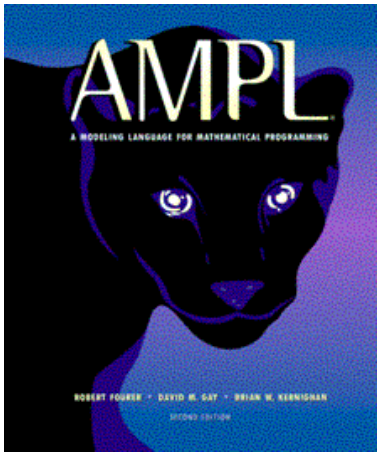
AMPL

New Solver Support
in the AMPL Modeling Language



Robert Fourer, David M. Gay

AMPL Optimization LLC, www.ampl.com

Department of Industrial Engineering & Management Sciences,
Northwestern University

Sandia National Laboratories

INFORMS Annual Meeting

Pittsburgh, Pennsylvania, November 5-8, 2006

AMPL

What it is . . .

- A modeling language for optimization
- A system to support optimization modeling

What I'll cover . . .

- Brief examples, briefer history
- Recent developments . . .
  - 64-bit versions
  - floating license manager
  - AMPL Studio graphical interface & Windows COM objects
- Solver support
  - new KNITRO 5.1 features
  - new CPLEX 10.1 features

Ex 1: Airline Fleet Assignment

```
set FLEETS;
set CITIES;
set TIMES circular;

set FLEET_LEGS within
   {f in FLEETS, c1 in CITIES, t1 in TIMES,
                 c2 in CITIES, t2 in TIMES: c1 <> c2 and t1 <> t2};

         # (f,c1,t1,c2,t2) represents the availability of fleet f
         # to cover the leg that leaves c1 at t1 and
         # whose arrival time plus turnaround time at c2 is t2

param leg_cost {FLEET_LEGS} >= 0;
param fleet_size {FLEETS} >= 0;
```

Ex 1: Derived Sets

```
set LEGS := setof {(f,c1,t1,c2,t2) in FLEET_LEGS} (c1,t1,c2,t2);

          # the set of all legs that can be covered by some fleet

set SERV_CITIES {f in FLEETS} :=
   union {(f,c1,c2,t1,t2) in FLEET_LEGS} {c1,c2};

          # for each fleet, the set of cities that it serves

set OP_TIMES {f in FLEETS, c in SERV_CITIES[f]} circular by TIMES :=
   setof {(f,c,c2,t1,t2) in FLEET_LEGS} t1 union
   setof {(f,c1,c,t1,t2) in FLEET_LEGS} t2;

          # for each fleet and city served by that fleet,
          # the set of active arrival and departure times at that city,
          # with arrival time adjusted for the turn requirement
```

Ex 1: Network Model

```
minimize Total_Cost;

node Balance {f in FLEETS, c in SERV_CITIES[f], OP_TIMES[f,c]};
          # for each fleet and city served by that fleet,
          # a node for each possible time

arc Fly {(f,c1,t1,c2,t2) in FLEET_LEGS} >= 0, <= 1,
   from Balance[f,c1,t1], to Balance[f,c2,t2],
   obj Total_Cost leg_cost[f,c1,t1,c2,t2];
          # arcs for fleet/flight assignments

arc Sit {f in FLEETS, c in SERV_CITIES[f], t in OP_TIMES[f,c]} >= 0,
   from Balance[f,c,t], to Balance[f,c,next(t)];
          # arcs for planes on the ground
```

Ex 1: Linking Constraints

```
subj to Service {(c1,t1,c2,t2) in LEGS}:
   sum {(f,c1,t1,c2,t2) in FLEET_LEGS} Fly[f,c1,t1,c2,t2] = 1;

         # each leg must be served by some fleet

subj to Capacity {f in FLEETS}:
   sum {(f,c1,t1,c2,t2) in FLEET_LEGS:
      ord(t2,TIMES) < ord(t1,TIMES)} Fly[f,c1,t1,c2,t2] +
   sum {c in SERV_CITIES[f]} Sit[f,c,last(OP_TIMES[f,c])]
                                                <= fleet_size[f];

         # the number of planes used is the number in the air at the
         # last time (arriving "earlier" than they leave) +
         # the number on the ground at the last time in each city
```

