

# New and Forthcoming Developments in the AMPL Modeling Language & System



*Robert Fourer*

AMPL Optimization LLC  
[www.ampl.com](http://www.ampl.com) — 773-336-AMPL

Industrial Engineering & Management Sciences,  
Northwestern University

**OR2011**

**International Conference on Operations Research**

Zürich, Switzerland — 30 August to 2 September, 2011

Session TC-22, *Advances in Modeling Languages*

# **New and Forthcoming Developments in the AMPL Modeling Language and System**

This presentation describes planned and ongoing projects to extend and enhance the AMPL modeling language and system, with the aim of helping modelers to get optimization projects running sooner and more successfully. Following an introductory survey using a scheduling optimization example, projects are organized according to the primary aspects of AMPL that they will affect. Extensions to AMPL's core language will be designed to allow for more natural description of discrete models, through the introduction of logical and other non-arithmetic operators. New solver interfaces will automate sophisticated conversions from human analysts' formulations to the problem types that solvers recognize, providing enhanced access to nontraditional solvers in areas such as conic programming, global optimization, and hybrid constraint-integer programming. New interfaces to the AMPL system will facilitate "optimization as a service" and encourage business deployment.

# Motivation

## *Optimization modeling cycle*

- ❖ Communicate with problem owner
- ❖ Build model
- ❖ Build datasets
- ❖ Generate optimization problems
- ❖ Feed problems to solvers
- ❖ Run solvers
- ❖ Process results for analysis & reporting to client
- ❖ ***Repeat!***

## *Goals*

- ❖ Do this quickly and reliably
- ❖ Get results before client loses interest
- ❖ Deploy for application

# Example: Scheduling Optimization

## *Cover demands for workers*

- ❖ Each “shift” requires a certain number of employees
- ❖ Each employee works a certain “schedule” of shifts

## *Satisfy scheduling rules*

- ❖ Only “valid” schedules from given list may be used
- ❖ *Each schedule that is used at all must be used for at least \_\_ employees*

## *Minimize total workers needed*

- ❖ Which schedules should be used?
- ❖ How many employees should work each schedule?

# AMPL

*Algebraic modeling language: symbolic data*

```
set SHIFTS;                # shifts
param Nsched;              # number of schedules;
set SCHEDS = 1..Nsched;    # set of schedules

set SHIFT_LIST {SCHEDS} within SHIFTS;

param rate {SCHEDS} >= 0;    # pay rates
param required {SHIFTS} >= 0; # staffing requirements
param least_assign >= 0;     # min workers on any schedule used
```

# AMPL

*Algebraic modeling language: symbolic model*

```
var Work {SCHEDS} >= 0 integer;
var Use  {SCHEDS} >= 0 binary;

minimize Total_Cost:
    sum {j in SCHEDS} rate[j] * Work[j];

subject to Shift_Needs {i in SHIFTS}:
    sum {j in SCHEDS: i in SHIFT_LIST[j]} Work[j] >= required[i];

subject to Least_Use1 {j in SCHEDS}:
    least_assign * Use[j] <= Work[j];

subject to Least_Use2 {j in SCHEDS}:
    Work[j] <= (max {i in SHIFT_LIST[j]} required[i]) * Use[j];
```

# AMPL

## *Explicit data independent of symbolic model*

```
set SHIFTS := Mon1 Tue1 Wed1 Thu1 Fri1 Sat1
             Mon2 Tue2 Wed2 Thu2 Fri2 Sat2
             Mon3 Tue3 Wed3 Thu3 Fri3 ;

param Nsched := 126 ;

set SHIFT_LIST[1] := Mon1 Tue1 Wed1 Thu1 Fri1 ;
set SHIFT_LIST[2] := Mon1 Tue1 Wed1 Thu1 Fri2 ;
set SHIFT_LIST[3] := Mon1 Tue1 Wed1 Thu1 Fri3 ;
set SHIFT_LIST[4] := Mon1 Tue1 Wed1 Thu1 Sat1 ;
set SHIFT_LIST[5] := Mon1 Tue1 Wed1 Thu1 Sat2 ; .....

param required := Mon1 100 Mon2 78 Mon3 52
                  Tue1 100 Tue2 78 Tue3 52
                  Wed1 100 Wed2 78 Wed3 52
                  Thu1 100 Thu2 78 Thu3 52
                  Fri1 100 Fri2 78 Fri3 52
                  Sat1 100 Sat2 78 ;
```

# AMPL

*Solver independent of model & data*

```
ampl: model sched1.mod;
ampl: data sched.dat;

ampl: let least_assign := 7;

ampl: option solver cplex;
ampl: solve;

CPLEX 12.2.0.2: optimal integer solution; objective 266
1119 MIP simplex iterations
139 branch-and-bound nodes

ampl: option omit_zero_rows 1, display_1col 0;
ampl: display Work;

Work [*] :=
  6 28    20 9    36 7    66 11    82 18    91 25    118 18    122 36
  18 18    31 9    37 18    78 26    89 9    112 27    119 7
;
```

# AMPL

*Language independent of solver*

```
ampl: option solver gurobi;
```

```
ampl: solve;
```

```
Gurobi 4.0.1: optimal solution; objective 266
```

```
857 simplex iterations
```

```
29 branch-and-cut nodes
```

```
ampl: display Work;
```

```
Work [*] :=
```

```
  1 21    21 36    52  7    89 29    94  7    109 16    124 36  
  3  7    37 29    71 13    91 16    95 13    116 36;
```

# AMPL Scripts

## *Multiple solutions*

```
param nSols default 0;
param maxSols = 20;

set USED {1..nSols} within SCHEDULES;

subject to exclude {k in 1..nSols}:
    sum {j in USED[k]} (1-Use[j]) +
    sum {j in SCHEDULES diff USED[k]} Use[j] >= 1;

repeat {
    solve;
    display Work;
    let nSols := nSols + 1;
    let USED[nSols] := {j in SCHEDULES: Use[j] > .5};
} until nSols = maxSols;
```

