulierung durch vier weitere Anweisungen ergänzt:

- UNIT wird verwendet, um Masseinheiten zu definieren.
- READ liest die Modelldaten von externen Dateien ein.
- MINIMIZE übergibt das Modell dem Lösungsalgorithmus und übernimmt anschliessend die Lösungsdaten.
- PRINT schreibt die entsprechenden Resultate-Tabellen in Dateien.

Statt einzelne Buchstaben können auch Wörter als Namen verwendet werden. So könnte z.B. *x(i,t)* durch StromMenge(Typ,Zeitzone) ersetzt werden.

Ein kurzer Überblick über die LPL-Sprache

Anhand des letzten Beispiels sollen nun die einzelnen Elemente der LPL-Sprache kurz vorgestellt werden.

Indexmengen

Eine Indexmenge ist eine ungeordnete (oder geordnete) Menge von Objekten. Die Menge der Generatorentypen *i* in unserem Beispiel ist eine solche Menge. Sie wird in LPL deklariert als

SET i; «Generatorentypen»

Die Elemente der Menge sind damit noch nicht festgelegt; diese werden als Teil der Modelldaten (Tab. III) betrachtet. Der Modellierer ist allerdings frei, bei der Deklaration auch gleich die Elemente zu definieren. So hätte man *i* auch als

SET i = /G1:G3/; «i ist die Liste aller Generatorentypen»

deklarieren können. Die beiden Elemente *G1* und *G3* getrennt durch einen Doppelpunkt definieren einen Bereich als untere und obere Grenze. Die explizite Deklaration

SET j = /G1 G2 G3/;

wäre dazu äquivalent. Für die Modelldokumentation noch besser wäre, die Kürzel *G1*,...,*G3* durch sprechende Namen zu ersetzen.

Indexmengen können selber indexiert sein. In diesem Fall besteht die Indexmenge aus einer Tupelliste und die Indexmenge heisst indexiert. Wenn *p* und *t* zwei einfache Mengen bezeichnen, die definiert sind als

SET p = / P1:P180 /; «180 Elemente»
SET t = / T1:T15 /; «15 Elemente»

Gegeben

i	Generatortypen (i=1,..,N) mit N=3
t	Zeitzonen (t=1,..,T) mit T=5...(zyklisch)

Daten

m_i	minimaler Betriebsoutput pro Generatortyp i (in Gigawatt)
M_i	maximale Kapazität des Generatortyps t (in Gigawatt)
C_i	minimale Betriebskosten pro Generatortyp i (in Franken)
E_i	Extra Betriebskosten pro Megawatt über dem Minimum je Typ i
F_i	Anschaltkosten pro Generatortyp i
L_i	Anzahl Generatoren des Typs i
D_t	geschätzte Stromnachfrage in der Zeitzone t
N_t	Länge der Zeitzone t (in Stunden)

Unbekannte

n_{it}	Anzahl Generatoren vom Typ i in Betrieb zur Zeit t
s_{it}	Anzahl gestartete Generatoren vom Typ i zur Zeit t
x_{it}	Stromproduktion von Generatoren des Typs i zur Zeit t (in gW)

Minimiere die Kosten:

$$\sum_{i=1}^{N}\sum_{t=1}^{T}\left(N_t E_i\left(x_{it}-m_i n_{it}\right)+N_t C_i n_{it}+F_i s_{it}\right)$$

mit den Restriktionen

$$\sum_{i=1}^{N} x_{it} \geq D_t \quad \text{mit } t=1,..,T \qquad \text{(Nachfragedeckung)}$$

$$\sum_{i=1}^{N} M_i n_{it} \geq \frac{115}{100} D_t \quad \text{mit } t=1,..,T \qquad \text{(15\% extra Produktionsmarge)}$$

$$m_i n_{it} \leq x_{it} \leq M_i n_{it} \quad \text{mit } i=1,..,N,\ t=1,..,T \qquad \text{(Strommengenproduktion)}$$

$$s_{it} \geq n_{it} - n_{it-1} \quad \text{mit } i=1,..,N,\ t=1,..,T \qquad \text{(Anzahl gestartete Generatoren)}$$

$$n_{it} \leq L_i \quad \text{mit } i=1,..,N,\ t=1,..,T \qquad \text{(maximale Anzahl Generatoren)}$$

$$x_{it} \geq 0,\ s_{it} \geq 0,\ n_{it} \geq 0 \text{ und ganzzahlig}$$

Bild 1 Optimierungsmodell

dann könnte eine indexierte Tupelmenge *Tupel* deklariert werden als

SET Tupel(p,t) = / P1 T2, P2 T6, P3 T7 /;

