Specifying “Logical” Conditions in AMPL Optimization Models

Robert Fourer

AMPL Optimization
www.ampl.com — 773-336-AMPL

INFORMS Annual Meeting

Phoenix, Arizona — 14-17 October 2012
Session SA15, *Software Demonstrations*

New and Forthcoming Developments in the AMPL Modeling Language and System

Optimization modelers are often stymied by the complications of converting problem logic into algebraic constraints suitable for solvers. The AMPL modeling language thus allows various logical conditions to be described directly. Additionally a new interface to the ILOG CP solver handles logic in a natural way not requiring conventional transformations.

AMPL News

Free AMPL book chapters

AMPL for Courses

Extended function library

Extended support for "logical" conditions

- ❖ AMPL driver for CPLEX Opt Studio "Concert" C++ interface
- ❖ Support for ILOG CP constraint programming solver
- ❖ Support for "logical" constraints in CPLEX

INFORMS Impact Prize to . . .

- ❖ Originators of AIMMS, AMPL, GAMS, LINDO, MPL
- ❖ Awards presented Sunday 8:30-9:45, Conv Ctr West 101
- ❖ Doors close 8:45!

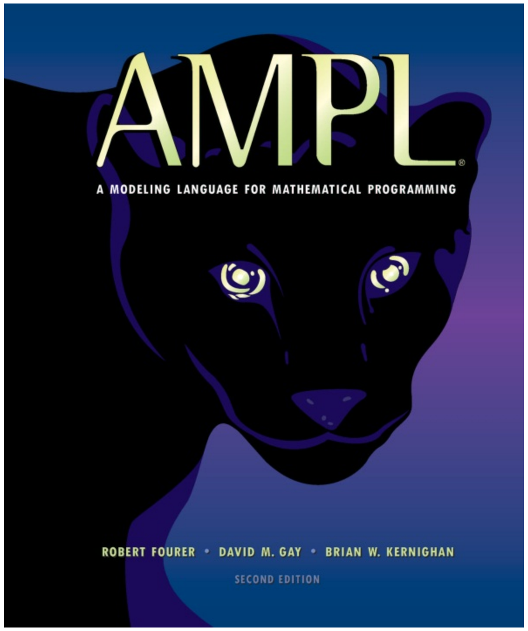
AMPL Book

Chapters now free for download

- www.ampl.com/BOOK/download.html

Bound copies remain available

- purchase from usual sources



AMPL for Courses

Streamlined for quick setup

- One-page application form for each course offering
- AMPL & solvers in one compressed file for each platform
  - ***No problem size limitations***
- Freely install on any computer supporting the course
- Freely distribute to students for their own computers
  - ***Times out after your specified course end date***

Includes top-quality solvers

- CONOPT, CPLEX, Gurobi, KNITRO, MINOS, SNOPT

Used in over 50 courses this fall

- More information: www.ampl.com/courses.html
- Application form: www.ampl.com/AMPLforCourses.pdf

. . . or stop by our booth

Extended Function Library

AMPL bindings for GNU Scientific Library

- Over 300 free open-source functions
  - probability distributions: pdf, cdf
  - special functions: Bessel, erf, gamma, . . .
  - random number generators
- Easy to “install”
  - download `amplgsl.dll` to your AMPL folder/directory

Accessible to AMPL

- Invoke `load amplgsl.dll;` at start of session
- Specify `function gsl_...;` for each function needed

Accessible to solvers

- Apply to variable expressions in objective, constraints
- 1st & 2nd derivatives provided

Extended Function Library

Example

hs069 (minimum-cost inspection plan)

```
function gsl_cdf_ugaussian_P;

param l {1..4};
param u {1..4};

var x {j in 1..4} >= l[j], <= u[j] := 1;

param a := 0.1;
param b := 1000;
param d := 1;
param n := 4;

minimize obj:
  ( a*n - (b*(exp(x[1])-1) - x[3])*x[4]/(exp(x[1]) - 1 + x[4]) )/x[1] ;

subject to constr1:
    x[3] = 2*gsl_cdf_ugaussian_P(-x[2]);

subject to constr2:
    x[4] = gsl_cdf_ugaussian_P(-x[2] + d*sqrt(n)) +
           gsl_cdf_ugaussian_P(-x[2] - d*sqrt(n));
```

