

Recent & Forthcoming Developments in AMPL



Robert Fourer, David M. Gay***

AMPL Optimization LLC

www.ampl.com — +1 773-336-AMPL

* Industrial Eng & Management Sciences, Northwestern Univ

** Computer Science, University of New Mexico

Modeling Languages in Mathematical Optimization

85th GOR working group, “Real World Mathematical Optimization”

Bad Honnef, Germany — 18-19 November 2010

Recent Developments in Model and Solver Support in the AMPL Modeling Language

We describe extensions that enable the AMPL modeling language to more naturally express certain discrete and stochastic optimization problems. Only a modest number of straight-forward changes to the language are necessitated by these extensions. However we have also faced a range of challenges in conveying these extensions from models to problem instances to varied solvers, in such a way that each solver can take best advantage of the problem structure. To explain these challenges we describe several AMPL-solver links that have been implemented in the past year.

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 12.2.0.0: optimal integer solution; objective 266
419 MIP simplex iterations
39 branch-and-bound nodes

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

Work [*] :=
  3  7   18  9   37  7   66  7   82 16   112 23   124 15
  6 21   20  7   41  9   72 13   91 20   118 29
 16 13   29  7   53 13   78 20   94  9   122 21
;
```

AMPL

Language independent of solver

```
ampl: option solver gurobi;  
ampl: solve;  
  
Gurobi 4.0.0: 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  
  
;
```

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)
- ❖ AMPL commercialization (1993)
- ❖ AMPL Optimization LLC (2002)

Developments

- ❖ People
- ❖ Business

People

Bob Fourer

❖ Founder & . . .

Dave Gay

❖ Founder & . . .

Bill Wells

❖ Director of business development

Business Developments

AMPL intellectual property

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

CPLEX with AMPL

- ❖ 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
 - * MOSEK, TOMLAB
 - * OptiRisk



The image is a Japanese advertisement for AMPL. At the top left is the AMPL logo featuring a horse head. At the top right is the GOR logo. 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 '究極のスケールビリティ' (Ultimate scalability). The main body of the advertisement contains a paragraph of Japanese text describing AMPL's capabilities in solving large-scale optimization problems. To the right of this text is a small image showing a control room or data center. At the bottom right, the slogan 'AMPL MEANS BUSINESS' is displayed.

The Language

Background

- ❖ 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

- ❖ Extract elements from a set
 - * arg min/arg max
 - * arbitrary selection from an unordered set
- ❖ Sort a set by parameter values
- ❖ Define “tuples” more generally

Enhanced scripting

- ❖ Faster loops
- ❖ Arguments to script functions
- ❖ Local definitions

More natural formulations . . .

Modeling of uncertainty . . .

More Natural Formulations

Motivation

- ❖ Common formulation confusions

Extensions already implemented

Extensions in progress

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: $C_i = H_i$ 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?

More Natural Formulations

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];`

More Natural Formulations

Prospective Extensions

Existing operators allowed on variables

- ❖ Nonsmooth terms
- ❖ Conditional expressions

New forms

- ❖ Operators on constraints
- ❖ Aggregate (“global”) operators
- ❖ Generalized indexing: variables in subscripts
- ❖ New types of variables: object-valued, set-valued

More Natural Formulations

Prospective Solution Strategies

Recognize special cases

- ❖ Semi-continuous variables
- ❖ Second-order cone constraints

Transform to standard problem types

- ❖ Mixed-integer linear
- ❖ Smooth nonlinear

Send to alternative solvers

- ❖ Constraint programming
- ❖ Global optimization

. . . more on all this in discussion of solvers

More Natural Formulations

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);
```

Location-transportation

```
subj to Capacity {i in WHSE}:  
    if Build[i] = 1  
        then sum {j in CUST} Ship[i,j] <= cap[i]  
        else forall {j in CUST} Ship[i,j] = 0;
```

More Natural Formulations

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}:
```

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

More Natural Formulations

“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

More Natural Formulations

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]] );
```

More Natural Formulations

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;
```

More Natural Formulations

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)] = { };
```

Modeling of Uncertainty

Let distributions replace known values in models

- ❖ param demand {DEST} random ?
 - * like data, except value is uncertain
- ❖ var demand {DEST} random ?
 - * handled internally like variables
 - * known as “random variables”

Handle algorithmically

- ❖ Convert to deterministic equivalent
- ❖ Send distributions to solvers

Definitions of Distributions

Assigned in the model

```
param avail_mean >= 0;  
param avail_var >= 0;  
param supply {1..T} random  
    := Normal (avail_mean, avail_var);
```