Diese Tupelliste definiert eine Relation zwischen den Elementen *p* und den Elementen *t*. Wichtig ist, dass damit nicht die gesamte Tupelliste (das Kartesische Produkt), sondern nur eine kleine Untermenge, bestehend aus nur drei Elementen, definiert wird. Ein weiteres Beipiel definiert zwei Untermengen der einfachen Menge *p* als

SET Sp1(p) = / P1 P45 P56 P67 P78 P122 /;
SET Sp2(p) = / P2 P67 P123 P145 P12 P178 /;

Statt eine explizite Liste anzugeben, können die indexierten Mengen auch aus mathematischen und logischen Ausdrücken gebildet werden. Die Vereinigungs- Schnitt- und Differenzmenge von Sp1 und Sp2 werden folgendermassen generiert:

SET Vereinigung(p) = Sp1 or Sp2;
SET Schnitt(p) = Sp1 and Sp2;
SET Differenz(p) = Sp1 and not Sp2;

Numerische Daten

Unter Koeffizienten werden numerische Werte verstanden, die in das Modell eingehen. Die einfachsten Koeffizienten besitzen nur einen Wert.

COEF TMAX INTEGER [0,100] = 50;

Koeffizienten können durch Bedingungen eingeschränkt werden. So bestimmt *INTEGER [0,100]*, dass *TMAX* nur einen ganzzahligen Wert zwischen 0 und 100 annehmen darf. Der LPL-Compiler wird einen Wert ausserhalb dieses Bereichs als Fehler erkennen. LPL besitzt auch die CHECK-Anweisung, mit welcher komplexere Bedingungen getestet und verschiedene Modellkomponenten auf Konsistenz geprüft werden können.

Koeffizienten können auch durch Indexmengen in Tabellen zusammengefasst werden. *m(i)* ist ein solcher Koeffizient in unserem Beispiel. Er deklariert für jeden Generatortyp den minimalen Betriebsoutput in Gigawatt. Die Daten können direkt in der LPL-Formulierung eingegeben werden. Gewöhnlich werden sie jedoch von externen Dateien mit Hilfe des Dateneingabe-Generators (der READ-Anweisung) gelesen und müssen nicht in LPL-Form vorliegen. Tabellen sind auch nicht auf einen Index beschränkt: zwei-, drei- oder mehrdimensionale Tabellen sind möglich. So deklariert *n(i,t)* eine zweidimensionale Variablentabelle, welche die Anzahl Generatoren vom Typ *i* angibt, die zur Zeit *t* in Betrieb sind. Daten können auch umdefiniert werden. Folgende Anweisungsfolge illustriert diesen Punkt:

COEF a = 10; «der Koeffizient a erhält den Wert 10»
COEF b = a; «dieser Wert wird nach b kopiert»
COEF a = 20; «a erhält einen neuen Wert, der alte geht verloren, b behält aber den Wert 10».

Ferner ist auch möglich, dass Daten-Tabellen durch arithmetische Ausdrücke aus anderen Tabellen gebildet werden können.

COEF a(i,j) = b[j,i];
COEF d(i,j) = a[i,j] + b[i,j];
EQUATION e(i,k) = SUM(j) a[i,j]*c[j,k];
PRINT(i,j): SUM(k) c[j,k] + a[i,j];
COEF f(i,j | i, > j);

