
Robert Fourer
President, AMPL Optimization Inc.
Professor Emeritus, Northwestern University
4er@ampl.com — +1 773-336-2675

Symposium on Recent Trends in Operations Research and Data Sciences
Indian Statistical Institute, Kolkata — 19 December 2017

For a great variety of large-scale optimization problems arising in Operations Research applications, it has become practical to rely on “off-the-shelf” software, without any special programming of algorithms. As a result the use of optimization within business systems has grown dramatically in the past decade.

One key factor in this success has been the adoption of model-based optimization. Using this approach, an optimization problem is conceived as a particular minimization or maximization of some function of decision variables, subject to varied equations, inequalities, and other constraints on the variables. A range of computer modeling languages have evolved to allow these optimization models to be described in a concise and readable way, separately from the data that determines the size and shape of the resulting problem that may have thousands (or even millions) of variables and constraints.

After an optimization problem is instantiated from the model and data, it is automatically put into a standard mathematical form and solved by sophisticated general-purpose algorithmic software packages. Numerous heuristic routines embedded within these packages enable them to adapt to many problem structures without any special effort from the model builder.

The evolution and current state of both modeling and solving software for optimization will be presented in the main part of this talk. The presentation will then conclude with a consideration of current trends and likely future directions.
Word cloud, ORSI presentation titles
Optimization

Given a function of some variables
- An Objective Function

Choose values of the variables to minimize or maximize the function
- Optimize the Objective

Possibly subject to some restrictions on the values of the variables
- Subject to the Constraints
Two Approaches to Optimization

An example from calculus

- Min/Max $f(x_1, \ldots, x_n)$
  … where $f$ is a smooth (differentiable) function

**Approach #1**

- Form $\nabla f(x_1, \ldots, x_n) = 0$
- Find an expression for the solution to these equations

**Approach #2**

- Choose a starting point $x^0 = (x_1^0, \ldots, x_n^0)$
- Iterate $x^{k+1} = x^k + d$, where $\nabla^2 f(x^k) \cdot d = \nabla f(x^k)$
  … until the iterates converge

What makes these different?
Where You Put the Most Effort

**Approach #1: Method-oriented**

- Finding a method for solving $\nabla f(x_1, \ldots, x_n) = 0$
  
  . . . a new challenge for each new form of $f$
  
  . . . usually requires a mathematician

**Approach #2: Model-oriented**

- Choosing $f$ to model your problem
  
  . . . same iteration method applies to any $f$ and $x^0$
  
  . . . can be implemented by general, off-the-shelf software

*Am I leaving something out?*

- Need to compute $\nabla f(x^k)$ and $\nabla^2 f(x^k)$ for any given $f
No . . . Software Handles Everything

**Modeling**
- Describing of $f$ as a function of variables

**Evaluation**
- Computing $f(x^k)$ from the description
- Computing $\nabla f(x^k)$, $\nabla^2 f(x^k)$ by automatic differentiation

**Solution**
- Applying the iterative algorithm
  - Computing the iterates
  - Testing for convergence
Example 1: Shekel Function

A small test case for solvers

Mathematical Formulation

Given

\( m \) number of locally optimal points
\( n \) number of variables

and

\( a_{ij} \) for each \( i = 1, \ldots, m \) and \( j = 1, \ldots, n \)
\( c_i \) for each \( i = 1, \ldots, m \)

Determine

\( x_j \) for each \( j = 1, \ldots, n \)

to maximize

\[
\sum_{i=1}^{m} 1/(c_i + \sum_{j=1}^{n} (x_j - a_{ij})^2)
\]
Modeling Language Formulation

Symbolic model (AMPL)

```AMPL
param m integer > 0;
param n integer > 0;
param a {1..m, 1..n};
param c {1..m};

var x {1..n};

maximize objective:
  sum {i in 1..m} 1 / (c[i] + sum {j in 1..n} (x[j] - a[i,j])^2);
```

\[
\sum_{i=1}^{m} 1/(c_i + \sum_{j=1}^{n} (x_j - a_{ij})^2)
\]
Modeling Language Data

*Explicit data (independent of model)*

```plaintext
param m := 5 ;
param n := 4 ;

param a:  1   2   3   4  :=
          1    4   4   4   4
          2    1   1   1   1
          3    8   8   8   8
          4    6   6   6   6
          5    3   7   3   7  ;

param c :=
          1   0.1
          2   0.2
          3   0.2
          4   0.4
          5   0.4  ;
```
Modeling Language Solution

*Model + data = problem instance to be solved*

```ampl
ampl: model shekelEX.mod;
ampl: data shekelEX.dat;
ampl: option solver knitro;
ampl: let {j in 1..n} x[j] := Uniform(0,10);
ampl: solve;
```

Knitro 10.2.0: Locally optimal solution.
objective 2.682860396; feasibility error 0
6 iterations; 11 function evaluations

```ampl
display x;
x [*] :=
1 5.99875
2 6.00029
3 5.99875
4 6.00029
;
```
Solution (cont’d)