Extended Function Library

Example *(cont'd)*

hs069 solution

```
model hs069.mod;
data hs069.dat;

load amplgsl.dll;

ampl: option solver knitro;

ampl: solve;

KNITRO 8.0.0: Locally optimal solution.
objective -956.7128867; feasibility error 3.41e-11
10 iterations; 11 function evaluations

ampl: display x;

1  0.0293714
2  1.19025
3  0.233947
4  0.791668
```

Licensing

GNU General Public License

Suitable for noncommercial uses

- Research
- Stand-alone modeling
- Open-source development

Contact us for commercial alternatives

- More permissive open-source licenses
- Licensed commercial libraries

Support for “Logical” Conditions

Introductory examples

- Spatial location
- Multicommodity transportation

Supported “logical” operators

- General forms
- Examples

Prospective enhancements . . .

Logic Example 1

Spatial location

- Build n observation posts
- Locate at points on an n-by-n grid
- Incur equal construction costs

Non-interference constraints

- No post blocks any other's view
- along any row, column or diagonal of the grid

Logic Example 1

Mixed-Integer Linear Formulation

One variable per grid point

```
param n integer > 0;

var Build {1..n,1..n} binary;

subj to row_conflicts {i in 1..n}:
   sum {j in 1..n} Build[i,j] = 1;

subj to col_conflicts {j in 1..n}:
   sum {i in 1..n} Build[i,j] = 1;

subj to diag1_conflicts {k in 3..2*n-1}:
   sum {i in max(1,k-n)..min(n,k-1)} Build[i,k-i] <= 1;

subj to diag2_conflicts {k in -(n-2)..(n-2)}:
   sum {i in max(k,0)+1..min(k,0)+n} Build[i,i-k] <= 1;
```

Logic Example 1

CP-Style Formulation

One variable per grid row

```
param n integer > 0;

var Col {1..n} integer >= 1 <= n;

subj to col_conflicts:   alldiff {i in 1..n} Col[i];

subj to diag1_conflicts: alldiff {i in 1..n} (Col[i] + i);

subj to diag2_conflicts: alldiff {i in 1..n} (Col[i] - i);
```

Logic Example 1

Solve with CPLEX or ILOG CP

```
ampl: model locMIP.mod;
ampl: let n := 8;

ampl: option solver cplex;
ampl: solve;

CPLEX 12.4.0.1: optimal integer solution; objective 0
26 MIP simplex iterations
0 branch-and-bound nodes
Objective = find a feasible point.
```

```
ampl: model locCP.mod;
ampl: let n := 8;

ampl: option solver ilogcp;

ampl: solve;
ilogcp 12.4.0: feasible solution
1731 choice points, 1458 fails
```

Logic Example 1

Solve Times

	CP	MIP
n	`ilogcp`	`cplex`
5	0.02	0.02
10	0.08	0.03
15	0.14	0.08
20	0.14	0.12
25	0.14	0.06
30	0.16	0.03
35	0.28	0.25
40	0.20	0.69
45	0.23	0.12
50	0.34	0.95

Logic Example 1

Solve Times *(cont'd)*

	CP	MIP
n	`ilogcp`	`cplex`
50	0.27	0.94
100	0.39	1.36
150	0.61	5.73
200	1.20	125.86
250	1.11	441.39
300	1.62	1470.29
350	1.56	
400	2.18	
450	2.70	
500	4.15	

Logic Example 2

Multicommodity transportation

- Inventory of each product at each origin
- Demand for each product at each destination
- Limited shipment capacity of each origin-destination link

Minimum-shipment constraints

- From each origin to each destination,
- ***either*** ship nothing
- ***or*** ship at least `minload` units

Logic Example 2

Mixed-Integer Linear 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 cost {ORIG,DEST,PROD} >= 0;   # shipment cost

param minload >= 0;                 # minimum shipment size
```

Logic Example 2

Mixed-Integer Linear Formulation

Symbolic model (variables and objective)

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

minimize Total_Cost:
   sum {i in ORIG, j in DEST, p in PROD} cost[i,j,p] * Trans[i,j,p];
```