Ex 2: Catalyst Mixing (Nonlinear)

```
param nh;             # Number of subintervals
param x1_0;           # Initial condition for x1
param x2_0;           # Initial condition for x2

param alpha;          # Smoothing parameter

param tf = 1;         # Final time
param h = tf/nh;      # Width of subintervals

var u {0..nh} <= 1, >= 0;    # Control

var x1 {0..nh};              # Catalyst 1 state
var x2 {0..nh};              # Catalyst 2 state

minimize ObjWithSmoothedControl:

   -1 + x1[nh] + x2[nh]
      + alpha*h * sum {i in 0..nh-1} (u[i+1] - u[i])^2;
```

Ex 2: Constraints and Data

```
subject to DiscreteODE1 {i in 0..(nh-1)}:

   x1[i+1] = x1[i] + (h/2)*(u[i]*(10*x2[i]-x1[i])
                   + u[i+1]*(10*x2[i+1]-x1[i+1]));

subject to DiscreteODE2 {i in 0..(nh-1)}:

   x2[i+1] = x2[i] + (h/2)*(u[i]*(x1[i]-10*x2[i])
                   - (1-u[i])*x2[i] + u[i+1]*(x1[i+1]-10*x2[i+1])
                   - (1-u[i+1])*x2[i+1]);

subject to ic1: x1[0] = x1_0;
subject to ic2: x2[0] = x2_0;
```

```
param nh := 800;
param x1_0 := 1;
param x2_0:= 0;
param alpha := 0.0;

var u default 0;
var x1 default 1;
var x2 default 0;
```

Ex 2: Solving with CONOPT

```
ampl: model catmix.mod; data catmix.dat;
ampl: option solver conopt; solve;

Presolve eliminates 2 constraints and 2 variables.
Adjusted problem:
2401 variables:
        2400 nonlinear variables
        1 linear variable
1600 constraints, all nonlinear; 9596 linear nonzeros
1 linear objective; 2 nonzeros.

CONOPT 3.14D: Locally optimal; objective -0.04805583957
62 iterations; evals: nf = 94, ng = 0, nc = 214, nJ = 49, nH = 3, nHv = 12

ampl: display u, x1, x2;

:     u      x1          x2      :=
0     1   1           0
1     1   0.998759    0.00124146
2     1   0.997534    0.00246598
3     1   0.996326    0.00367377
4     1   0.995135    0.00486506
5     1   0.99396     0.00604009  .......
```

Ex 2: Solving with KNITRO

```
ampl: model catmix.mod; data catmix.dat;
ampl: option solver knitro; solve;

KNITRO 5.1:
Automatic algorithm selection: Interior/Direct

LOCALLY OPTIMAL SOLUTION FOUND.
objective -1.047950e+00; feasibility error 2.381117e-09
12 major iterations; 13 function evaluations

ampl: display u, x1, x2;

:      u            x1          x2       :=
0     0.988228     1           0
1     0.993913     0.99877     0.00123043
2     0.993783     0.997552    0.00244759
3     0.99365      0.996352    0.00364807
4     0.993512     0.995168    0.00483208
5     0.993372     0.994       0.00599986 ...
```

AMPL Optimization LLC

The new "AMPL company"

Develops, sells, and maintains AMPL

Sells related products

- AMPL Studio and AMPL COM objects, for development & deployment of AMPL-based applications

Sells solver packages

- CONOPT
- FortMP
- KNITRO

Recent Developments

64-bit versions

Flexible license manager

AMPL Studio

- Graphical user interface
- COM objects for Windows

Solver support . . .

64-bit Versions

Needed for

- Very large problems
  - Millions of variables
  - Complex set and data manipulations
- Very large databases

. . . AMPL trades space for speed

Availability

- Various incompatible platforms
  - Linux vs. Windows
  - AMD vs. Intel
- Some we have in-house, some we must borrow

License Managers

AMPL fixed

- Tied to computer hardware
- Time-limited or perpetual

AMPL floating

- Server tied to computer hardware
  - Accepting clients from prespecified subnets or IP addresses
  - Accepting all clients ***(server behind firewall)***
- Up to a specified number of clients active at once
- "Checkout" feature to keep a copy indefinitely

. . . vendors have their own license managers

AMPL Studio / Optirisk Systems Ltd.