Assigned as data (with dependencies)

```
param mktbas {PROD} >= 0;  
param grow_min {PROD} >= 0;  
param grow_max {PROD} >= 0;  
var demand {PROD,1..T} random;  
.....  
let {p in PROD} demand[p,1] := mktbas[p];  
let {p in PROD, t in 2..T} demand[p,t] :=  
    else demand[p,t-1] + Uniform (grow_min[p], grow_max[p]);
```

New Functions

Of random variable rv

- ❖ `Expected(rv)`
- ❖ `Moment(rv,n)`, $n = 1, 2, 3, \dots$
- ❖ `Percentile(rv,p)`, $0 \leq p \leq 100$
- ❖ `StdDev(rv)`
- ❖ `Variance(rv)`
- ❖ `Sample(rv)`
 - * samples a value from rv's distribution

Of a logical condition

- ❖ `Probability(<condition>)`
 - * where <condition> is an equality or inequality

Use in Models

Objectives

```
param cost {ORIG,DEST} random;  
var Ship {ORIG,DEST} >= 0;  
minimize TotalCost:  
    Expected (sum {i in ORIG, j in DEST} cost[i,j] * Ship[i,j]);
```

Constraints

```
var demand {DEST} random;  
var Ship {ORIG,DEST} >= 0;  
subject to MeetDemand {j in DEST}:  
    Probability (sum {i in ORIG} Ship[i,j] >= demand[j]) >= meetProb;
```

Stages: What Happens When

Each decision variable is assigned to a stage

- ❖ Stage = event followed by decision
 - * perhaps with first stage “event” known.
- ❖ Variable is split into separate copies, one for each realization of its stage
 - * but not of subsequent stages
 - * see SMPS format, myweb.dal.ca/gassmann/smeps2.htm

Suffix .stage indicates stage number

- ❖ In variable definition
 - * `var Market {PROD, t in 1..T} suffix stage t;`
- ❖ In assignment after variable definition
 - * `var Market {PROD, 1..T};`
 - ...
 - * `let {p in PROD, t in 1..T} Market[p,t].stage := t;`

Example

Stochastic diet model

- ❖ Buy in two stages
 - * constrain budget in first stage
 - * suffer random price changes in second stage
- ❖ What to buy in first stage?

```
set T = 1 .. 2; # times (stages)
var Buy {FOOD, t in T} integer >= 0 suffix stage t;
subj to FoodBounds {j in FOOD}:
    f_min[j] <= sum {t in T} Buy[j,t] <= f_max[j];
subj to InitialBudget:
    sum {j in FOOD} Buy[j,1] <= init_budget;
subj to NutrientNeeds {i in NUTR}:
    sum {j in FOOD, t in T} amt[i,j] * Buy[j,t] >= nutr_min[i];
```

Example *(cont'd)*

Random elements

```
param fall >= 0, <= 1;
param rise >= 1;

var CostAdj {FOOD} random = Uniform(fall,rise);

minimize TotalCost:
    sum {j in FOOD} cost[j] * Buy[j,1] +
    Expected (sum {j in FOOD} cost[j] * CostAdj[j] * Buy[j,2]);
```

Modeling of Uncertainty

Implemented in Prototype

Most details of random-variable handling

- ❖ Declarations
- ❖ Assignments of distributions
- ❖ Assignments of constants
- ❖ Printing and sampling (in AMPL sessions)
- ❖ Determining what the solver will see as linear

Writing problem (.nl) files with random distributions

Suffix .stage and functions of distributions

- ❖ Implicit handling of nonanticipativity
 - * .nl file indicates which vars appear in which stage

Modeling of Uncertainty

Work in Progress

Program to write .nl file for deterministic equivalent

Sampling updates to solver-interface library

Conversion routines

- ❖ Pose deterministic equivalents
 - * for example, stratified sampling such as Latin hypercube
 - * options set in AMPL would control sampling & discretization
- ❖ Write SMPS format

Solver drivers

- ❖ MSLiP (Gassmann)

Bound computations to support importance sampling

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

Simple convex quadratic constraints

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

... variables can be generalized to linear terms

Similarities

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

Differences

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

Equivalent Problems: Minimize

Sums of . . .

- ❖ norms & squared norms
- ❖ norms / linear terms

Max of . . .

- ❖ norms
- ❖ logarithmic Chebychev terms
 - * $\max_i |\log(a_i x) - \log(b_i)|$

Product of . . .