# AMPL Scripts

## *Multiple solutions run*

```
ampl: include scheds.run
```

```
Gurobi 4.0.1: optimal solution; objective 266
```

```
857 simplex iterations
```

```
29 branch-and-cut nodes
```

```
Work [*] :=
```

```
  1 21    21 36    52  7    89 29    94  7    109 16    124 36  
  3  7    37 29    71 13    91 16    95 13    116 36 ;
```

```
Gurobi 4.0.1: optimal solution; objective 266
```

```
1368 simplex iterations
```

```
59 branch-and-cut nodes
```

```
Work [*] :=
```

```
  1  9    17  9    38  7    59 21    75 36    94  7    114  8    124 35  
  4 20    33 27    56  7    71 27    86  8    107  9    116 36 ;
```

# AMPL Scripts

## *Multiple solutions run (cont'd)*

```
Gurobi 4.0.1: optimal solution; objective 266
```

```
982 simplex iterations
```

```
57 branch-and-cut nodes
```

```
Work [*] :=
```

```
  2 28   16  8   38 18   75 34   86  8   108  8   115 16   121 36
  7 18   28 10   70 18   85 18   97 18   109 10   116 18 ;
```

```
Gurobi 4.0.1: optimal solution; objective 266
```

```
144 simplex iterations
```

```
Work [*] :=
```

```
  2 29   16  7   76 36   88 29   106 16   116  7   123  7
  7 36   70 28   85  7   97  7   109 29   121 21   126  7 ;
```

```
Gurobi 4.0.1: optimal solution; objective 266
```

```
122 simplex iterations
```

```
Work [*] :=
```

```
  2 15   16 20   70 15   85 21   106 16   116 21   123 21
  7 36   53 14   76 36   97 21   109 15   121  8   126  7 ;
```

# AMPL Solver Control

## *Multiple solutions*

```
option solver cplex;
option cplex_options "poolstub=sched poolcapacity=20 \
  populate=1 poolintensity=4 poolgap=0";

solve;

for {i in 1..Current.npool} {
  solution ("sched" & i & ".sol");
  display Work;
}
```

# AMPL Solver Control

## *Multiple solutions run*

```
ampl: include schedsPool.run;
CPLEX 12.2.0.2: poolstub=sched
poolcapacity=20
populate=1
poolintensity=4
poolgap=0

CPLEX 12.2.0.2: optimal integer solution; objective 266
464 MIP simplex iterations
26 branch-and-bound nodes

Wrote 20 solutions in solution pool
to files sched1.sol ... sched20.sol.

Solution pool member 1 (of 20); objective 266

Work [*] :=
  1 15    7 14    27 7    70 29    78 29    103 7    115 14
  5 21    11 7    51 7    71 21    87 21    106 38    121 36 ;
```

# AMPL Solver Control

## *Multiple solutions run (cont'd)*

Solution pool member 2 (of 20); objective 266

Work [\*] :=

```
1 7      5 8      18 7      70 29      78 36      87 14      115 14      121 36
2 28      7 14      65 7      72 7      83 21      106 31      116 7 ;
```

Solution pool member 3 (of 20); objective 266

Work [\*] :=

```
5 21      29 13      51 7      71 34      98 7      115 13
7 15      35 8      64 8      78 16      101 13      116 15
21 7      40 13      70 8      83 8      106 24      121 36 ;
```

Solution pool member 4 (of 20); objective 266

Work [\*] :=

```
2 7      11 7      40 7      71 29      87 15      106 31      121 28
5 22      23 8      64 7      78 13      101 8      115 14      126 7
7 14      29 14      70 14      83 7      102 7      116 7 ;
```

# AMPL Algorithmic Scheme

*Difficult case:* least\_assign = 19

```
ampl: model sched1.mod;
ampl: data sched.dat;
ampl: let least_assign := 19;
ampl: option solver cplex;
ampl: solve;

CPLEX 12.2.0.2: optimal integer solution; objective 269
635574195 MIP simplex iterations
86400919 branch-and-bound nodes

ampl: option omit_zero_rows 1, display_1col 0;
ampl: display Work;

Work [*] :=
  4 22    16 39    55 39    78 39    101 39    106 52    122 39
;
```

... *94.8 minutes*

# AMPL Algorithmic Scheme

## *Alternative, indirect approach*

- ❖ Step 1: Relax integrality of **Work** variables  
Solve for zero-one **Use** variables
- ❖ Step 2: Fix **Use** variables  
Solve for integer **Work** variables

*. . . not necessarily optimal, but . . .*

# AMPL Algorithmic Scheme

## *Indirect approach (script)*

```
model sched1.mod;
data sched.dat;
let least_assign := 19;

let {j in SCHEDS} Work[j].relax := 1;
solve;

fix {j in SCHEDS} Use[j];
let {j in SCHEDS} Work[j].relax := 0;
solve;
```

# AMPL Algorithmic Scheme

## *Indirect approach (run)*

```
ampl: include sched1-fix.run;
```

```
CPLEX 12.2.0.2: optimal integer solution; objective 268.5  
32630436 MIP simplex iterations  
2199508 branch-and-bound nodes
```

```
Work [*] :=
```

```
  1 24      32 19      80 19.5    107 33      126 19.5  
  3 19      66 19      90 19.5    109 19  
 10 19      72 19.5    105 19.5    121 19 ;
```

```
CPLEX 12.2.0.2: optimal integer solution; objective 269  
2 MIP simplex iterations  
0 branch-and-bound nodes
```

```
Work [*] :=
```

```
  1 24    10 19    66 19    80 19    105 20    109 19    126 20  
  3 19    32 19    72 19    90 20    107 33    121 19 ;
```