Solver choice independent of model and data

```ampl
ampl: model shekelEX.mod;
ampl: data shekelEX.dat;
ampl: option solver loqo;
ampl: let {j in 1..n} x[j] := Uniform(0,10);
ampl: solve;
LOQO 7.03: optimal solution (13 iterations, 13 evaluations)
primal objective 5.055197729
dual objective 5.055197729
ampl: display x;
x [*] :=
  1  1.00013
  2  1.00016
  3  1.00013
  4  1.00016
;
```
Example 2: Protein Folding

Data
- 8 sets
- 30 indexed parameters

Variables
- 1 indexed problem variable
- 28 indexed defined variables

Objective
- Sum of 6 defined variables
Modeling Language Formulation

Problem variables & some defined variables

```plaintext
var x {i in Atoms, j in D3} := x0[i,j];

var aax {i in Angles, j in D3} = x[it[i],j] - x[jt[i],j];
var abx{ i in Angles, j in D3} = x[kt[i],j] - x[jt[i],j];
var a_axnorn {i in Angles} = sqrt(sum{j in D3} aax[i,j]^2);
var a_abdot {i in Angles} = sum {j in D3} aax[i,j]*abx[i,j];

......

var cosphi {i in Torsions} = (sum{j in D3} ax[i,j]*bx[i,j])
   / sqrt(sum{j in D3} ax[i,j]^2)
   / sqrt(sum{j in D3} bx[i,j]^2);

var term {i in Torsions} = if np[i] == 1 then cosphi[i]
   else if np[i] == 2 then 2*cosphi[i]^2 - 1
   else 4*cosphi[i]^3 - 3*cosphi[i];
```

Modeling Language Formulation

Some more defined variables

\[
\begin{align*}
\text{var } rinv14\{i \text{ in } Pairs14\} &= \frac{1}{\sqrt{\sum_{j \text{ in } D3} (x[i14[i],j] - x[j14[i],j])^2}}; \\
\text{var } r614\{i \text{ in } Pairs14\} &= \left(\left(\sigma[i14[i]] + \sigma[j14[i]]\right) \ast rinv14[i]\right)^6; \\
\text{var } rinv\{i \text{ in } Pairs\} &= \frac{1}{\sqrt{\sum_{j \text{ in } D3} (x[inb[i],j] - x[jnb[i],j])^2}}; \\
\text{var } r6\{i \text{ in } Pairs\} &= \left(\left(\sigma[inb[i]] + \sigma[jnb[i]]\right) \ast rinv[i]\right)^6;
\end{align*}
\]
Modeling Language Formulation

Components of total energy

```plaintext
var bond_energy = sum {i in Bonds} fcb[i] * 
    (sqrt( sum {j in D3} (x[ib[i],j] - x[jb[i],j])^2 ) - b0[i] ) ^ 2

var angle_energy = sum {i in Angles} fct[i] * 
    (atan2 (sqrt( sum{j in D3} 
        (abx[i,j]*a_axnorm[i] - a_abdot[i]/a_axnorm[i]*aax[i,j])^2),
        a_abdot[i] ) - t0[i]) ^ 2

var torsion_energy = 
    sum {i in Torsions} fcp[i]*(1 + cos(phase[i])*term[i])

var improper_energy = 
    sum {i in Improper} (fcr[i] * idi[i]^2);
```
Modeling Language Formulation

Components of total energy (cont’d)

```plaintext
var pair14_energy = 
    sum {i in Pairs14} ( 332.1667*q[i14[i]]*q[j14[i]]*rinv14[i]*0.5 
        + sqrt(eps[i14[i]]*eps[j14[i]])*(r614[i]^2 - 2*r614[i]) );

var pair_energy = 
    sum {i in Pairs} ( 332.1667*q[inb[i]]*q[jnb[i]]*rinv[i] 
        + sqrt(eps[inb[i]]*eps[jnb[i]])*(r6[i]^2 - 2*r6[i]) );

minimize energy:
    bond_energy + angle_energy + torsion_energy + 
    improper_energy + pair14_energy + pair_energy;
```
## Modeling Language Data

*Excerpts from parameter tables*

<table>
<thead>
<tr>
<th>param x0: 1</th>
<th>2</th>
<th>3</th>
<th>:=</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>2</td>
<td>1.0851518862529654</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>3</td>
<td>-0.35807838634224287</td>
<td>1.021365666308466</td>
<td>0</td>
</tr>
<tr>
<td>4</td>
<td>-0.36428404194337122</td>
<td>-0.50505976829103794</td>
<td>-0.90115715381950734</td>
</tr>
<tr>
<td>5</td>
<td>-0.52386736173121617</td>
<td>-0.69690490803763017</td>
<td>1.2465998798976687</td>
</tr>
</tbody>
</table>

......

<table>
<thead>
<tr>
<th>param: ib jb fcb</th>
<th>b0</th>
<th>:=</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 1 2 340.0000000</td>
<td>1.09000000</td>
<td></td>
</tr>
<tr>
<td>2 1 3 340.0000000</td>
<td>1.09000000</td>
<td></td>
</tr>
<tr>
<td>3 1 4 340.0000000</td>
<td>1.09000000</td>
<td></td>
</tr>
</tbody>
</table>

......