Logic Example 2

Mixed-Integer Linear 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];
```

Logic Example 2

Mixed-Integer Linear Formulation

Explicit data independent of symbolic model

```
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 ;
```

Logic Example 2

Mixed-Integer Linear Formulation

Explicit data (continued)

```
param cost :=

 [*,*,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 ;
```

Logic Example 2

Mixed-Integer Linear Formulation

Model + data = problem instance to be solved

```
ampl: model multminship.mod;
ampl: data multminship.dat;

ampl: option solver gurobi;

ampl: solve;

Gurobi 5.0.0: optimal solution; objective 201750
141 simplex iterations
13 branch-and-cut node

ampl: display Use;

Use [*,*]

:     DET FRA FRE LAF LAN STL WIN   :=
CLEV   1   1   1   0   1   1   0
GARY   0   0   1   1   0   1   0
PITT   1   1   1   1   0   1   1
;
```

Logic Example 2

Mixed-Integer Linear Formulation

Solver choice independent of model and data

```
ampl: model multminship.mod;
ampl: data multminship.dat;

ampl: option solver cplex;

ampl: solve;

CPLEX 12.4.0.1: optimal integer solution; objective 201750
155 MIP simplex iterations
17 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   1   1   0   1   0
PITT   1   1   1   1   0   1   1
;
```

Logic Example 2

Domain-Based Formulation

Model

```
var Trans {ORIG,DEST,PROD} >= 0;
var SumTrans {i in ORIG, j in DEST}
   in {0} union interval[minload,limit[i,j]];

minimize Total_Cost:
   sum {i in ORIG, j in DEST, p in PROD} cost[i,j,p] * Trans[i,j,p];

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}:
   SumTrans[i,j] = sum {p in PROD} Trans[i,j,p];
```

Logic Example 2

Domain-Based Formulation

Solution

```
ampl: model multminshipA.mod;
ampl: data multminship.dat;

ampl: option solver gurobi;

ampl: solve;

Gurobi 5.0.0: optimal solution; objective 201750
154 simplex iterations
14 branch-and-cut node

display SumTrans;

SumTrans [*,*]

:       DET   FRA   FRE   LAF   LAN   STL   WIN   :=
CLEV    625   425   425     0   500   625     0
GARY      0     0   375   425     0   600     0
PITT    525   475   375   575     0   575   375
;
```

Logic Example 2

Domain-Based Formulation

Solution

```
ampl: model multminshipA.mod;
ampl: data multminship.dat;

ampl: option solver cplex;

ampl: solve;

CPLEX 12.4.0.1: optimal integer solution; objective 201750
155 MIP simplex iterations
17 branch-and-bound nodes

display SumTrans;

SumTrans [*,*]

:      DET   FRA   FRE   LAF   LAN   STL   WIN   :=
CLEV   625   425   425     0   500   625     0
GARY     0     0   375   400     0   625     0
PITT   525   475   375   600     0   550   375
;
```

Logic Example 2

Implication-Based Formulation

Model

```
var Trans {ORIG,DEST,PROD} >= 0;
var Use {ORIG, DEST} binary;

minimize Total_Cost:
   sum {i in ORIG, j in DEST, p in PROD} cost[i,j,p] * Trans[i,j,p];

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}:
   Use[i,j] = 0 ==> sum {p in PROD} Trans[i,j,p] = 0 else
         minload <= sum {p in PROD} Trans[i,j,p] <= limit[i,j];
```

Logic Example 2

Implication-Based Formulation

Solution

```
ampl: model multminshipB.mod;
ampl: data multminship.dat;

ampl: option solver gurobi;

ampl: solve;

Gurobi 5.0.0: Sorry, gurobi cannot handle logical constraints.

exit code 7
<BREAK>
```

Logic Example 2

Implication-Based Formulation

Solution

```
ampl: model multminshipB.mod;
ampl: data multminship.dat;

ampl: option solver cplex;

ampl: solve;

CPLEX 12.4.0.1: optimal integer solution; objective 201750
159 MIP simplex iterations
17 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   1   1   0   1   0
PITT   1   1   1   1   0   1   1
;
```

Logic Example 2

Disjunctive Formulation

Model