... *2.85 minutes*

# AMPL Modeling Alternatives

## *Linear constraints*

```
subject to Least_Use1 {j in SCHEDS}:  
    least_assign * Use[j] <= Work[j];  
subject to Least_Use2 {j in SCHEDS}:  
    Work[j] <= (max {i in SHIFT_LIST[j]} required[i]) * Use[j];
```

## *Logic constraints*

```
subject to Least_Use {j in SCHEDS}:  
    Use[j] = 1 ==> Work[j] >= least_assign else Work[j] = 0;
```

## *Variable domains*

```
var Work {j in SCHEDS} integer, in {0} union  
    interval [least_assign, (max {i in SHIFT_LIST[j]} required[i])];
```

# Topics

## *The company*

- ❖ People
- ❖ Business developments

## *The language*

- ❖ Varied prospective enhancements
- ❖ More natural formulations

## *The solvers*

- ❖ Conic programming
- ❖ Nontraditional alternatives

## *The system*

- ❖ APIs & IDEs
- ❖ AMPL as a service (in the cloud)

# The Company

## *Background*

- ❖ AMPL at Bell Labs (1986)
  - \* Bob Fourer, David Gay, Brian Kernighan
- ❖ AMPL commercialization (1993)
- ❖ AMPL Optimization LLC (2002)

## *Developments*

- ❖ People
- ❖ Business

# Current Principals

*Bob Fourer*

- ❖ Founder & . . .

*Dave Gay*

- ❖ Founder & . . .

*Bill Wells*

- ❖ Director of business development

???

- ❖ Currently looking for someone to join us in software development and customer support

# Business Developments

## *AMPL intellectual property*

- ❖ Full rights acquired from Alcatel-Lucent USA
  - \* corporate parent of Bell Laboratories
- ❖ More flexible licensing terms available

## *CPLEX & Gurobi for AMPL*

- ❖ CPLEX sales transferred from IBM to AMPL Optimization
- ❖ Full lineup of licensing arrangements available

## *AMPL distributors*

- ❖ New for Japan: *October Sky Co., Ltd.* →
- ❖ Others continue active
  - \* Gurobi, Ziena/Artelys
  - \* MOSEK, TOMLAB
  - \* OptiRisk



The image shows a promotional graphic for AMPL. At the top left is the AMPL logo featuring a horse head. To the right is a circular logo with a globe. Below these is a blue banner with the word "AMPL" in white. Underneath the banner, the text reads "最強の最適化モデリング言語" (Strongest optimization modeling language) and "究極のスケラビリティ" (Ultimate scalability). Below this is a paragraph of Japanese text describing AMPL's capabilities, followed by a small image of a control room or industrial setting. At the bottom right of the graphic, it says "AMPL MEANS BUSINESS".

# Academic Developments

*Highly discounted prices for academic use*

- ❖ AMPL
- ❖ Nonlinear solvers: KNITRO, MINOS, SNOPT, CONOPT

*Free MIP solvers to academic users*

- ❖ Gurobi & CPLEX
- ❖ 1-year licenses

*Free AMPL & solvers for courses*

- ❖ One-page application ([www.ampl.com/courses.html](http://www.ampl.com/courses.html))
- ❖ Single file for distribution to students
- ❖ Streamlined installation — no license file
- ❖ Expires when the course is over

# The Language

## *Versatility*

- ❖ Power & convenience
  - \* Linear and nonlinear modeling
  - \* Extensive indexing and set expressions
- ❖ Prototyping & deployment
  - \* Integrated scripting language
- ❖ Business & research
  - \* Major installations worldwide
  - \* Hundreds of citations in scientific & engineering literature

## *Plans . . .*

# The Language

## *Plans*

- ❖ Further set operations
  - \* arg min/arg max
  - \* sort set by parameter values
  - \* arbitrary selection from an unordered set
- ❖ Random parameters/variables
  - \* send as input to stochastic solvers
- ❖ Enhanced scripting
  - \* faster loops
  - \* functions defined by scripts
- ❖ *More natural formulations . . .*

# Common Areas of Confusion

## *Examples from my e-mail . . .*

- ❖ I have been trying to write a stepwise function in AMPL but I have not been able to do so:

```
fc[wh] = 100 if x[wh] <=5
        300 if 6 <= x[wh] <=10
        400 if 11 <= x[wh]
```

where **fc** and **x** are variables.

- ❖ *I have a set of nonlinear equations to be solved, and variables are binary. Even I have an xor operator in the equations. How can I implement it and which solver is suitable for it?*
- ❖ I'm a recent IE grad with just one grad level IE course under my belt. . . .

```
minimize Moves: sum{emp in GROUPA}
  (if Sqrt((XEmpA[emp] - XGrpA)^2 +
    (YEmpA[emp] - YGrpA)^2) > Ra then 1 else 0)
```

Is there some documentation on when you can and cannot use the if-then statements in AMPL (looked through the related forum posts but still a bit confused on this)?

# Common Areas of Confusion

## *Examples from my e-mail (cont'd)*

- ❖ I have a problem need to add a such kind of constraint:

Max[ sum( $P_i * H_i$ ) ];  $i$  is from 1 to 24;

in which  $P_i$  are constant and  $H_i$  need to be optimized.

Bound is  $-180 \leq H_i \leq 270$ . One of the constraints is

sum( $C_i$ ) = 0; here  $C_i = H_i$  if  $H_i > 0$  and  $C_i = H_i/1.38$  if  $H_i < 0$

Is it possible to solve this kind of problem with `lp_solve`?  
and how to setup the constraint?

- ❖ ... *is there a way to write a simple "or" statement in AMPL like in Java or C++?*
- ❖ I need to solve the following optimization problem:

Minimize  $-|x_1| - |x_2|$

subject to

$x_1 - x_2 = 3$

Do you know how to transform it to standard linear program?