<table>
<thead>
<tr>
<th>param: inb jnb</th>
<th>:=</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 1 10</td>
<td></td>
</tr>
<tr>
<td>2 1 11</td>
<td></td>
</tr>
<tr>
<td>3 1 12</td>
<td></td>
</tr>
</tbody>
</table>

......
Solution

Local optimum from Knitro run

```
ampl: model pfold.mod;
ampl: data pfold3.dat;
ampl: option solver knitro;
ampl: option show_stats 1;
ampl: solve;
```

Substitution eliminates 762 variables.
Adjusted problem:
66 variables, all nonlinear
0 constraints
1 nonlinear objective; 66 nonzeros.

**Knitro 10.2.0**: Locally optimal solution.
objective $-32.38835099$; feasibility error 0
13 iterations; 25 function evaluations
## Solution

### Details from Knitro run

| Iter | Objective        | FeasError | OptError | ||Step|| | CGits |
|------|-----------------|-----------|----------|--------|--------|-------|
| 0    | -2.777135e+001  | 0.000e+000|          |        |        |       |
| 1    | -2.874955e+001  | 0.000e+000| 1.034e+001| 5.078e-001| 142   |
| 2    | -2.890054e+001  | 0.000e+000| 1.361e+001| 1.441e+000| 0     |
| ...  |                 |           |          |        |       |       |
| 11   | -3.238829e+001  | 0.000e+000| 1.601e-001| 5.705e-002| 0     |
| 12   | -3.238835e+001  | 0.000e+000| 2.517e-004| 6.216e-003| 0     |
| 13   | -3.238835e+001  | 0.000e+000| 3.269e-008| 1.994e-005| 0     |

# of function evaluations  =  25
# of gradient evaluations  =  14
# of Hessian evaluations   =  13

Total program time (secs)   =  0.044
Time spent in evaluations (secs) =  0.023
Origins of Model-Based Optimization

Linear Programming

- Linear expressions
  - cost, quality, contribution proportional to activity
- Inequalities
  - decision variables $\geq 0$
  - amount shipped $\leq$ amount available
  - nutrients provided $\geq$ nutrients needed
- Objective function
  - minimize cost, distance, deviation
  - maximize profit, flow, preference

Strongly model-based

- Solving requires an iterative algorithm
- Implementing the algorithm requires an expert
Origins of Model-Based Optimization

**Linear Programming**

- Linear expressions
  - ... *proportional* to activity
- Inequalities
  - decision variables $\geq 0$
  - shipped $\leq$ available
  - provided $\geq$ needed
- Objective function
  - min cost, distance, deviation
  - max profit, flow, preference

**Strongly model-based**

- Need an iterative algorithm
- Challenging to implement
Example 3: Multiperiod Production

Maximize profits
- Total revenue from sales
- ... less production and inventory costs

Subject to production time available
- Separate limit each week
- Inventories may be carried from one week to the next
Mathematical Formulation

*Given*

- $P$ a set of products
- $T$ the number of time periods

*and operational data*

- $r_p$ tons per hour produced, for each $p \in P$
- $i_p$ tons of initial inventory, for each $p \in P$
- $h_t$ hours available in week $T$, for each $t = 1, \ldots, T$
- $m_{pt}$ market demand for product $p$ in week $T$,
  for each $p \in P$ and $t = 1, \ldots, T$

*and objective data*

- $c_p$ unit production cost, for each $p \in P$
- $d_p$ unit inventory cost, for each $p \in P$
- $r_{pt}$ revenue per unit of product $p$ in week $T$,
  for each $p \in P$ and $t = 1, \ldots, T$
Mathematical Formulation

Determine

\[ x_{pt}^{\text{make}} \geq 0 \] tons produced, for each \( p \in P \) and \( t = 1, \ldots, T \)

\[ x_{pt}^{\text{inv}} \geq 0 \] tons inventoried at the end of period \( t \),
for each \( p \in P \) and \( t = 0, \ldots, T \)

\[ x_{pt}^{\text{sell}} \geq 0 \] tons sold, for each \( p \in P \) and \( t = 1, \ldots, T \)

to maximize

\[ \sum_{p \in P} \sum_{t=1}^{T} \left( r_{pt} x_{pt}^{\text{sell}} - c_{p} x_{pt}^{\text{make}} - d_{p} x_{pt}^{\text{inv}} \right) \]
Mathematical Formulation

Subject to

\[ \sum_{p \in P} \left( \frac{1}{r_p} \right) x_{pt}^{\text{make}} \leq h_t, \text{ for each } t = 1, \ldots, T \]

- time used must not exceed time available

\[ x_{p0}^{\text{inv}} = i_p, \text{ for each } p \in P \]

- first period inventory must be as given

\[ x_{pt}^{\text{make}} + x_{p,t-1}^{\text{inv}} = x_{pt}^{\text{sell}} + x_{pt}^{\text{inv}}, \text{ for each } p \in P \text{ and } t = 1, \ldots, T \]