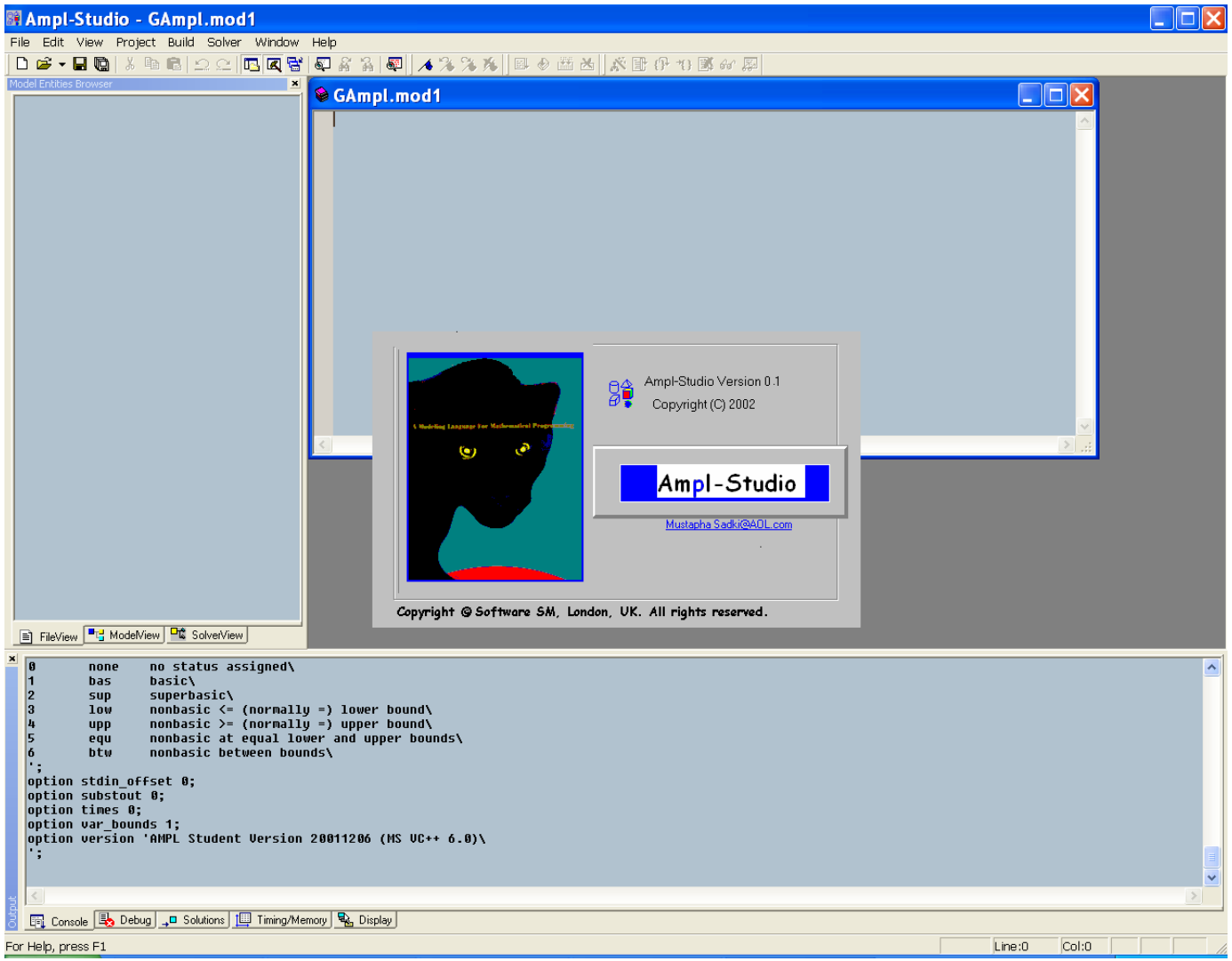
Windows IDE for AMPL

- Manage projects, edit files
- Set solver options and solve
- View results
- Run command scripts