# Currently Implemented

## *Extension to mixed-integer solver*

- ❖ CPLEX indicator constraints
  - \* `Use[j] = 1 ==> Work[j] >= least_assign;`

## *Translation to mixed-integer programs*

- ❖ General variable domains
  - \* `var Work {j in SCHEDULES} integer,  
in {0} union interval[lo_assign, hi_assign];`
- ❖ Separable piecewise-linear terms
  - \* `<<avail_min[t]; 0,time_penalty[t]>> Use[t]`

## *Translation to general nonlinear programs*

- ❖ Complementarity conditions
  - \* `0 <= ct[cr,u] complements  
ctcost[cr,u] + cv[cr] >= p["C",u];`

# Prospective Extensions

## *Existing operators allowed on variables*

- ❖ Nonsmooth terms
- ❖ Conditional expressions

## *New forms*

- ❖ Operators on constraints
- ❖ New aggregate operators
- ❖ Generalized indexing: variables in subscripts
- ❖ New types of variables: object-valued, set-valued

## *Solution strategies*

- ❖ Transform to standard MIPs
- ❖ *Send to alternative solvers (will return to this)*

*Extensions*

## **Piecewise-Linear Terms**

*Transportation (multiple rates)*

```
minimize Total_Cost:  sum {i in ORIG, j in DEST}
  << limit1[i,j], limit2[i,j];
    rate1[i,j], rate2[i,j], rate3[i,j] >> Trans[i,j];
```

```
minimize Total_Cost:  sum {i in ORIG, j in DEST}
  << {p in 1..npiece[i,j]-1} limit[i,j,p];
    {p in 1..npiece[i,j]} rate[i,j,p] >> Trans[i,j];
```

*Production (overtime)*

```
maximize Total_Profit:  sum {p in PROD, t in 1..T}
  (rev[p,t]*Sell[p,t] - pcost[p]*Make[p,t] - icost[p]*Inv[p,t]) -
  sum {t in 1..T} << avail_min[t]; 0,time_penalty[t] >> Use[t];
```

*Extensions*

# General Variable Domains

## *Workforce Scheduling*

```
param least_assign >= 0;  
  
var Work {j in SCHEDS} integer, in {0} union  
    interval [least_assign, (max {i in SHIFT_LIST[j]} required[i])];
```

*Extensions*

# Logical Operators

*Flow shop scheduling*

```
subj to NoConflict {i1 in JOBS, i2 in JOBS: ord(i1) < ord(i2)}:  
    Start[i2] >= Start[i1] + setTime[i1,i2] or  
    Start[i1] >= Start[i2] + setTime[i2,i1];
```

*Balanced assignment*

```
subj to NoIso {(i1,i2) in TYPE, j in ROOM}:  
    not (Assign[i1,i2,j] = 1 and  
        sum {ii1 in ADJ[i1]: (ii1,i2) in TYPE} Assign[ii1,i2,j] = 0);
```

*Extensions*

# Implication Operator

*Multicommodity flow with fixed costs*

```
subject to DefineUsedA {i in ORIG, j in DEST}:  
    Use[i,j] = 0 ==> sum {p in PROD} Trans[i,j,p] = 0;
```

```
subject to DefineUsedB {i in ORIG, j in DEST, p in PROD}:  
    Use[i,j] = 0 ==> Trans[i,j,p] = 0;
```

*Workforce planning*

```
var NoShut {m in MONTHS} binary;  
var LayoffCost {m in MONTHS} >=0;  
subj to NoShutDefn1 {m in MONTHS}:  
    NoShut[m] = 1 ==> LayoffCost[m] = 0;  
subj to NoShutDefn2 {m in MONTHS}:  
    NoShut[m] = 0 ==> LayoffCost[m] =  
        snrLayoffWages * ShutdownDays[m] * maxNumberSnrEmpl;
```

*Extensions*

# Counting Operators

*Transportation*

```
subj to MaxServe {i in ORIG}:
```

```
  card {j in DEST: sum {p in PRD} Trans[i,j,p] > 0} <= mxsrv;
```

```
subj to MaxServe {i in ORIG}:
```

```
  count {j in DEST} (sum {p in PRD} Trans[i,j,p] > 0) <= mxsrv;
```

```
subj to MaxServe {i in ORIG}:
```

```
  atmost mxsrv {j in DEST} (sum {p in PRD} Trans[i,j,p] > 0);
```

*Extensions*

# “Structure” Operators

*Assignment*

```
subj to OneJobPerMachine:  
    alldiff {j in JOBS} (MachineForJob[j]);
```

```
subj to CapacityOfMachine {k in MACHINES}:  
    numberof k {j in JOBS} (MachineForJob[j]) <= cap[k];
```

*... argument in ( ) may be a more general list*

*Extensions*

# Variables in Subscripts

*Assignment*

```
minimize TotalCost:  
  sum {j in JOBS} cost[j,MachineForJob[j]];
```

*Sequencing*

```
minimize CostPlusPenalty:  
  sum {k in 1..nSlots} setupCost[JobForSlot[k-1],JobForSlot[k]] +  
  sum {j in 1..nJobs} duePen[j] * (dueTime[j] - ComplTime[j]);  
  
subj to TimeNeeded {k in 0..nSlots-1}:  
  ComplTime[JobForSlot[k]] =  
    min( dueTime[JobForSlot[k]],  
         ComplTime[JobForSlot[k+1]]  
         - setupTime[JobForSlot[k],JobForSlot[k+1]]  
         - procTime[JobForSlot[k+1]] );
```