```
var Trans {ORIG,DEST,PROD} >= 0;

minimize Total_Cost:
   sum {i in ORIG, j in DEST, p in PROD} cost[i,j,p] * Trans[i,j,p];

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}:
   forall {p in PROD} Trans[i,j,p] = 0  or
   minload <= sum {p in PROD} Trans[i,j,p] <= limit[i,j];
```

Disjunctive Formulation

Solution

```
ampl: model multminshipC.mod;
ampl: data multminship.dat;

ampl: option solver cplex;

ampl: solve;

CPLEX 12.4.0.1:
logical constraint _slogcon[1] is not an indicator constraint.
```

Logic Example 2

Disjunctive Formulation

Solution

```
ampl: model multminshipC.mod;
ampl: data multminship.dat;

ampl: option solver ilogcp;

ampl: solve;

ilogcp 12.4.0: CP Optimizer doesn't support continuous variables.

exit code 1
<BREAK>
```

Logic Example 2

Disjunctive Formulation

Solution (variables declared integer)

```
ampl: model multminshipC.mod;
ampl: data multminship.dat;

ampl: option solver ilogcp;

ampl: solve;

ilogcp 12.4.0:

exit code 3
<BREAK>
```

. . . takes longer than you want to wait

Logic Example 2

Disjunctive Formulation

Solution by CPLEX

```
ampl: model multminshipC.mod;
ampl: data multminship.dat;

ampl: option solver ilogcp;
ampl: option ilogcp_options 'optimizer=cplex';

ampl: solve;

ilogcp 12.4.0: optimizer=cplex
ilogcp 12.4.0: optimal solution
35 nodes, 381 iterations, objective 201750

ampl: display {i in ORIG, j in DEST} sum {p in PROD} Trans[i,j,p];

sum{p in PROD} Trans[i,j,p] [*,*]

:      DET   FRA   FRE   LAF   LAN   STL   WIN    :=
CLEV   625   425   425     0   500   625     0
GARY     0     0   375   425     0   600     0
PITT   525   475   375   575     0   575   375
;
```

Logic Example 1

CP-Style Location Model *(revisited)*

Solution by CPLEX?

```
ampl: model locCP.mod;
ampl: let n := 8;

ampl: option solver ilogcp;
ampl: option ilogcp_options 'optimizer=cplex';

ampl: solve;

ilogcp 12.4.0: optimizer=cplex

Error: IloAlgorithm cannot extract
   extractables 51, 51, 302, 302, 553 and 553

exit code 1
<BREAK>
```

Supported CP-Style Operators

General forms

- Logical
- Counting
- Global

Examples

- Scheduling
- Assignment
- Matching
- Transportation
- Sequencing

Logical Operators

Unary and binary

- *constraint-expr* `and` *constraint-expr*
- *constraint-expr* `or` *constraint-expr*
- `not` *constraint-expr*

Iterated forms

- `exists` { *indexing* } *constraint-expr*
- `forall` { *indexing* } *constraint-expr*

General Forms

Logical Operators *(cont'd)*

Implication expressions

- `if` *constraint-expr* `then` *expr*
- `if` *constraint-expr* `then` *expr* `else` *expr*

Implication constraints

- *constraint-expr* `==>` *constraint-expr*
- *constraint-expr* `==>` *constraint-expr* `else` *constraint-expr*
- *constraint-expr* `<==` *constraint-expr*
- *constraint-expr* `<==` *constraint-expr* `else` *constraint-expr*
- *constraint-expr* `<==>` *constraint-expr*

Counting Operators

Counting expressions

- `count` { *indexing* } (*constraint*)
- `numberof` *num-expr* `in` (*expr1*, *expr2*, . . .)

Counting constraints

- `atmost` *num-expr* { *indexing* } (*constraint*)
- `atleast` *num-expr* { *indexing* } (*constraint*)
- `exactly` *num-expr* { *indexing* } (*constraint*)

"Global" Operators

All-different constraint

- `alldiff` { *indexing* } *expr*
- `alldiff` (*expr1, expr2, . . .*)

Counting expression

- `numberof` *const-expr* `in` (*expr1, expr2, . . .*)

. . . consolidate all having same expr-list

Form of expr1, expr2, . . . in list

- *expr*
- { *indexing* } *expr*

Examples

Logical Conditions

Every job either precedes or follows every other job