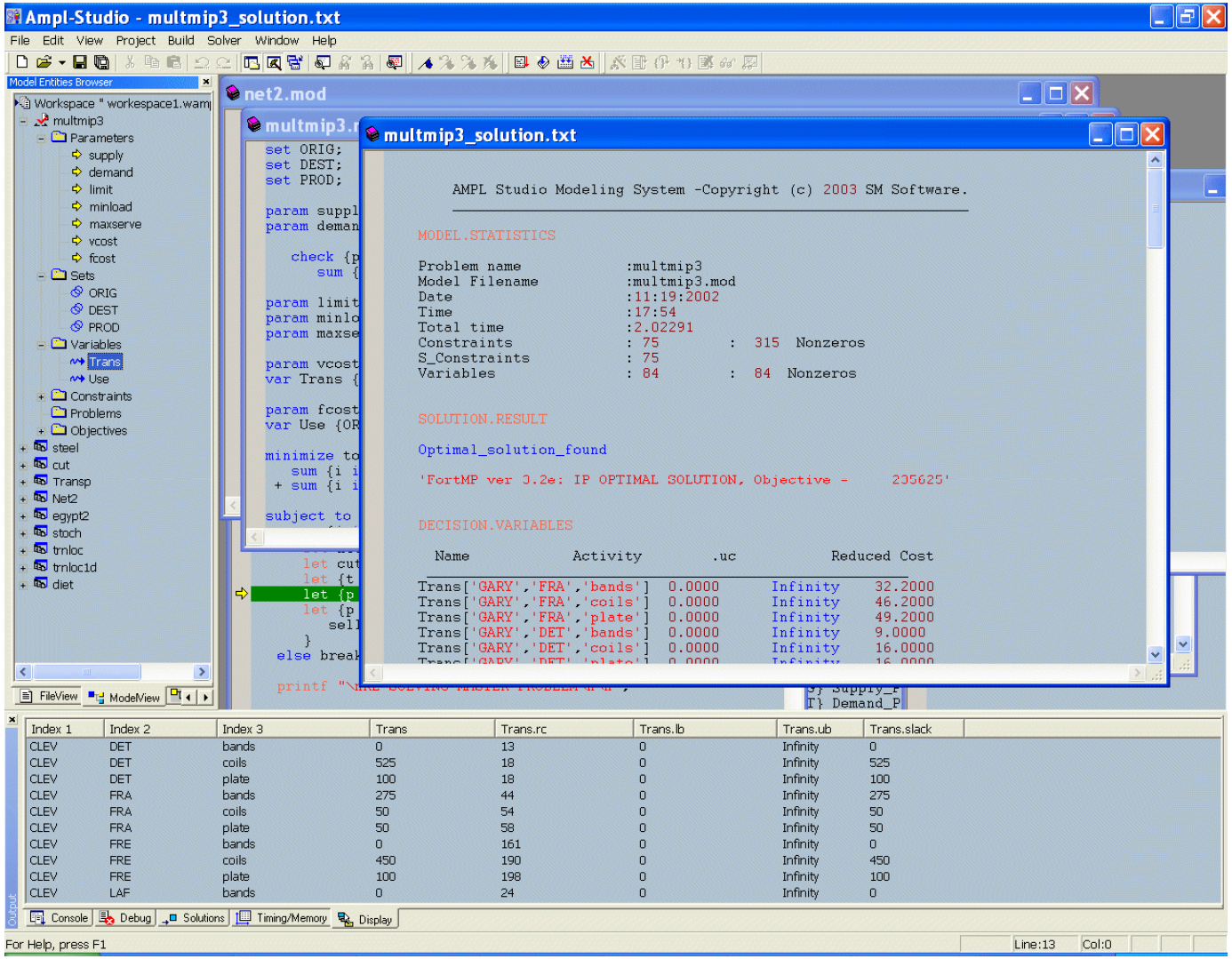
COM objects for AMPL

- Embed AMPL in applications

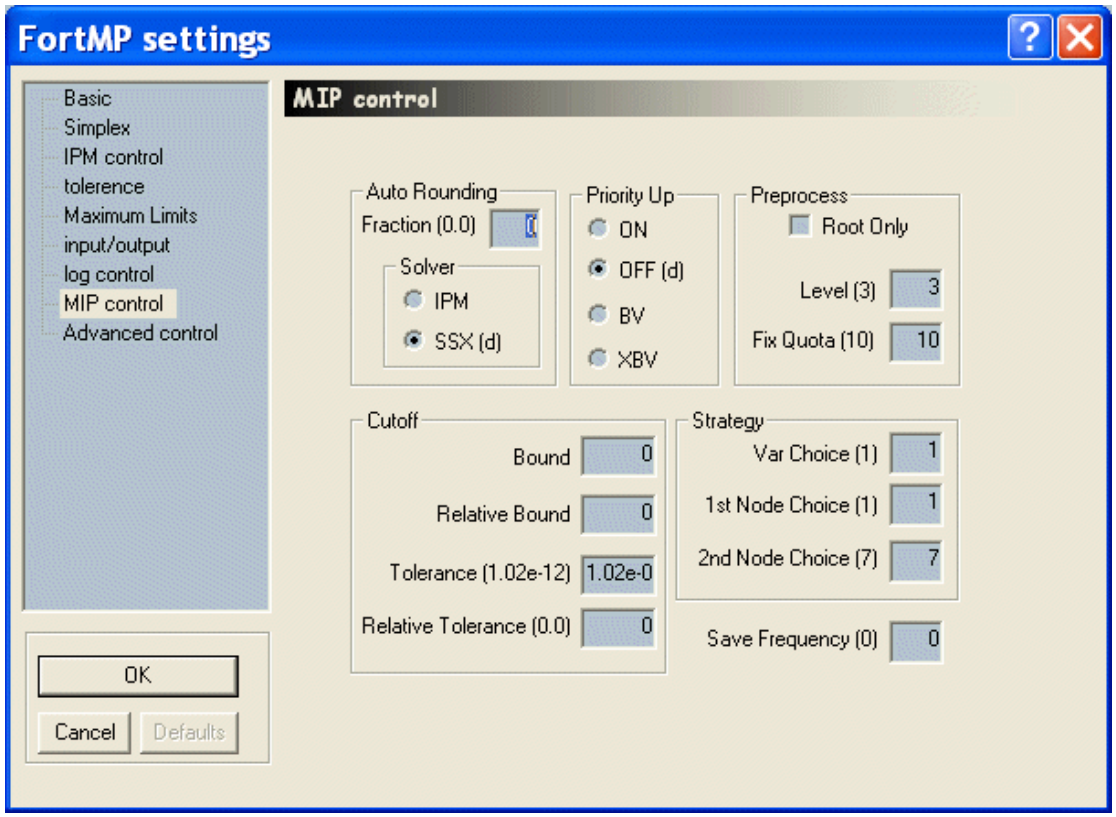
AMPL Studio *(continued)*



AMPL Studio *(continued)*



AMPL Studio *(continued)*



Solver Support

Access to solver packages

- Our company (KNITRO, CONOPT, FortMP)
- Solver developers
  - Some sell AMPL-solver packages
- **NEOS Server**

Easy to install

- Connection is simple, file-based
- Solvers can be acquired independently

Recent extensions

- **Support for new KNITRO 5.1 features**
- **Support for new CPLEX 10.1 features**

Solvers Supported by AMPL

Full list at www.ampl.com/solvers.html

- *Linear programming:* MINOS, PCx
- *Linear & linear integer programming:* CPLEX, FortMP, lp_solve, MINTO, MOSEK, SOPT, XA, Xpress-MP
- *Quadratic & convex programming:* LOQO, OOQP
- *Quadratic & quadratic integer programming:* CPLEX, FortMP, MOSEK, OOQP, Xpress-MP
- *Differentiable nonlinear programming:* CONOPT, DONLP2, IPOPT, KNITRO, LOQO, MINOS, SNOPT
- *Nondifferentiable, global, and integer nonlinear programming:* ACRS, Bonmin, CONDOR, LaGO, MINLP
- *Complementarity:* PATH
- *Problem analysis:* MProbe

AMPL Solver Support . . . via NEOS

Many solvers available, including

- *Linear programming:* MINOS, PCx
- *Linear & linear integer programming:* CPLEX, FortMP, lp_solve, MINTO, MOSEK, SOPT, XA, Xpress-MP
- *Quadratic & convex programming:* LOQO, OOQP
- *Quadratic & quadratic integer programming:* CPLEX, FortMP, MOSEK, OOQP, Xpress-MP

Minimal restrictions

- Expandable capacity
- Permissive size and time limits
- No charge

Support for KNITRO 5.1

Hessian-vector products

- Faster iterations (but perhaps more)
- Handled by the automatic differentiation routines of AMPL's solver interface

Complementarity constraints

- Equilibrium problems in economics and engineering
- Optimality conditions for nonlinear programs, bi-level linear programs, etc.