Der erste Ausdruck kopiert die transponierte Tabelle *b* in die Tabelle *a*. Der zweite Ausdruck weist die Matrixaddition von *a* und *b* der Tabelle *d* zu. Der dritte Ausdruck definiert eine Matrixmultiplikation, führt diese aber nicht unmittelbar aus. Erst wenn *e* seinerseits in einem Ausdruck verwendet wird, wird der Ausdruck auf der rechten Seite evaluiert (Delayed Evaluation). Der vierte Ausdruck gibt die Berechnung als zweidimensionale Ta-

```
PROGRAM Stromproduktion;
SET
 i;                    "Generatorentypen"
 t;                    "Zeitzonen"

UNIT
 SFr;                  "Geldeinheit"
 MW;                   "Megawatt"
 GW=1000*MW;           "Gigawatt"
 hour;                 "Stunden"

COEF
 m(i) UNIT GW;         "minimaler Betriebsmenge pro Generatortyp i (in Gigawatt)"
 M(i) UNIT GW;         "maximale Kapazität des Generatortyps t (in Gigawatt)"
 C(i) UNIT SFr;        "minimale Betriebskosten pro Generatortyp i (in SFr)"
 E(i) UNIT SFr/MW;     "Extra Betriebskosten pro MW über dem Minimum je Typ i"
 F(i) UNIT SFr;        "Anschaltkosten pro Generatortyp i"
 L(i);                 "Anzahl von Generatoren des Typs i"
 D(t) UNIT GW;         "geschätzte Stromnachfrage zur Zeit t"
 N(t) UNIT hour;       "Länge der Zeitzone t (in Stunden)"

VAR
 n(i,t);               "Anzahl Generatoren vom Typ i in Betrieb zur Zeit t"
 s(i,t);               "Anzahl gestartete Generatoren vom Typ i zur Zeit t"
 x(i,t) UNIT GW;       "Stromproduktion des Typs i zur Zeit t (in GW)"
 n,s,x INTEGER;        {Variablen sind ganzzahlig}

MODEL {Modellrestriktionen}
 Nachfrage(t) UNIT GW: SUM(i) x >= D;
 Extrakapazitaet(t) UNIT GW: M*n >= 1.15*D;
 MinK,MaxK(t) UNIT GW: m*n <= x <= M*n;
 Gestartet(i,t): s[i,t] >= n[i,t] - n[i,(#t+t-2)%#t+1];
 ObereSchranke(i,t): n[i,t] <= L[i];

COEF dummy; {Hilfskoeffizient, wird zum Lesen verwendet}
READ FROM 'STROM1.DAT' : ROW(i) (i , m , M , C , E , F , D);
READ FROM 'STROM2.DAT' : ROW(t) (t , dummy , N , L);

MINIMIZE Kosten UNIT SFr: SUM(i,t) (N*E*(x-m*n) + N*C*n + F*s);
PRINT /VAR/; Kosten;
END
```

Bild 2 LPL-Programm

belle aus. Tabellen sind oft nur dünn besetzt. Die letzte Deklaration der zweidimensionalen Tabelle *f*, deklariert eine trianguläre Matrix: Nur die Elemente, welche die Bedingung $i > j$ erfüllen, sind definiert.

Tabelle III

Typ	min. Kapa.	max. Kapa.	Kosten bei min. Betrieb	extra Kosten pro produzierte MW	Start - kosten	Anzahl Generatoren
G1	850	2000	1000	2	2000	12
G2	1250	1750	2600	1.3	1000	10
G3	1500	4000	3000	3	500	5

Unbekannte

Die Unbekannten werden gleich definiert wie die Koeffizienten. Auch sie besitzen einen numerischen Wert, der allerdings von einem Lösungsalgorithmus bestimmt werden soll. Eine typische Deklaration einer Unbekannten ist

```
VAR x(i,t); «Anzahl Generatoren in Betrieb».
```

Nichts hindert den Modellierer daran, den Unbekannten auch Werte zuzuweisen, wie das für Koeffizienten möglich ist. Dadurch hat die ‹Unbekannte› einen bestimmten Wert, bis der Lösungsalgorithmus diesen Wert unter Umständen überschreibt. Auch die Werte der Unbekannten können durch Restriktionen eingeschränkt werden. Geläufig sind für Unbekannte eine obere oder untere Schranke. Häufig findet man auch die Beschränkung, dass eine Variable nur ganzzahlige Werte annehmen darf. Die Deklaration

```
VAR n, s, x INTEGER;
```

bestimmt, dass alle Variablen *n, s* und *x* ganzzahlig sein müssen. Wird das reservierte Wort INTEGER durch BOOLEAN oder LOGICAL ersetzt, so darf die Variable nur die beiden Werte 0 oder 1 annehmen. Der LPL-Compiler übergibt diese Informationen dem LP/MIP-Lösungsalgorithmus über die BOUND-Section und über INT-MARKERS in der COLUMN-Section im MPS-Kode (für eine eingehendere Diskussion des MPS-Kodes siehe [2]).