```
subj to NoConflict12
  {i1 in JOBS, i2 in JOBS: ord(i1) < ord(i2)}:

    Start[i2] >= Start[i1] +
      setupTime[i1,i2] - BIG * (1 - Prec[i1,i2]);

subj to NoConflict21
  {i1 in JOBS, i2 in JOBS: ord(i1) < ord(i2)}:

    Start[i2] >= Start[i1] +
      setupTime[i2,i1] - BIG * Prec[i1,i2];
```

```
subj to NoConflict
  {i1 in JOBS, i2 in JOBS: ord(i1) < ord(i2)}:

   Start[i2] >= Start[i1] + setupTime[i1,i2] or
   Start[i1] >= Start[i2] + setupTime[i2,i1];
```

Examples

Logical Conditions *(cont'd)*

No one should feel isolated within an assigned group

```
subj to NoIso0 {(i1,i2) in TYPE, j in GROUP}:
  Assign[i1,i2,j] <= upperbnd[i1,i2,j] * Any[i1,i2,j];

subj to NoIso1a {(i1,i2) in TYPE, j in GROUP}:
  Assign[i1,i2,j] >= Any[i1,i2,j];

subj to NoIso1b {(i1,i2) in TYPE, j in GROUP}:
  Assign[i1,i2,j] +
    sum {ii1 in ADJ[i1]: (ii1,i2) in TYPE} Assign[ii1,i2,j]
      >= 2 * Any[i1,i2,j];
```

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

Examples

Cardinality Restrictions

A warehouse may not serve too many different customers

```
var Serve {WHSE,CUST} binary;

subj to UDef {i in WHSE, j in CUST, p in PROD}:
   sum {p in PROD} Trans[i,j,p] <= limit[i,j] * Serve[i,j];

subj to MaxServe {i in WHSE}: sum {j in CUST} Serve[i,j] <= mxsrv;
```

```
subj to MaxServe {i in WHSE}:
   count {j in CUST} (sum {p in PRD} Trans[i,j,p] > 0) <= mxsrv;
```

```
subj to MaxServe {i in WHSE}:
   atmost mxsrv {j in CUST} (sum {p in PRD} Trans[i,j,p] > 0);
```

Examples

Matching

Assign each job to a different machine

```
var Assign {JOBS,MACHINES} binary;

subj to OneJobPerMachine {k in MACHINES}:
   sum {j in JOBS} Assign[j,k] = 1;
```

```
var MachineforJob {JOBS} in MACHINES;

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

Examples

Assignment

Assign a limited number of jobs to each machine

```
var Assign {JOBS,MACHINES} binary;

subj to CapacityOfMachine {k in MACHINES}:
   sum {j in JOBS} Assign[j,k] <= cap[k];
```

```
var MachineforJob {JOBS} in MACHINES;

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

What's missing (currently)?

Domain-based formulations

- Sending directly to solver
- Object-valued variables

CP-style formulations

- Variables in subscripts

Support for additional solvers

What's Missing?

Domain-Based Formulation

Variable `in` arbitrary set of numbers

```
var SumTrans {i in ORIG, j in DEST}
   in {0} union interval[minload,limit[i,j]];
```

Transform in AMPL (current)

- Convert to linear MIP
- Add "SOS" markers used by some MIP solvers

Send directly to solver (prospective)

- Write representation of domain in problem file
- Let interface decide how to convert for solver
  - Semi-continuous variables (CPLEX)
  - Arbitrary domains (CP)

What's Missing?

Variables in Subscripts

Return to spatial location problem

- Build n observation posts
- Locate at points on an n-by-n grid
- *Minimize total construction cost*

Non-interference constraints

- No post blocks any other's view
- along any row, column or diagonal of the grid

Mixed-Integer Linear Formulation

Sum of n^2 costs times binary variables