- tons available must balance tons used

\[ x_{pt}^{\text{sell}} \leq m_{pt}, \text{ for each } p \in P \text{ and } t = 1, \ldots, T \]

- sales are limited by demand
The Optimization Modeling Lifecycle

1. Communicate with Client
2. Build Model
3. Prepare Data
4. Generate Optimization Problem
5. Submit Problem to Solver
6. Report & Analyze Results
Managing the Modeling Lifecycle

Goals for optimization software

- Repeat the cycle quickly and reliably
- Get results before client loses interest
- Deploy for application

Complication: two forms of an optimization problem

- Modeler’s form
  - Mathematical description, easy for people to work with
- Solver’s form
  - Explicit data structure, easy for solvers to compute with

Challenge: translate between these two forms
Matrix Generators

Write a program

- Reads data and translates to solver’s form
- Reads solver’s results and translates back

Advantages

- Power & flexibility of a general programming language or
- Convenience of a specialized matrix generation language

Disadvantages

- Challenge to debug
  * hard to check solver’s form for correctness
  * hard to distinguish modeling from programming errors
- Challenge to maintain
  * program does not look like a model
  * model is not separate from data
Over the past seven years we have perceived that the size distribution of general structure LP problems being run on commercial LP codes has remained about stable. . . . A 3000 constraint LP model is still considered large and very few LP problems larger than 6000 rows are being solved on a production basis. . . . That this distribution has not noticeably changed despite a massive change in solution economics is unexpected.

We do not feel that the linear programming user’s most pressing need over the next few years is for a new optimizer that runs twice as fast on a machine that costs half as much (although this will probably happen). Cost of optimization is just not the dominant barrier to LP model implementation. The process required to manage the data, formulate and build the model, report on and analyze the results costs far more, and is much more of a barrier to effective use of LP, than the cost/performance of the optimizer.

Why aren’t more larger models being run? It is not because they could not be useful; it is because we are not successful in using them. . . . They become unmanageable. LP technology has reached the point where anything that can be formulated and understood can be optimized at a relatively modest cost.

Modeling Languages

Describe your model

- Write your symbolic model in a computer-readable modeler’s form
- Prepare data for the model
- Let computer translate to & from the solver’s form

Disadvantages

- Need to learn a new language
- Incur overhead in translation

Advantages

- Faster modeling cycles
- More reliable modeling
- More maintainable applications
The aim of this system is to provide one representation of a model which is easily understood by both humans and machines. . . . With such a notation, the information content of the model representation is such that a machine can not only check for algebraic correctness and completeness, but also interface automatically with solution algorithms and report writers.

. . . a significant portion of total resources in a modeling exercise . . . is spent on the generation, manipulation and reporting of models. It is evident that this must be reduced greatly if models are to become effective tools in planning and decision making.

The heart of it all is the fact that solution algorithms need a data structure which, for all practical purposes, is impossible to comprehend by humans, while, at the same time, meaningful problem representations for humans are not acceptable to machines. We feel that the two translation processes required (to and from the machine) can be identified as the main source of difficulties and errors. GAMS is a system that is designed to eliminate these two translation processes, thereby lifting a technical barrier to effective modeling . . .

These two forms of a linear program — the modeler’s form and the algorithm’s form — are not much alike, and yet neither can be done without. Thus any application of linear optimization involves translating the one form to the other. This process of translation has long been recognized as a difficult and expensive task of practical linear programming.

In the traditional approach to translation, the work is divided between modeler and machine. . . .

There is also a quite different approach to translation, in which as much work as possible is left to the machine. The central feature of this alternative approach is a *modeling language* that is written by the modeler and translated by the computer. A modeling language is not a programming language; rather, it is a declarative language that expresses the modeler’s form of a linear program in a notation that a computer system can interpret.

Algebraic Modeling Languages

Formulation concept

- Define data in terms of sets & parameters
  - Analogous to database keys & records
- Define decision variables
- Minimize or maximize a function of decision variables
- Subject to equations or inequalities that constrain the values of the variables

Advantages

- Familiar
- Powerful
- Proven
## Modeling Language Formulation

### Sets, parameters, variables

<table>
<thead>
<tr>
<th>Set</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>PROD</td>
<td># products</td>
</tr>
<tr>
<td>T</td>
<td># number of weeks</td>
</tr>
<tr>
<td>rate {PROD}</td>
<td># tons per hour produced</td>
</tr>
<tr>
<td>inv0 {PROD}</td>
<td># initial inventory</td>
</tr>
<tr>
<td>avail {1..T}</td>
<td># hours available in week</td>
</tr>
<tr>
<td>market {PROD,1..T}</td>
<td># limit on tons sold in week</td>
</tr>
<tr>
<td>prodcost {PROD}</td>
<td># cost per ton produced</td>
</tr>
<tr>
<td>invcost {PROD}</td>
<td># carrying cost/ton of inventory</td>
</tr>
<tr>
<td>revenue {PROD,1..T}</td>
<td># revenue per ton sold</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Variable</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Make {PROD,1..T}</td>
<td># tons produced</td>
</tr>
<tr>
<td>Inv {PROD,0..T}</td>
<td># tons inventoried</td>
</tr>
<tr>
<td>Sell {PROD,1..T}</td>
<td># tons sold</td>
</tr>
</tbody>
</table>
Modeling Language Formulation