Das Modell

Das eigentliche Modell wird durch die MODEL-Anweisung definiert. Jede Restriktion beginnt mit einem Namen, die Zielfunktion(en) eingeschlossen. Gefolgt wird der Name von einem Doppelpunkt und der linearen Beziehung. Die Restriktionen können gleich wie die Koeffizienten oder die Unbekannten über mehrere Indexe laufen. Dadurch wird nicht nur eine einzige, sondern eine ganze Menge von Restriktionen definiert. Die Restriktion

```
MODEL Nachfrage(t) UNIT gW:
SUM(i) x >= D;
```

definiert für jedes Element der Menge *t* eine Nachfragerestriktion. Wieviele Restriktionen definiert werden, hängt einmal mehr von den Datentabellen ab. Beliebige algebraische Ausdrücke sind erlaubt, solange diese Beziehungen linear sind. Indexierte Summen werden mit dem reservierten Wort SUM eingeleitet. Der Ausdruck

```
... SUM(i) x[i,t]...
```

summiert alle *x* der Zeitzone *t* auf und ist direkt mit der algebraischen Notation zu vergleichen. Die Indexe nach *x* können bei Unzweideutigkeiten auch weggelassen werden, was bei einfachen Ausdrücken wie in diesem Falle die Lesbarkeit erhöht:

```
... SUM(i) x...
```

Die Zielfunktion beginnt mit dem reservieren Wort MINIMIZE oder MAXIMIZE, je nachdem ob die Funktion maximiert oder minimiert werden soll. Diese Anweisung ruft den Lösungsalgorithmus als externes Programm automatisch auf.

Der Lösungsalgorithmus

Der Lösungsalgorithmus gehört nicht zur LPL-Sprache. LPL besitzt aber eine offene und vom Modellierer definierbare Schnittstelle zu den meisten marktgängigen LP/MIP-Lösungspaketen. Die Anweisung MINIMIZE oder MAXIMIZE produziert zunächst aus den in MODEL deklarierten Restriktionen eine MPS-Datei, die Eingabedatei der meisten LP/MIP-Paketen. Danach wird der Lösungsalgorithmus mit den richtigen Parametern aufgerufen. Wurde das Modell erfolgreich gelöst, so werden die Lösungsdaten von LPL übernommen und den Unbekannten zugewiesen. Diese Schnittstelle ist ausführlich im Benützermanual [14] beschrieben. Um korrekt zu funktionieren, muss der Lösungsalgorithmus mindestens eine MPS-Datei übernehmen können. Die Schnittstelle zum Lösungspaket XA [15] ist standardmässig im LPL-Compiler eingebaut. Andere Lösungspakete wie CPLEX oder HyperLingo sind ebenfalls möglich.

Resultate

Mit LPL kann nicht nur ein Modell formuliert, sondern es können auch die entsprechenden Resultatetabellen produziert werden. Das reservierte Wort PRINT leitet die Tabellen-Generierung ein. Die einfachsten Tabellen liefert LPL, indem PRINT gefolgt wird von den Namen der Tabellen, die der Modellierer ausgeben möchte wie im Modell:

```
PRINT n; s; x; Kosten;
```

Diese Instruktion produziert vier Tabellen in einem vordefinierten Format. Die Anweisung kann auch komplexe Tabellen produzieren, bei denen der Benützer das Format und eine Ausgabemaske angeben kann.

Masseinheiten

Numerische Daten werden meistens in einer Masseinheit (Meter, Kilogramm usw.) angegeben. In LPL kann die Deklaration jeder numerischen Entität erweitert werden mit der Angabe der Masseinheit. Die Masseinheiten kann der Modellierer selber definieren. Dies erhöht die Lesbarkeit eines Modells. Ausserdem erlaubt dies, Ausdrücke automatisch auf Masseinheitsverträglichkeit zu prüfen. Eine fehlerhafte Formel kann dadurch vom Compiler eher entdeckt werden.

Masseinheiten werden deklariert in der UNIT-Anweisung, welche durch das reservierte Wort UNIT eingeleitet wird. Basismasse (wie Meter) werden durch den blossen Namen deklariert. Bei abgeleiteten Masseinheiten muss der entsprechende Ausdruck angegeben werden.

```
UNIT
  meter; «Basismass der Länge»
  kilo = 1000; «Mass in 1000 Einheiten»
  km = kilo × meter; «ein von Meter abgeleitetes Mass»
  cm = m/100; «ein weiteres von Meter abgeleitetes Mass»
  speed = meter/sec; «Mass der Geschwindigkeit»
```

Zeitzonen		geschätzte Nachfrage (in Megawatt)	Anzahl Stunden
t1	"Mitternacht bis 6 Uhr"	15000	6
t2	"6 - 9 Uhr"	30000	3
t3	"9 - 15 Uhr"	25000	6
t4	"15 - 18 Uhr"	40000	3
t5	"18 Uhr bis Mitternacht"	27000	6

Tabelle IV

Die Verwendung der Masseinheiten in LPL ist einfach: die Deklaration muss um die Masseinheitsangabe erweitert werden. Eine Zahl innerhalb eines Ausdrucks wird von der Angabe *[<Masseinheit>]* gefolgt. Auch der Tabellen-Generator akzeptiert Masseinheiten. So möchte man beispielsweise die Tabelle *m* in Megawatt statt Gigawatt ausgeben. Folgende Anweisung führt dies aus:

PRINT m UNIT mW;

Zu bemerken bleibt, dass die Angaben der Masseinheiten in LPL nicht obligatorisch sind.