- ❖ negative powers
 - * $\prod_i (a_i x + b_i)^{-\alpha_i}$ for rational $\alpha > 0$
- ❖ minus positive powers

. . . and certain sum-max combinations

Conic Programming

Equivalent Problems: Subject to

Similar expressions involving

- ❖ norms & squared norms
- ❖ norms / linear terms
- ❖ negative & minus positive powers
- ❖ minus positive powers

. . . thesis project with Jared Erickson

Conic Programming

Modeling 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

Conic Programming

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 := ...
```

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 Programming

Example: Converted to Quadratic

```
var x {1..n};
var Max {1..p};

minimize SumOfNorms: sum {i in 1..p} Max[i];

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

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}:
    Max[i]^2 >= sum {k in 1..m[i]} Fxplusg[i,k]^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 iters

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

Multipatform 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

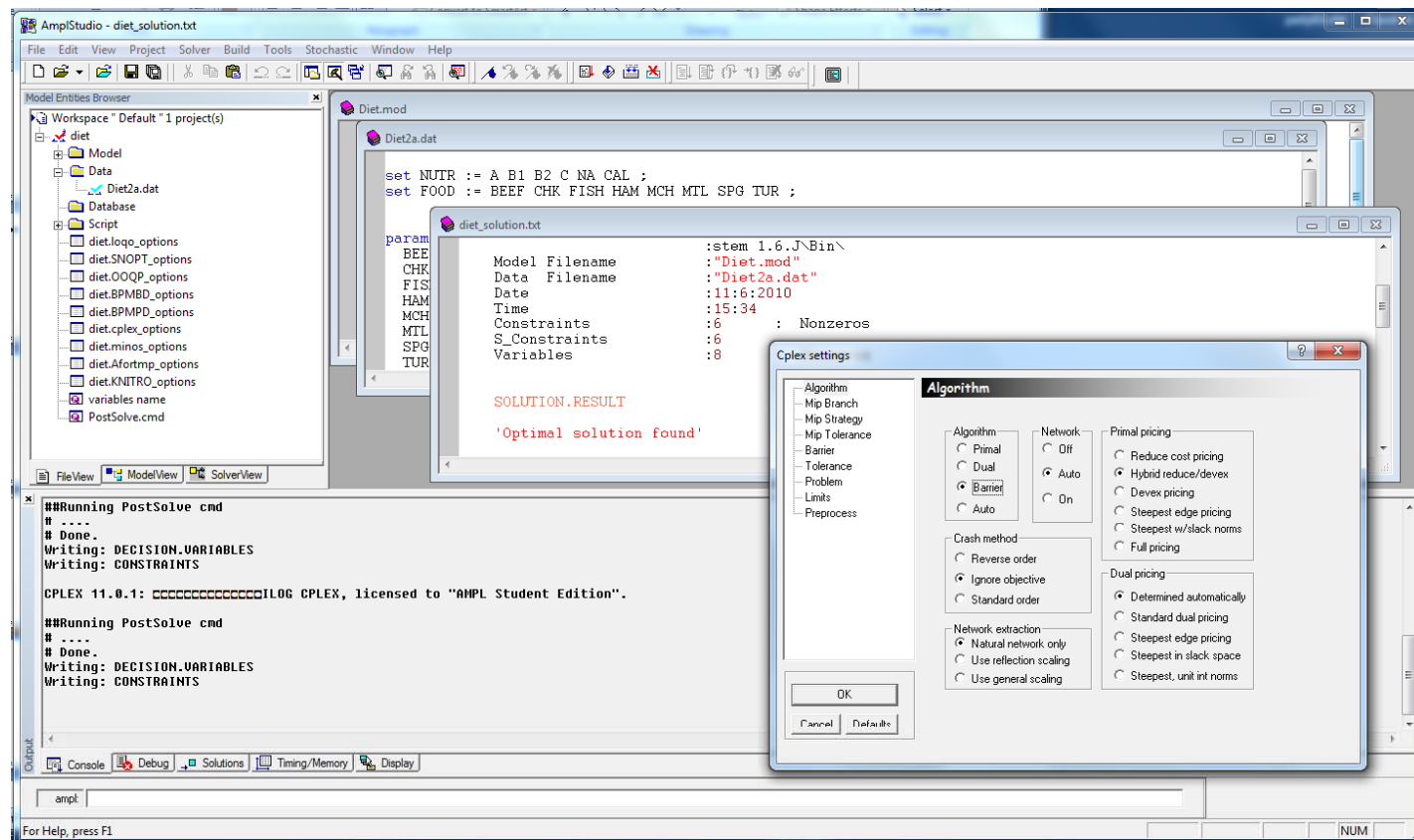
Menu-based GUI (1990s)

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

AMPL Studio

Menu-based GUI (2000s)

- ❖ Created by OptiRisk Systems
- ❖ Windows-based



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