*Extensions*

# Object-Valued Variables

*Location*

```
set CLIENTS;  
set WHSES;  
  
param srvCost {CLIENTS, WHSES} > 0;  
param bdgCost > 0;  
  
var Serve {CLIENTS} in WHSES;  
var Open {WHSES} binary;  
  
minimize TotalCost:  
    sum {i in CLIENTS} srvCost[i, Serve[i]] +  
    bdgcost * sum {j in WHSES} Open[j];  
  
subject to OpenDefn {i in CLIENTS}:  
    Open[Serve[i]] = 1;
```

*Extensions*

# Set-Valued Variables

*Crew scheduling*

```
set SKILLset {SKILLS} within STAFF;
var CREWset {FLIGHTS} within STAFF;
. . . . .
subject to CrewSize {j in FLIGHTS}:
    card (CREWset[j]) = nbCrew[j];
subject to SkillReq {i in SKILLS, j in FLIGHTS}:
    card (SKILLset[i] inter CREWset[j]) >= nbSkills[i,j];
subject to NonConsecutive {j in FLIGHTS}:
    CREWset[j] inter CREWset[next(j)] = { };
```

# The Solvers

## *Communication while solver is active*

- ❖ Speed up multiple solves
- ❖ Support callbacks

## *Conic programming*

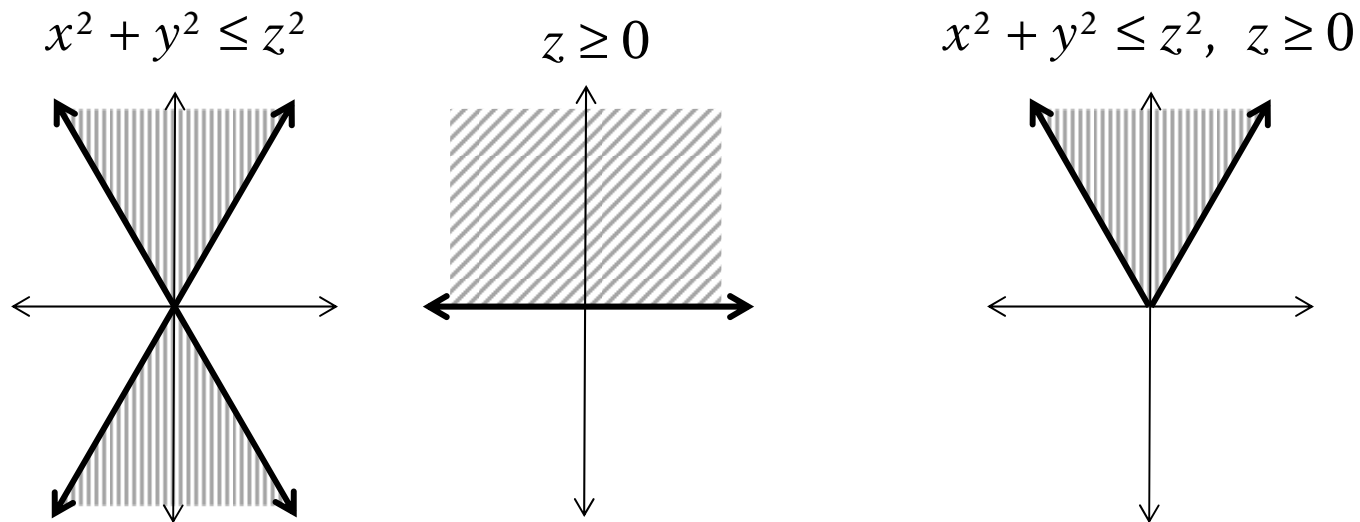
- ❖ Barrier solvers available
- ❖ Stronger modeling support needed

## *Nontraditional alternatives*

- ❖ Global optimization
- ❖ Constraint programming
- ❖ Varied hybrids

# Conic Programming

## *Standard cone*



*... convex region,  
nonsmooth boundary*

## *Rotated cone*

$$x^2 \leq yz, y \geq 0, z \geq 0 \dots$$

*Conic Quadratic*

# Conic vs. Ordinary Quadratic

*Convex quadratic constraint regions*

- ❖ Ball:  $x_1^2 + \dots + x_n^2 \leq b$
- ❖ Cone:  $x_1^2 + \dots + x_n^2 \leq y^2, y \geq 0$
- ❖ Cone:  $x_1^2 + \dots + x_n^2 \leq yz, y \geq 0, z \geq 0$

*... second-order cone programs (SOCPs)*

*Similarities*

- ❖ Describe by lists of coefficients
- ❖ Solve by extensions of LP barrier methods; extend to MIP

*Differences*

- ❖ Quadratic part not positive semi-definite
- ❖ Nonnegativity variables essential
- ❖ Boundary not quite differentiable
- ❖ ***Many convex problems can be reduced to these ...***

*Conic Quadratic*

# Equivalent Problems: Minimize

*Sums of . . .*

❖ norms or squared norms

$$* \sum_i \|F_i x + g_i\|$$

$$* \sum_i (F_i x + g_i)^2$$

❖ quadratic-linear fractions

$$* \sum_i \frac{(F_i x + g_i)^2}{a_i x + b_i}$$

*Max of . . .*

❖ norms

$$* \max_i \|F_i x + g_i\|$$

❖ logarithmic Chebychev terms

$$* \max_i |\log(F_i x) - \log(g_i)|$$

*Conic Quadratic*

# Equivalent Problems: Objective

*Products of . . .*

❖ negative powers

\*  $\min \prod_i (F_i x + g_i)^{-\alpha_i}$  for rational  $\alpha_i > 0$

❖ positive powers

\*  $\max \prod_i (F_i x + g_i)^{\alpha_i}$  for rational  $\alpha_i > 0$

*Combinations by . . .*

❖ sum, max, positive multiple

\* except log Chebychev and some positive powers

$$\text{minimize } \max \left\{ \sum_{i=1}^p (a_i x + b_i)^2, \sum_{j=1}^q \frac{\|F_j x + g_j\|^2}{y_j} \right\} + \prod_{k=1}^r (c_k x)^{-\pi_k}$$