Objective, constraints

maximize Total_Profit:
    sum {p in PROD, t in 1..T} (revenue[p,t]*Sell[p,t] - prodcost[p]*Make[p,t] - invcost[p]*Inv[p,t])

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

subject to Init_Inv {p in PROD}:
    Inv[p,0] = inv0[p];

subject to Balance {p in PROD, t in 1..T}:
    Make[p,t] + Inv[p,t-1] = Sell[p,t] + Inv[p,t];

subject to Limit {p in PROD, t in 1..T}:
    Sell[p,t] <= market[p,t];
# Modeling Language Data

## Values for one scenario

```plaintext
param T := 4;
set PROD := bands coils;

param avail :=  1 40  2 40  3 32  4 40 ;
param rate := bands 200  coils 140 ;
param inv0 := bands 10  coils 0 ;

param prodcost := bands 10  coils 11 ;
param invcost := bands 2.5  coils 3 ;

param revenue:  1  2  3  4 :=
    bands    25    26    27    27
    coils    30    35    37    39 ;

param market:  1  2  3  4 :=
    bands    6000  6000  4000  6500
    coils    4000  2500  3500  4200 ;
```
Modeling Language Solution

Model + data = problem instance to be solved

```ampl
ampl: model steelT.mod;
ampl: data steelT.dat;
ampl: option solver cplex;
ampl: solve;

CPLEX 12.7.1.0: optimal solution; objective 515033
18 dual simplex iterations (0 in phase I)

ampl: display Make;
:        1      2      3      4      :=
bands   5990   6000   1400   2000
coils   1407   1400   3500   4200

ampl: display Sell;
:        1      2      3      4      :=
bands   6000   6000   1400   2000
coils    307   2500   3500   4200
```
Modeling Language Solution

Model + data = problem instance to be solved

```
ampl: model steelT.mod;
ampl: data steelT.dat;
ampl: option solver gurobi;
ampl: solve;

Gurobi 7.5.0: optimal solution; objective 515033
16 simplex iterations

ampl: display Make;
  :   1   2   3   4   :=
bands 5990 6000 1400 2000
coils 1407 1400 3500 4200

ampl: display Sell;
  :   1   2   3   4   :=
bands 6000 6000 1400 2000
coils 307 2500 3500 4200
```
Large-Scale Optimization Today

Model-based versus Method-based

Progress in off-the-shelf solvers
  - Mixed-integer linear
  - Constrained nonlinear

Developments in algebraic modeling languages
  - Solver-independent versus solver-specific
  - Executable versus declarative
Model-Based versus Method-Based

Model-based optimization is standard for . . .

- Diverse application areas
  - Operations research & management science
  - Business analytics
  - Engineering & science
  - Economics & finance

- Diverse kinds of users
  - Anyone who took an “optimization” class
  - Anyone else with a technical background
  - Newcomers to optimization

. . . and trends favor this direction

- Steadily improving off-the-shelf solvers
- Increasingly easy access to data
Model-Based versus Method-Based

Method-based optimization still seen in . . .

- Very large, specialized problems embedded in apps
  * Routing delivery trucks nationwide
  * Finding shortest routes in mapping apps
- Metaheuristic frameworks
  * Evolutionary methods, simulated annealing, . . .
- Computer science
  * Constraint programming
  * Training deep neural networks

. . . but with trends toward model-based optimization

- More general and powerful solvers
- Easier ways to embed models into applications
**Solvers: Mixed-Integer Linear**

*Linear with integer variables*

*Most popular large-scale model type*
- Model indivisible quantities with integer variables
- Model *logic* with binary (zero-one) variables

*Extended to quadratic problems*
- Convex and non-convex objectives
- Elliptic constraints
  - $x^T Ax \leq b, A \succeq 0$
- Conic constraints
  - $\sum_j x_j^2 \leq y^2, y \geq 0$
  - $\sum_j x_j^2 \leq yz, y \geq 0, z \geq 0$
Mixed-Integer Linear

Most successful solver category

- Multi-strategy approach to very hard problems
  - Presolve routines to reduce size, improve formulation
  - Feasibility heuristics for better upper bounds
  - Constraint ("cut") generators for better lower bounds
  - Multi-processor branching search
- Solve times reduced by many orders of magnitude
  - Better algorithmic ideas and implementations
  - Faster computers with more processors
- Continuing improvements for 25 years

Dominated by commercial solvers

- CPLEX
- Gurobi
- Xpress
Example 3: Multicommodity Network

Given

- $O$ Set of origins (factories)
- $D$ Set of destinations (stores)
- $P$ Set of products

and

- $a_{ip}$ Amount available, for each $i \in O$ and $p \in P$
- $b_{jp}$ Amount required, for each $j \in D$ and $p \in P$
- $l_{ij}$ Limit on total shipments, for each $i \in O$ and $j \in D$
- $c_{ijp}$ Shipping cost per unit, for each $i \in O$, $j \in D$, $p \in P$
- $d_{ij}$ Fixed cost for shipping any amount from $i \in O$ to $j \in D$
- $s$ Minimum total size of any shipment
- $n$ Maximum number of destinations served by any origin
Multicommodity Transportation