Der Dateneingabe-Generator

Daten müssen nicht in LPL-Form vorliegen. Sie können durch den Dateneingabe-Generator, in LPL repräsentiert durch die READ-Anweisung, eingelesen werden. Dieser Generator kann eine komplexe Struktur von Daten einlesen. Ganze Tabellen können mit einer einzigen Anweisung gelesen werden. Die Anweisung

READ FROM ‹STROM1.DAT›:
ROW(i) (i, m, M, C, E, F, D);

liest von der Datei STROM1.DAT sieben Daten pro Zeile ein. Die Daten werden der Menge *i* und den numerischen Tabellen *m*, M, C, E, F und D in dieser Reihenfolge zugewiesen. Die Angabe *ROW(i)* weist den Eingabe-Generator an, das Einlesen zu wiederholen. Aus einer Datei können auch nur Teile eingelesen werden, oder Zeilen können übersprungen werden. Diese Anweisung ist sehr flexibel und mächtig. Die Erfahrungen mit dem Dateneingabe-Generator sind vielversprechend: Für ein konkretes LP-Modell von 1300 Restriktionen und 1500 Variablen musste ein Pascal-Programm von 32 Seiten geschrieben werden, welches die Modelldaten manipulierte und in die richtige Form brachte. Unter Benützung des Dateneingabe-Generators konnte dieses Programm durch einen LPL-Kode von 2 Seiten [16] ersetzt werden!

Zusammenfassung

Dieser Artikel ist eine notwendigerweise unvollständige Zusammenfassung der Modellierspache LPL. Das Benützerhandbuch [14] gibt eine detaillierte Beschreibung der Sprache. Eine umfangreiche Modellbibliothek in LPL aus verschiedenen Anwendungsgebieten wurde erstellt, um die Nützlichkeit der LPL-Sprache zu testen.

Die Entwicklung von LPL war von Anfang an motiviert durch den praktischen Einsatz von grossen Modellen. Verschiedene LP-Modelle mit 1500–2000 Restriktionen, 2000–3500 Unbekannten und einer Matrixbesetzung von 7500–12000 Elementen werden am Institut für Automation und Operations Research an der Universität Freiburg im Auftrage des Bundes gewartet. Sie sind in LPL formuliert worden [17]. Da diese Modelle noch vor kurzem einem Grossrechner zur Lösung übergeben werden mussten, stand für die LPL-Sprache die automatische Produktion des MPS-Kodes im Vordergrund. Dank der raschen Entwicklung der Personal Computer und den darauf implementierten Lösungsalgorithmen (XA, CPLEX, u.a.) ist es heute möglich, alle diese Modelle lokal auf dem PC zu lösen. Damit rückt der Modellierungszyklus (Modell ändern – Lösen – Resultate generieren) immer mehr in den Vordergrund. Die Produktion und Selektion von Resultate-Tabellen wird daher ebenso wichtig wie die Generierung des Eingabekodes für den Lösungsalgorithmus. LPL unterstützt diesen Zyklus voll, indem es automatisch einen externen Lösungsalgorithmus aufrufen und die Lösung zur weiteren Verarbeitung übernehmen kann. Der eingebaute Tabellengenerator produziert die gewünschten Resultate-Tabellen. Der Modellierer muss sich daher nicht mehr um die vielen technischen Details kümmern, sondern kann sich der eigentlichen Modellierung widmen. Die praktische Erfahrung mit den genannten Modellen hat gezeigt, dass sich der Modellierungszyklus einer Modelländerung von einigen Stunden, ja sogar Tagen, auf einige Minuten reduziert hat.

Der LPL Compiler wurde mit Turbo Pascal 6.0 von Borland Inc. entwickelt und läuft somit unter dem Betriebssystem MS/DOS. Eine Version in Ansi C wurde ebenfalls implementiert. Die Pascal-Version des LPL-Compilers ist erhältlich bei der Adresse des Verfassers.