AMPL's `complements` operator

Two single inequalities

- *single-ineq1* `complements` *single-ineq2*
  - Both inequalities must hold,
  - at least one at equality

One double inequality

- *double-ineq* `complements` *expr*
  expr `complements` *double-ineq*
  - The double-inequality must hold, and
  - if at lower limit then $expr \geq 0$,
    if at upper limit then $expr \leq 0$,
    if between limits then $expr = 0$

Examples

Price-dependent demands

```
var Price {i in PROD};
var Level {j in ACT};

subject to Pri_Compl {i in PROD}:
   Price[i] >= 0 complements
      sum {j in ACT} io[i,j] * Level[j]
         >= demzero[i] - demrate[i] * Price[i];

subject to Lev_Compl {j in ACT}:
   Level[j] >= 0 complements
      sum {i in PROD} Price[i] * io[i,j] <= cost[j];
```

Examples *(cont'd)*

Prices of coal shipments

```
subject to delct {cr in creg, u in users}:

  0 <= ct[cr,u] complements

    ctcost[cr,u] + cv[cr] >= p["C",u];
```

Height of membrane

```
subject to dv {i in 1..M, j in 1..N}:

  lb[i,j] <= v[i,j] <= ub[i,j] complements

     (dy/dx) * (2*v[i,j] - v[i+1,j] - v[i-1,j])
   + (dx/dy) * (2*v[i,j] - v[i,j+1] - v[i,j-1])
   - c * dx * dy ;
```

. . . more at Complementarity Problem Net

http://www.cs.wisc.edu/cpnet/

KNITRO 5.1 *(cont'd)*

maxcrossit

- Try a *crossover* procedure to make complementarity of the solution more exact

multistart and ms_maxsolves

- Try multiple random starting points to search for improved results

Support for CPLEX 10.1

Lazy constraints

- Required by any feasible solution, but . . .
- Most will be satisfied even if left out

User cuts

- Satisfied by any integer-feasible solution, but . . .
- Cut off some fractional solutions

AMPL "suffix" settings

- `.lazy = 1` indicates a lazy constraint
- `.lazy = 2` indicates a user cut
- AMPL generates all, but CPLEX only includes some

Support for CPLEX 10.1 *(cont'd)*

Indicator constraints

- (*binary variable* = 0) implies *constraint*
- (*binary variable* = 1) implies *constraint*

AMPL "implies" operator

- Use `==>` for "implies"
- Also recognize an `else` clause
- Similarly define `<==` and `<==>`
  - `if-then-else` expressions & statements as before

Example 1

Multicommodity flow with fixed costs

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

param supply {ORIG,PROD} >= 0;  # amounts available at origins
param demand {DEST,PROD} >= 0;  # amounts required at destinations
param limit {ORIG,DEST} >= 0;

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

var Trans {ORIG,DEST,PROD} >= 0;   # actual units to be shipped
var Use {ORIG, DEST} binary;       # = 1 iff link is used

minimize total_cost:
   sum {i in ORIG, j in DEST, p in PROD} vcost[i,j,p] * Trans[i,j,p]
 + sum {i in ORIG, j in DEST} fcost[i,j] * Use[i,j];
```

Example 1 *(cont'd)*

Conventional 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 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 UseDefinition {i in ORIG, j in DEST, p in PROD}:
   Trans[i,j,p] <= min(supply[i,p], demand[j,p]) * Use[i,j];
```

Example 1 *(cont'd)*

User cuts

```
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 UseDefinition {i in ORIG, j in DEST, p in PROD}:
   Trans[i,j,p] <= min(supply[i,p], demand[j,p]) * Use[i,j];
```