Mathematical Formulation

Determine

\[ X_{ijp} \] Amount of each \( p \in P \) to be shipped from \( i \in O \) to \( j \in D \)

\[ Y_{ij} \] 1 if any product is shipped from \( i \in O \) to \( j \in D \)

0 otherwise

to minimize

\[ \sum_{i \in O} \sum_{j \in D} \sum_{p \in P} c_{ijp} X_{ijp} + \sum_{i \in O} \sum_{j \in D} d_{ij} Y_{ij} \]

Total variable cost plus total fixed cost
**Multicommodity Transportation**

**Mathematical Formulation**

**Subject to**

\[ \sum_{j \in D} X_{ijp} \leq a_{ip} \quad \text{for all } i \in O, p \in P \]

Total shipments of product \( p \) out of origin \( i \)
must not exceed availability

\[ \sum_{i \in O} X_{ijp} = b_{jp} \quad \text{for all } j \in D, p \in P \]

Total shipments of product \( p \) into destination \( j \)
must satisfy requirements

\[ \sum_{p \in P} X_{ijp} \leq l_{ij} Y_{ij} \quad \text{for all } i \in O, j \in D \]

When there are shipments from origin \( i \) to destination \( j \),
the total may not exceed the limit, and \( Y_{ij} \) must be 1
Multicommodity Transportation

Mathematical Formulation

Subject to

\[ \sum_{p \in P} x_{ijp} \geq s y_{ij} \quad \text{for all } i \in O, j \in D \]

When there are shipments from origin \(i\) to destination \(j\), the total amount of shipments must be at least \(s\)

\[ \sum_{j \in D} y_{ij} \leq n \quad \text{for all } i \in O \]

Number of destinations served by origin \(i\) must be as most \(n\)
Multicommodity Transportation

AMPL Formulation

Symbolic data

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

param supply {ORIG,PROD} >= 0;  # availabilities at origins
param demand {DEST,PROD} >= 0;  # requirements at destinations
param limit {ORIG,DEST} >= 0;   # capacities of links

param vcost {ORIG,DEST,PROD} >= 0; # variable shipment cost
param fcost {ORIG,DEST} > 0;      # fixed usage cost

param minload >= 0;             # minimum shipment size
param maxserve integer > 0;     # maximum destinations served
```
Multicommodity Transportation

AMPL Formulation

Symbolic model: variables and objective

\[
\text{var } \text{Trans} \{\text{ORIG,DEST,PROD}\} \geq 0; \quad \text{# actual units to be shipped}
\]

\[
\text{var } \text{Use} \{\text{ORIG, DEST}\} \text{ binary}; \quad \text{# 1 if link used, 0 otherwise}
\]

\[
\text{minimize } \text{Total-Cost:}
\]

\[
\sum \{i \in \text{ORIG, j \in DEST, p \in PROD}\} \text{vcost}[i,j,p] \cdot \text{Trans}[i,j,p]
\]

\[
+ \sum \{i \in \text{ORIG, j \in DEST}\} \text{fcost}[i,j] \cdot \text{Use}[i,j];
\]

\[
\sum_{i \in O} \sum_{j \in D} \sum_{p \in P} c_{ijp} X_{ijp} + \sum_{i \in O} \sum_{j \in D} d_{ij} Y_{ij}
\]
Multicommodity Transportation

AMPL Formulation

Symbolic model: constraints

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

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

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

subject to Min_Ship {i in ORIG, j in DEST}:
    sum {p in PROD} Trans[i,j,p] >= minload * Use[i,j];

subject to Max_Serve {i in ORIG}:
    sum {j in DEST} Use[i,j] <= maxserve;
### AMPL Formulation

**Explicit data independent of symbolic model**

```AMPL
set ORIG := GARY CLEV PITT ;
set DEST := FRA DET LAN WIN STL FRE LAF ;
set PROD := bands coils plate ;

param supply (tr):  GARY   CLEV   PITT :=
   bands    400    700    800
   coils    800   1600   1800
   plate    200    300    300 ;

param demand (tr):  FRA   DET   LAN   WIN   STL   FRE   LAF :=
   bands    300    300    100    75    650   225   250
   coils    500    750    400   250    950   850   500
   plate    100    100     0    50    200   100   250 ;

param limit default 625 ;
param minload := 375 ;
param maxserve := 5 ;
```

**Multicommodity Transportation**
### Explicit data (continued)