*Conic Quadratic*

# Equivalent Problems: Constraints

*Sums of . . .*

❖ norms or squared norms

$$* \sum_i \|F_i x + g_i\| \leq F_0 x + g_0$$

$$* \sum_i (F_i x + g_i)^2 \leq (F_0 x + g_0)^2$$

❖ quadratic-linear fractions

$$* \sum_i \frac{(F_i x + g_i)^2}{a_i x + b_i} \leq F_0 x + g_0$$

*Max of . . .*

❖ norms

$$* \max_i \|F_i x + g_i\| \leq F_0 x + g_0$$

*Conic Quadratic*

# Equivalent Problems: Constraints

*Products of . . .*

❖ negative powers

$$* \sum_j \prod_i (F_{ji}x + g_{ji})^{-\alpha_{ji}} \leq F_0x + g_0 \text{ for rational } \alpha_{ji} > 0$$

❖ positive powers

$$* \sum_j - \prod_i (F_{ji}x + g_{ji})^{\alpha_{ji}} \leq F_0x + g_0 \text{ for rational } \alpha_{ji} > 0, \sum_i \alpha_{ji} \leq 1$$

*Combinations by . . .*

❖ sum, max, positive multiple

*Conic Quadratic*

# **Applications**

*Portfolio optimization with loss risk constraints*

*Traffic flow optimization*

*Engineering design of many kinds*

- ❖ Lobo, Vandenberghe, Boyd, Lebret, Applications of Second-Order Cone Programming. *Linear Algebra and Its Applications* 284 (1998) 193-228.

## *Conic Quadratic*

# Example: Sum of Norms

```
param p integer > 0;
param m {1..p} integer > 0;
param n integer > 0;

param F {i in 1..p, 1..m[i], 1..n};
param g {i in 1..p, 1..m[i]};
```

```
param p := 2 ;
param m := 1 5 2 4 ;
param n := 3 ;

param g (tr): 1 2 :=
    1 12 2
    2 7 11
    3 7 1
    4 8 0
    5 4 . ;

param F := ...
```

*Conic Quadratic*

# Example: Original Formulation

```
var x {1..n};  
minimize SumOfNorms:  
    sum {i in 1..p} sqrt(  
        sum {k in 1..m[i]} (sum {j in 1..n} F[i,k,j] * x[j] + g[i,k])^2 );
```

3 variables, all nonlinear

0 constraints

1 nonlinear objective; 3 nonzeros.

CPLEX 12.2.0.0: at12228.nl **contains a nonlinear objective.**

*Conic Quadratic*

# Example: Converted to Quadratic

```
var x {1..n};
var Max {1..p} >= 0;
minimize SumOfNorms: sum {i in 1..p} Max[i];
subj to MaxDefinition {i in 1..p}:
    sum {k in 1..m[i]} (sum {j in 1..n} F[i,k,j] * x[j] + g[i,k])^2
    <= Max[i]^2;
```

5 variables, all nonlinear  
2 constraints, all nonlinear; 8 nonzeros  
1 linear objective; 2 nonzeros.

CPLEX 12.2.0.0: **QP Hessian is not positive semi-definite.**

## Conic Quadratic

# Example: Simpler Quadratic

```
var x {1..n};
var Max {1..p} >= 0;
var Fxplusg {i in 1..p, 1..m[i]};
minimize SumOfNorms: sum {i in 1..p} Max[i];
subj to MaxDefinition {i in 1..p}:
    sum {k in 1..m[i]} Fxplusg[i,k]^2 <= Max[i]^2;
subj to FxplusgDefinition {i in 1..p, k in 1..m[i]}:
    Fxplusg[i,k] = sum {j in 1..n} F[i,k,j] * x[j] + g[i,k];
```

14 variables:  
    11 nonlinear variables  
    3 linear variables  
11 constraints; 41 nonzeros  
    2 nonlinear constraints  
    9 linear constraints  
1 linear objective; 2 nonzeros.

**CPLEX 12.2.0.0: primal optimal; objective 11.03323293**  
**11 barrier iterations**

*Conic Quadratic*

# Example: Integer Quadratic

```
var xint {1..n} integer;  
var x {j in 1..n} = xint[j] / 10;  
.....
```

Substitution eliminates 3 variables.

14 variables:

11 nonlinear variables

3 integer variables

11 constraints; 41 nonzeros

2 nonlinear constraints

9 linear constraints

1 linear objective; 2 nonzeros.

CPLEX 12.2.0.0: optimal integer solution; objective 11.12932573

88 MIP simplex iterations

19 branch-and-bound nodes

*Conic Quadratic*

# Example: Traffic Network

*Nonlinear objective due to congestion effects*

```
var Flow {(i,j) in ROADS} >= 0, <= .9999 * cap[i,j];
var Time {ROADS} >= 0;

minimize Avg_Time:
  (sum {(i,j) in ROADS} Time[i,j] * Flow[i,j]) / through;

subject to Travel_Time {(i,j) in ROADS}:
  Time[i,j] = base[i,j] + (sens[i,j]*Flow[i,j]) / (1-Flow[i,j]/cap[i,j]);

subject to Balance_Node {i in INTERS}:
  sum{(i,j) in ROADS} Flow[i,j] = sum{(j,i) in ROADS} Flow[j,i];

subject to Balance_Enter:
  sum{(EN,j) in ROADS} Flow[EN,j] = through;
```

*. . . sum of squares / linear*

*Conic Quadratic*

# **AMPL Design for SOCPs**

## *Current situation*

- ❖ Each solver recognizes some elementary forms
- ❖ Modeler must convert to these forms

## *Goal*

- ❖ Recognize many equivalent forms
- ❖ Automatically convert to a canonical form
- ❖ Further convert as necessary for each solver