```
param n integer > 0;
param cost {1..n,1..n};

var Build {1..n,1..n} binary;

minimize TotalCost:
   sum {i in 1..n, j in 1..n} cost[i,j] * Build[i,j];

subj to row_conflicts {i in 1..n}:
   sum {j in 1..n} Build[i,j] = 1;

subj to col_conflicts {j in 1..n}:
   sum {i in 1..n} Build[i,j] = 1;

subj to diag1_conflicts {k in 3..2*n-1}:
   sum {i in max(1,k-n)..min(n,k-1)} Build[i,k-i] <= 1;

subj to diag2_conflicts {k in -(n-2)..(n-2)}:
   sum {i in max(k,0)+1..min(k,0)+n} Build[i,i-k] <= 1;
```

CP-Style Formulation

Sum of n^2 conditional terms

```
param n integer > 0;
param cost {1..n,1..n};

var Col {1..n} integer >= 1 <= n;

minimize TotalCost:
   sum {i in 1..n, j in 1..n} if Col[i] = j then cost[i,j];

subj to col_conflicts:   alldiff {i in 1..n} Col[i];

subj to diag1_conflicts: alldiff {i in 1..n} (Col[i] + i);

subj to diag2_conflicts: alldiff {i in 1..n} (Col[i] - i);
```

CP-Style Formulation

Sum of n costs

```
param n integer > 0;
param cost {1..n,1..n};
var Col {1..n} integer >= 1 <= n;

minimize TotalCost:
   sum {i in 1..n} cost[i,Col[i]];

subj to col_conflicts:   alldiff {i in 1..n} Col[i];
subj to diag1_conflicts: alldiff {i in 1..n} (Col[i] + i);
subj to diag2_conflicts: alldiff {i in 1..n} (Col[i] - i);
```

. . . "variable in subscript"

Domain-Based Formulation

*Variable **in** arbitrary set*

```
var Serve {CLI} in LOC;
var Open {LOC} binary;

minimize TotalCost:
   sum {i in 1..mCLI} srvCost[i,Serve[i]] +
   bdgCost * sum {j in 1..nLOC} Open[j];