```plaintext
param vcost :=
  [*,*,bands]:  FRA DET LAN WIN STL FRE LAF :=
    GARY  30  10  8  10  11  71  6
    CLEV  22  7  10  7  21  82  13
    PITT  19  11  12  10  25  83  15
  [*,*,coils]:  FRA DET LAN WIN STL FRE LAF :=
    GARY  39  14  11  14  16  82  8
    CLEV  27  9  12  9  26  95  17
    PITT  24  14  17  13  28  99  20
  [*,*,plate]:  FRA DET LAN WIN STL FRE LAF :=
    GARY  41  15  12  16  17  86  8
    CLEV  29  9  13  9  28  99  18
    PITT  26  14  17  13  31  104 20

param fcost:  FRA DET LAN WIN STL FRE LAF :=
  GARY  3000 1200 1200 1200 2500 3500 2500
  CLEV  2000 1000 1500 1200 2500 3000 2200
  PITT  2000 1200 1500 1500 2500 3500 2200
```

---

*Multicommodity Transportation*

**AMPL Formulation**

---

---
Multicommodity Transportation

AMPL Solution

Model + data = problem instance to be solved

```ampl
ampl: model multmip3.mod;
ampl: data multmip3.dat;
ampl: option solver gurobi;
ampl: solve;
Gurobi 7.0.0: optimal solution; objective 235625
332 simplex iterations
23 branch-and-cut nodes
ampl: display Use;
Use [*,*]
:   DET FRA FRE LAF LAN STL WIN  :=
  CLEV  1  1  1  0  1  1  0
  GARY  0  0  0  1  0  1  1
  PITT  1  1  1  0  1  0
;
```
Multicommodity Transportation

AMPL Solution

Solver choice independent of model and data

```ampl
ampl: model multmip3.mod;
ampl: data multmip3.dat;
ampl: option solver cplex;
ampl: solve;
CPLEX 12.7.0.0: optimal integer solution; objective 235625
135 MIP simplex iterations
0 branch-and-bound nodes
ampl: display Use;
Use [*,*]
  :  DET  FRA  FRE  LAF  LAN  STL  WIN  :=
CLEV  1  1  1  0  1  1  0
GARY  0  0  0  1  0  1  1
PITT  1  1  1  1  0  1  0
;
```

Robert Fourer, Optimization Software and Systems for OR
Indian Statistical Institute, Kolkata — 19 December 2017
Multicommodity Transportation

AMPL Solution

Solver choice independent of model and data

```
AMPL: model multip3.mod;
AMPL: data multip3.dat;
AMPL: option solver xpress;
AMPL: solve;

XPRESS 29.01: Global search complete
Best integer solution found 235625
4 integer solutions have been found, 7 branch and bound nodes

AMPL: display Use;
Use [*,*]
  :   DET  FRA  FRE  LAF  LAN  STL  WIN  :=
CLEV    1    1    1    0    1    1    0
GARY    0    0    0    1    0    1    1
PITT    1    1    1    1    0    1    0
;```

Robert Fourer, Optimization Software and Systems for OR
Indian Statistical Institute, Kolkata — 19 December 2017
Example 4: Linear Regression

*Given*
- Observed vector \( y \)
- Regressor vectors \( x_1, x_2, \ldots, x_p \)

*Choose multipliers* \( \beta_1, \beta_2, \ldots, \beta_p \) *to* . . .
- Approximate \( y \) by \( \sum_{i=1}^{p} x_i \beta_i \)
- Explain \( y \) convincingly
Approaches to Linear Regression

Method-based (traditional)

- Repeat
  - Solve a minimum-error problem
  - Remove and re-add regressors as suggested by results
- Until remaining regressors are judged satisfactory

Model-based

- Build a mixed-integer optimization model of the “best” choice of regressors
- Send model and data to an off-the-shelf solver

Linear Regression

Algebraic Formulation

Given

\[ m \quad \text{number of observations} \]
\[ n \quad \text{number of regressors} \]

and

\[ y_i \quad \text{observations, for each } i = 1, \ldots, m \]
\[ x_{ji} \quad \text{regressor values corresponding to observation } i, \]
\[ \quad \text{for each } j = 1, \ldots, n \text{ and } i = 1, \ldots, m \]
**Linear Regression**

**Algebraic Formulation**

**Determine**

- \( \beta_j \) Multiplier for regressor \( j \), for each \( j = 1, \ldots, n \)
- \( z_j \) 1 if \( \beta_j \neq 0 \): regressor \( j \) is used,
  0 if \( \beta_j = 0 \): regressor \( j \) is *not* used, for each \( j = 1, \ldots, n \)

**to minimize**

\[
\sum_{i=1}^{m} (y_i - \sum_{j=1}^{n} x_{ji}\beta_j)^2 + \Gamma \sum_{j=1}^{n} |\beta_j|
\]

Sum of squared errors
+ “lasso” term for regularization and robustness
Linear Regression

Algebraic Formulation

Subject to

\[-Mz_j \leq \beta_j \leq Mz_j \quad \text{for all } j = 1, \ldots, n\]

If the \(j^{th}\) regressor is used then \(z_j = 1\)

(where \(M\) is a reasonable bound on \(|\beta_j|\))

\[\sum_{j=1}^{n} z_j \leq k\]

At most \(k\) regressors may be used

\[z_{j_1} = \ldots = z_{j_{k(p)}} \quad \text{for } j_1, \ldots, j_{k(p)} \in G_S, p = 1, \ldots, n_{GS}\]