# Nontraditional Solvers

## *Global nonlinear*

- ❖ BARON \*
- ❖ LINDO Global \*
- ❖ LGO

## *Constraint programming*

- ❖ IBM ILOG CP
- ❖ ECLiPSe
- ❖ SCIP \*

*\* combined with mixed-integer*

# Implementation Challenges

## *Requirements*

- ❖ Full description of functions
- ❖ Hints to algorithm
  - \* convexity, search strategy

## *Variability*

- ❖ Range of expressions recognized
  - \* hence range of conversions needed
- ❖ Design of interface

# The System

## *APIs & IDEs*

- ❖ Current options
- ❖ Alternatives under consideration

## *AMPL in the cloud*

- ❖ AMPL & solver software as a service
- ❖ Issues to be resolved

# APIs (Programming Interfaces)

## *Current options*

- ❖ AMPL scripting language
- ❖ put/get C interface
- ❖ OptiRisk Systems COM objects

## *Alternatives under consideration*

- ❖ multiplatform C interface
- ❖ object-oriented interfaces in C++, Java, Python, . . .

# Scripting Language

## *Programming extensions of AMPL syntax*

```
for {i in WIDTHS} {
  let nPAT := nPAT + 1;
  let nbr[i,nPAT] := floor (roll_width/i);
  let {i2 in WIDTHS: i2 <> i} nbr[i2,nPAT] := 0;
};
repeat {
  solve Cutting_Opt;
  let {i in WIDTHS} price[i] := Fill[i].dual;
  solve Pattern_Gen;
  printf "\n%7.2f%11.2e  ", Number, Reduced_Cost;
  if Reduced_Cost < -0.00001 then {
    let nPAT := nPAT + 1;
    let {i in WIDTHS} nbr[i,nPAT] := Use[i];
  }
  else break;
  for {i in WIDTHS} printf "%3i", Use[i];
};
```

# put/get C Interface

## *Send AMPL commands & receive output*

- ❖ Ulong **put**(GetputInfo \*g, char \*s)
- ❖ int **get**(GetputInfo \*g, char \*\*kind, char \*\*msg, Ulong \*len)