Example 1 *(cont'd)*

Indicator constraint formulations

```
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;
```

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

Example 1 *(cont'd)*

Results for 3 origins, 7 destinations, 3 products

	iters	nodes	cuts used
no cuts	374	79	
all cuts	317	39	
user cuts	295	42	18
indic A	355	77	
indic B	406	56	
indic C	277	57	

Example 2

Assignment to groups with "no one isolated"

```
var Lone {(i1,i2) in ISO, j in REST} binary;

param give {ISO} default 2;
param giveTitle {TITLE} default 2;
param giveLoc {LOC} default 2;

param upperbnd {(i1,i2) in ISO, j in REST} :=
   min (ceil((number2[i1,i2]/card {PEOPLE}) * hiDine[j]) + give[i1,i2],
        hiTargetTitle[i1,j] + giveTitle[i1],
        hiTargetLoc[i2,j] + giveLoc[i2], number2[i1,i2]);

subj to Isolation1 {(i1,i2) in ISO, j in REST}:
   Assign2[i1,i2,j] <= upperbnd[i1,i2,j] * Lone[i1,i2,j];

subj to Isolation2a {(i1,i2) in ISO, j in REST}:
   Assign2[i1,i2,j] >= Lone[i1,i2,j];

subj to Isolation2b {(i1,i2) in ISO, j in REST}:
   Assign2[i1,i2,j] +
      sum {ii1 in ADJACENT[i1]: (ii1,i2) in TYPE2} Assign2[ii1,i2,j]
         >= 2 * Lone[i1,i2,j];
```

Example 2

Same using indicator constraints

```
var Lone {(i1,i2) in ISO, j in REST} binary;

subj to Isolation1 {(i1,i2) in ISO, j in REST}:
   Lone[i1,i2,j] = 0 ==> Assign2[i1,i2,j] = 0;

subj to Isolation2b {(i1,i2) in ISO, j in REST}:
   Lone[i1,i2,j] = 1 ==> Assign2[i1,i2,j] +
     sum {ii1 in ADJACENT[i1]: (ii1,i2) in TYPE2} Assign2[ii1,i2,j] >= 2;
```

Example 3

Workforce planning

```
var LayoffCost {m in MONTHS} >=0;

subj to LayoffCostDefn1 {m in MONTHS}:
   LayoffCost[m]
      <= snrLayOffWages * 31 * maxNbrSnrEmpl * (1 - NoShut[m]);

subj to LayoffCostDefn2a {m in MONTHS}:
   LayoffCost[m] - snrLayOffWages * ShutdownDays[m] * maxNbrSnrEmpl
      <= maxNbrSnrEmpl * 2 * dayAvail[m] * snrLayOffWages * NoShut[m];

subj to LayoffCostDefn2b {m in MONTHS}:
   LayoffCost[m] - snrLayOffWages * ShutdownDays[m] * maxNbrSnrEmpl
      >= -maxNbrSnrEmpl * 2 * dayAvail[m] * snrLayOffWages * NoShut[m];
```

Example 3

Same using indicator constraints

```
var LayoffCost {m in MONTHS} >=0;

subj to LayoffCostDefn1 {m in MONTHS}:
   NoShut[m] = 1 ==> LayoffCost[m] = 0;

subj to LayoffCostDefn2 {m in MONTHS}:
   NoShut[m] = 0 ==> LayoffCost[m] =
      snrLayoffWages * ShutdownDays[m] * maxNumberSnrEmpl;
```

CPLEX 10.1 *(cont'd)*

repairtries

- Given an infeasible MIP starting guess,
  how many times to try to derive a feasible one

symmetry

- Try symmetry-breaking during MIP preprocessing

repeatpresolve

- Re-apply presolve to a MIP model
  after processing at the root is otherwise complete

polishtime

- Time spent at the end
  trying to improve the best integer solution found

CPLEX 10.1 *(cont'd)*

`feasopt` *and* `feasoptobj`

- If infeasible . . .
- Find a feasible point for a relaxed problem
- Optionally find the "best" such point

`memoryemphasis`

- Conserve memory where possible

`numericalemphasis`

- Emphasize numerical stability where possible