All regressors in each group sparsity set \(G_S\)

are either used or not used

\[z_{j_1} + z_{j_2} \leq 1 \quad \text{for all } (j_1, j_2) \in HC\]

For any pair of highly collinear regressors,

only one may be used
Linear Regression

Algebraic Formulation

Subject to

\[ \sum_{j \in I_p} z_j \leq 1 \quad \text{for all } p = 1, \ldots, n_T \]

For a regressor and any of its transformations, only one may be used

\[ z_j = 1 \quad \text{for all } j \in J \]

Specified regressors must be used

\[ \sum_{j \in S_p} z_j \leq |S_p| - 1 \quad \text{for all } p = 1, \ldots, n_S \]

Exclude previous solutions using \( \beta_j, j \in S_p \)
Solvers: **Constrained Nonlinear**

**Varied situations where linearity cannot be assumed or approximated**
- Proportionality assumptions are not valid
- Desired effects require a nonlinear function
- Underlying physical processes are inherently nonlinear

**Varied solver capabilities and requirements**
- Smooth (differentiable) or nonsmooth functions
- Local or global optimality
- Continuous or integer variables

**Varied algorithmic approaches**
- Reduced gradient
- Interior-point / barrier
- Sequential quadratic
Nonlinear with Constraints

Variety of high-quality solvers available

- Local, continuous
  * CONOPT, Ipopt, LOQO, MINOS, SNOPT
- Local, continuous or integer
  * Bonmin, Knitro
- Global
  * BARON, Couenne, LGO

Variety of development approaches

- Fully commercial
  * Knitro
- Small-scale commercial
  * BARON, CONOPT, LOQO, MINOS, SNOPT
- Free open-source
  * Bonmin, Couenne, Ipopt
Example 5: Optimal Power Flow

Given
    • Electric power flow network parameters

Minimize
    • Total active power generation

Subject to
    • Power flow balance equations
    • Voltage limits and generation capacities
Example 5: Optimal Power Flow

Running one test case

ampl: *include opf.run*
Which case?
ampl? 662

Calling the nonlinear optimization solver:

Presolve eliminates 4364 constraints and 6895 variables.
Substitution eliminates 588 variables.

Adjusted problem:
1489 variables, all nonlinear
1324 constraints, all nonlinear; 10556 nonzeros
   1195 equality constraints
   129 range constraints
1 nonlinear objective; 440 nonzeros.

**KNITRO 9.1.0: Locally optimal solution.**

objective 1986.362077; feasibility error 7.55e-06
20 iterations; 27 function evaluations
Algebraic Modeling Languages

*Design approaches*
- Declarative: specialized optimization languages
- Executable: object libraries for programming languages

*Marketing approaches*
- Solver-independent vs. solver-specific
- Commercial vs. open-source
Algebraic Modeling Languages

Declarative, Solver-Independent

Commercial systems available since the 1990s
  • AIMMS, AMPL, GAMS, MPL

Open-source systems
  • CMPL, Gnu MathProg

Many enhancements and extensions
  • Interactive development environments
  • Generalized constraint forms
  • Variety of data sources
    • spreadsheets, relational databases
  • Programming features
    • loops, tests, assignments
  • Extensions for deployment
    • APIs for embedding models in applications
    • Tools for building applications around models
Algebraic Modeling Languages

Declarative, Solver-Specific

*From developers of large commercial systems*

- OPL for CPLEX (IBM)
- MOSEL for Xpress (FICO)
- OPTMODEL for SAS/OR (SAS)
**Algebraic Modeling Languages**

**Executable**

**Concept**
- Create an algebraic modeling language inside a general-purpose programming language
- Redefine operators like + and <= to return constraint objects rather than simple values

**Advantages**
- Modeling & application development in the same programming language
- Better access to advanced solver features

**Disadvantages**
- Models descriptions are harder to write and to read
- Modeling and programming bugs are hard to separate
- Efficiency issues are more of a concern
Executable

Examples (Gurobi/Python)

```python
model.addConstrs(x[i] + x[j] <= 1
    for i in range(5) for j in range(5))
```

```python
for i,j in arcs:
    m.addConstr(gurobipy.quicksum(flow[h,i,j] for h in commodities)
            <= capacity[i,j], 'cap_%s_%s' % (i, j))
```

```python
quicksum(data)
```

A version of the Python `sum` function that is much more efficient for building large Gurobi expressions (LinExpr or QuadExpr objects). The function takes a list of terms as its argument.

Note that while `quicksum` is much faster than `sum`, it isn’t the fastest approach for building a large expression. Use `addTerms` or the `LinExpr()` constructor if you want the quickest possible expression construction.
Algebraic Modeling Languages

Executable

Commercial, solver-specific

- C++, CPLEX
- Python, Gurobi
- MATLAB, Optimization Toolbox

Open-source, solver-independent

- Python: Pyomo, PuLP
- MATLAB: YALMIP, CVX
- Julia: JuMP
- C++: FLOPC++, Rehearse