Literatur

[1] Control Data Corp.: APEX-II Reference Manual, No. 59158100, Rev.C, Minneapolis, Minn., 1974.

[2] IBM World Trade Corporation: Matrix Generator and Report Writer (MGRW) Program Reference Manual, No. SH19-5014, New-York and Paris, 1972.

[3] Ketron Inc.: MPS III Dataform: User Manual, Arlington, Va.,1975.

[4] M.I.T. Center for Computational Research in Economics and Management Science: Datamat Reference Manual. 3rd ed., No. D0078, Cambridge, 1975.

[5] *Brooke A., Kendrick D.* and *Meeraus A.:* GAMS, A User's Guide. The Scientific Press, 1988.

[6] *Fourer R., Gay D.M.* and *Kernighan B.W.:* A Modeling Language for Mathematical Programming. Management Science 36:5 (May), 1990.

[7] *Cunningham K.* and *Schrage L.:* The Lingo Modeling Language. University of Chicago, Preliminary, Febr. 1989.

[8] *Geoffrion A.M.:* SML: A Model Definition Language for Structured Modeling. Western Management Science Institute, University of California, Los Angeles, Working Paper No. 60, revised Nov. 1989.

[9] *Blanning R.W.:* A Rational Theory of Model Management. In C.W. Holsapple and A.B. Whinston (eds.), Decision Support Systems: Theory and Application, Springer Verlag, 1987.

[10] *Dolk D.R.:* Model Management and Structured Modeling: The Role of an Information Resource Dictionary System. Communications ACM, 31(1988)6, pp. 704–718.

[11] *Greenberg H.J.:* A Primer for Analyse: A Computer-Assisted Analysis System for Mathematical Programming Models and Solutions. University of Colorado, Denver, Draft, June 28 1990.

[12] *Gass S.I.:* Documenting a Computer-Based Model. Interfaces, 14(1984)3, May–June, pp. 84–93.

[13] *Day R.E., Williams H.P.:* Magic: The Design and Use of an Interactive Modelling Language for Mathematical Programming. IMA Journal of Mathematics in Management 1(1986)1, p. 53–65.

[14] *Hürlimann T.:* Reference Manual for the LPL Modeling Language, Version 3.5. Institute for Automation and Operations Research, Working Paper No. 191, Fribourg, February 1992.

[15] Sunset Software: XA, A Professional Linear and Mixed 0/1 Integer Programming System. 1613 Chelsea Road, Suite 153, San Marino, Ca 91108, 1990.

[16] *Hürlimann T.:* The Input Generator of the Model RAP. Institute for Automation and Operations Research, Working Paper No. 190, November, Fribourg, 1991.

[17] *Hättenschwiler P., Kohlas J.:* Wissensbasierte Systeme auf der Grundlage linearer Modelle – Werkzeuge und Anwendungen. Output 18(1989)12, Goldach, Schweiz.

[18] *Hürlimann T., Kohlas J.:* LPL: A Structured Language for Linear Modeling. OR Spectrum 10(1988), pp. 55–63, Springer Verlag.

orbit
8.–12. Sept. 92 Basel
Halle 105, Stand B10

Für Unternehmer, die mit Teamwork gross werden. Ascoline.

Das formschöne Brigit 100
mit Lautsprecher, Display und
Message-Anzeige für
wirtschaftlichen Telefonkomfort.

Ob Ihre Mitarbeiter gut zusammenspielen, ist oft eine Frage der richtigen Einstellung. Deshalb fördert das Kommunikationssystem Ascoline den Teamgeist mit flexiblen Team- und Stellvertreterschaltungen, damit auch in hektischen Situationen keine Anrufe verlorengehen. Und dank einer gehörigen Portion High-Tech, bspw. die akustische Bedienerführung mittels Sprechtexten, stehen Sie mit dem System jederzeit auf du und du. Wie Sie also in Unternehmen mit 30 bis über 300 Mitarbeitern bedeutend mehr vom Telefon haben, erfahren Sie von der Ascom Business Systems AG. Rufen Sie noch heute eines unserer Regionalzentren in Ihrer Nähe an: Zürich: 01/823 14 14, Bern: 031/999 44 93, Lausanne: 021/641 42 11. Oder kontaktieren Sie Ihre zuständige Fernmeldedirektion, Telefon 113. TELECOM PTT

Teilnehmervermittlungsanlagen: ascom *denkt weiter.*