## *Limitations*

- ❖ Low-level unstructured interface
- ❖ Communication via strings

# OptiRisk COM Objects

## *Object-oriented API*

- ❖ Model management
- ❖ Data handling
- ❖ Solving

## *Limitations*

- ❖ Windows only
- ❖ Older technology
- ❖ Built on put/get interface

# API Development Directions

## *Multiplatform C interface*

- ❖ Native to AMPL code
- ❖ Similar scope to COM objects

## *Object-oriented interfaces*

- ❖ Built on C interface

# IDEs (Development Environments)

## *Previous & current options*

- ❖ AMPL Plus
- ❖ AMPL Studio

## *Alternatives under consideration*

- ❖ Multiplatform graphical interface
- ❖ Spreadsheet interface

# AMPL Plus

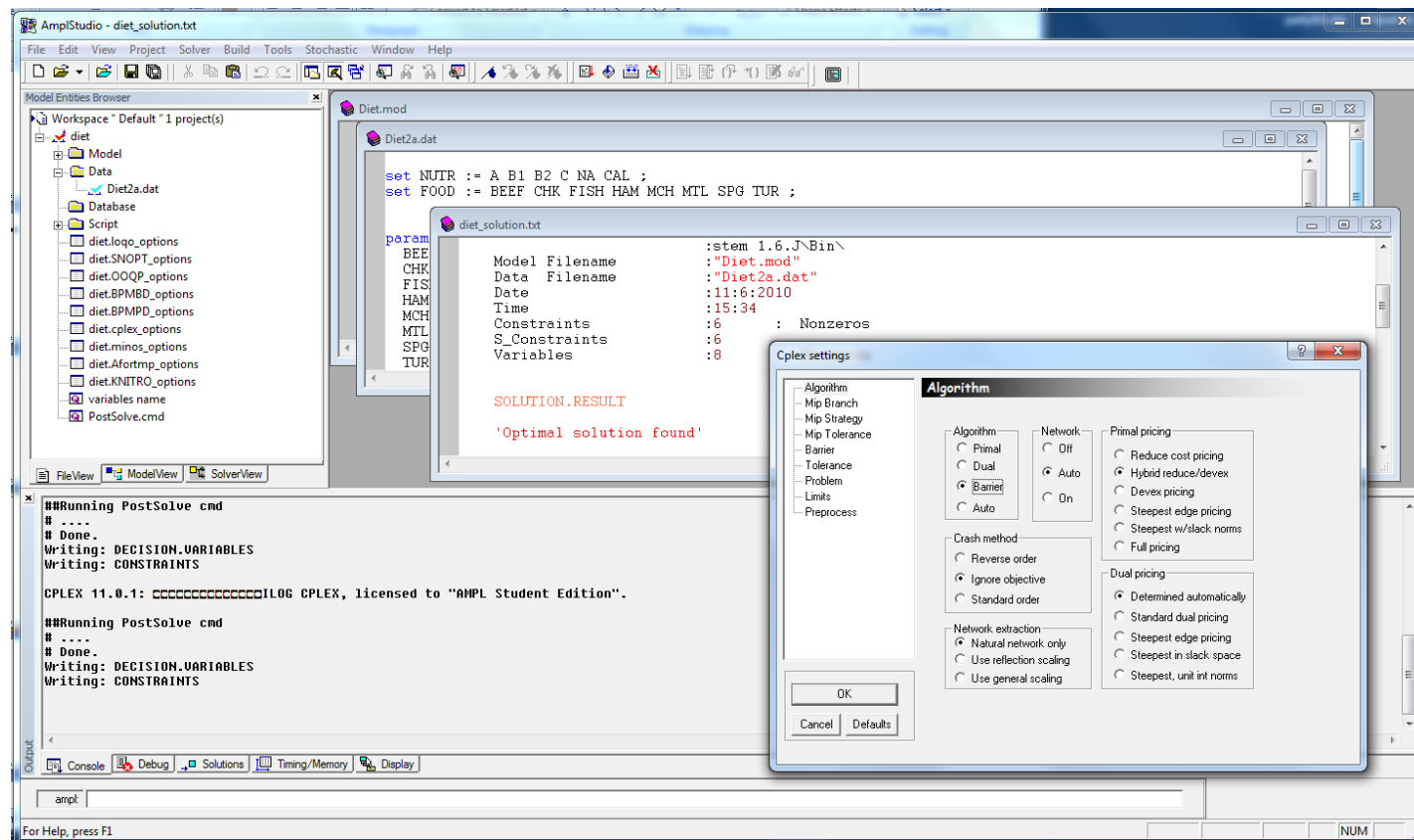
## *Menu-based GUI (1990s)*

- ❖ Created by Compass Modeling Solutions
- ❖ Discontinued by ILOG

# AMPL Studio

## *Menu-based GUI (2000s)*

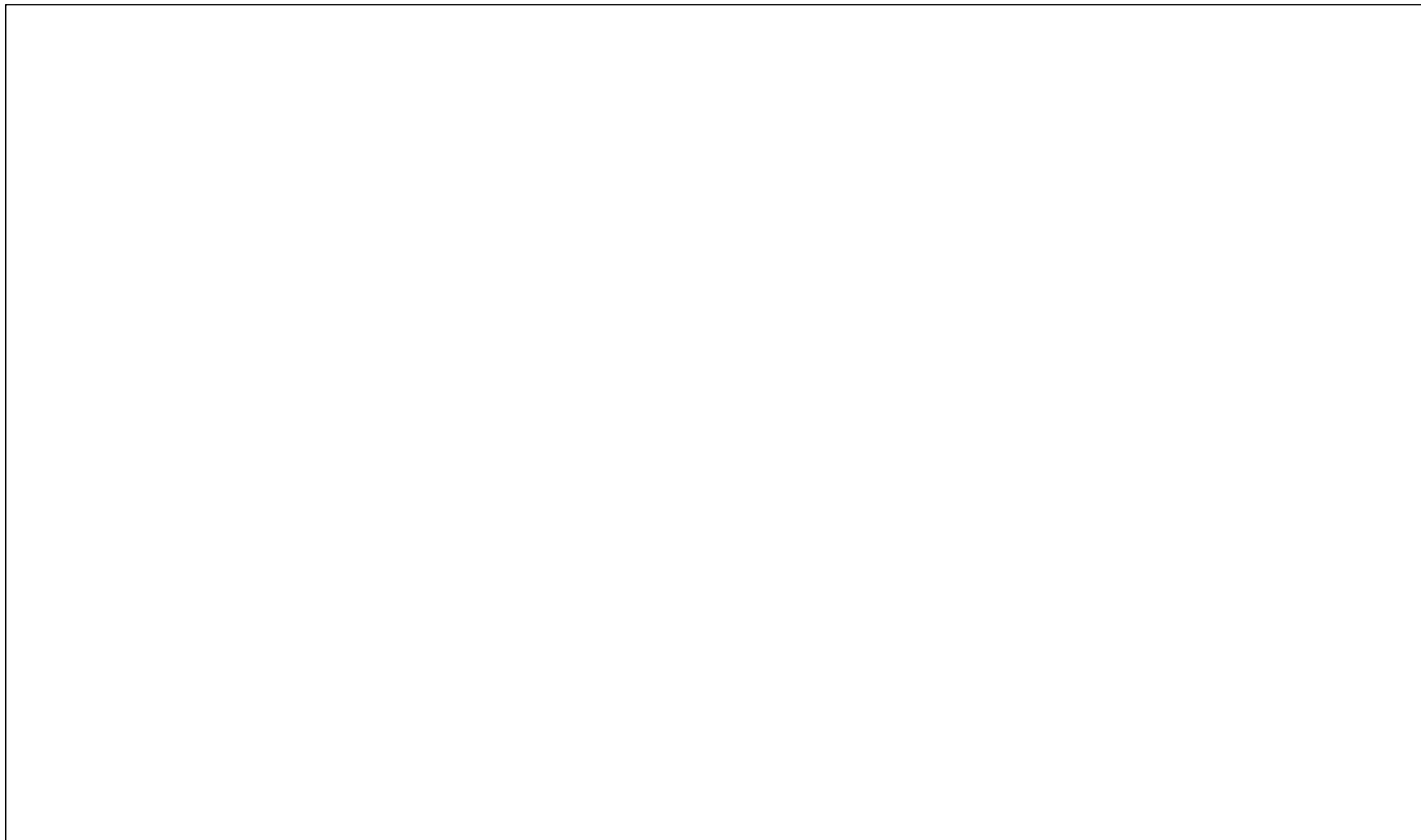
- ❖ Created by OptiRisk Systems
- ❖ Windows-based



# AMPLDev

## *Menu-based GUI (2010s)*

- ❖ Created by OptiRisk Systems
- ❖ Multi-platform



# IDE Development Directions

## *Multipatform graphical interface*

- ❖ Focused on command-line window
  - \* Same rationale as MATLAB
- ❖ Implemented using new API
- ❖ Tools for debugging, scripting, option selection . . .

## *Spreadsheet interface*

- ❖ Data in spreadsheet tables (like Excel solver)
- ❖ AMPL model in embedded application

# AMPL in the Cloud

## *AMPL as a service*

- ❖ Solvers included
  - \* optional automated solver choice
- ❖ Charges per elapsed minute
- ❖ Latest versions available

## *Issues to be resolved*

- ❖ Licensing arrangements with solvers
- ❖ Uploading & security of data
- ❖ Limitations of cloud services