subject to OpenDefn {i in 1..mCLI}: Open[Serve[i]] = 1;
```

Additional Solvers

ILOG CP is a good start

- Allows experimentation with
  a mature general-purpose CP solver
- Permits some comparisons to a MIP solver
- *. . . but IBM has a modeling language tailored to CP*

AMPL's strength is solver-independence

- Other solvers can benefit from this technology
- Various possibilities under consideration
  - Other constraint programming solvers
  - LINDO Global
  - SCIP
  - LocalSolver
  - . . .

How to Get the CP Interface

If you have an IBM CPLEX license from us

- Log in to your account at the AMPL download site
- Click on an ilogcp link to get a bundle of the needed files

If you have a AMPL academic license from us

- Join the IBM Academic Initiative
- Send info@ampl.com a request to add
  free 1-year licenses for the CPLEX and ILOG CP solvers

Or request a free 30-day AMPL trial

- Write in ILOG CP as one of the solvers to include

SolverStudio

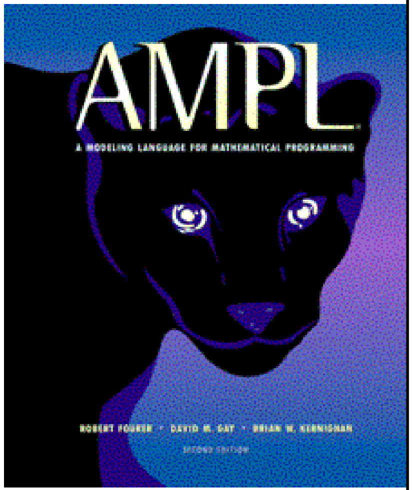
AMPL Modeling in Excel

SolverStudio is an integrated Excel add-in that makes it easy to develop and deliver AMPL optimization models using the familiar Excel environment. SolverStudio adds a text editor to Excel that allows an optimization model to be created using AMPL and then embedded and saved within a spreadsheet. SolverStudio also provides an integrated data editor that allows model data (AMPL parameters and sets) to be stored and edited on the spreadsheet. SolverStudio's *Solve* button runs the AMPL model while seamlessly managing data transfers with the spreadsheet. AMPL models can be solved locally, or in the cloud using NEOS. As well as working with AMPL, SolverStudio also supports the GAMS, PuLP and Gurobi Python modelling languages, amongst others.

SolverStudio is being developed and supported by Andrew Mason at the Department of Engineering Science, University of Auckland, New Zealand.

http://solverstudio.org

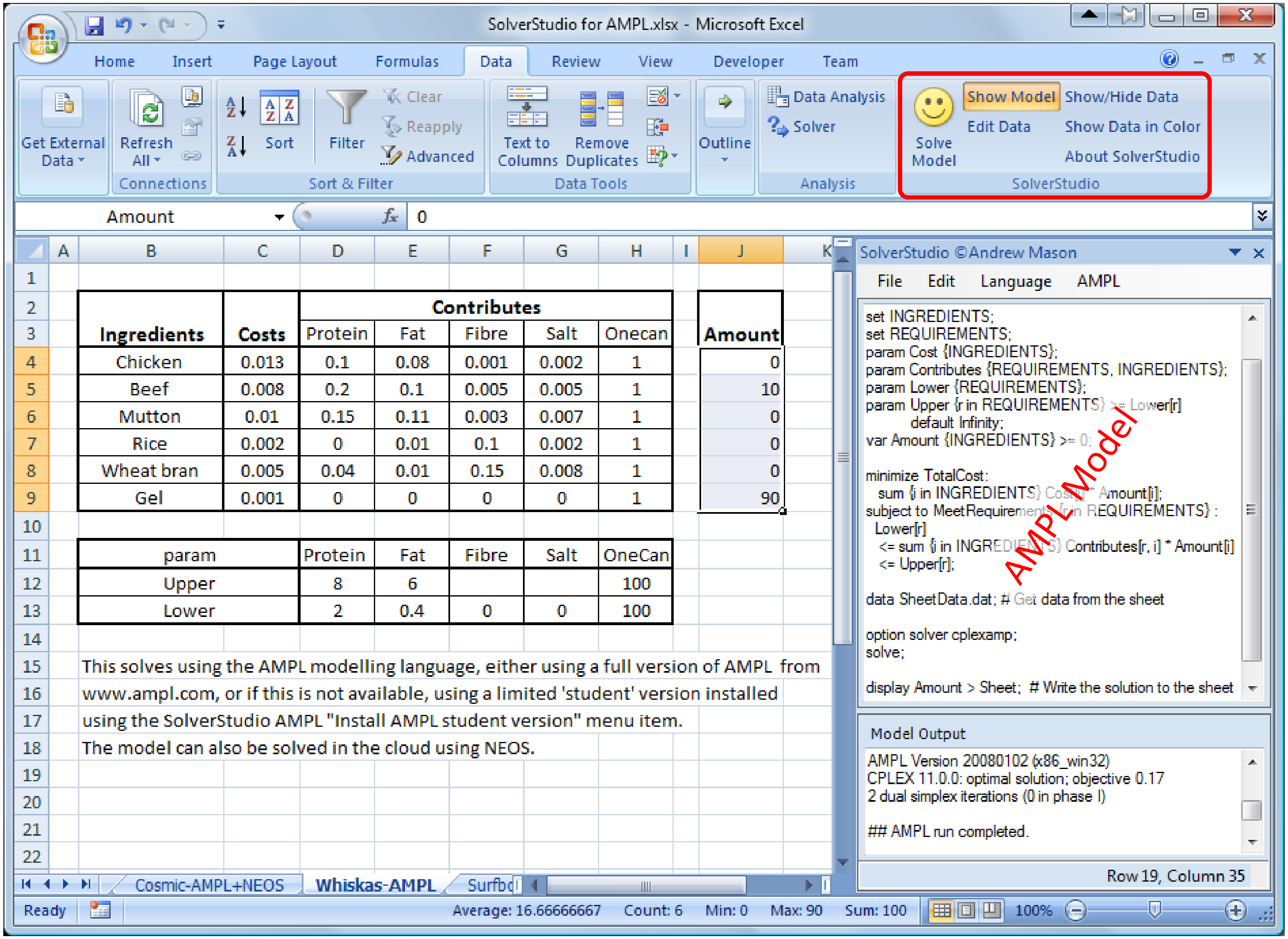
NEOS SERVER



Microsoft Office Excel

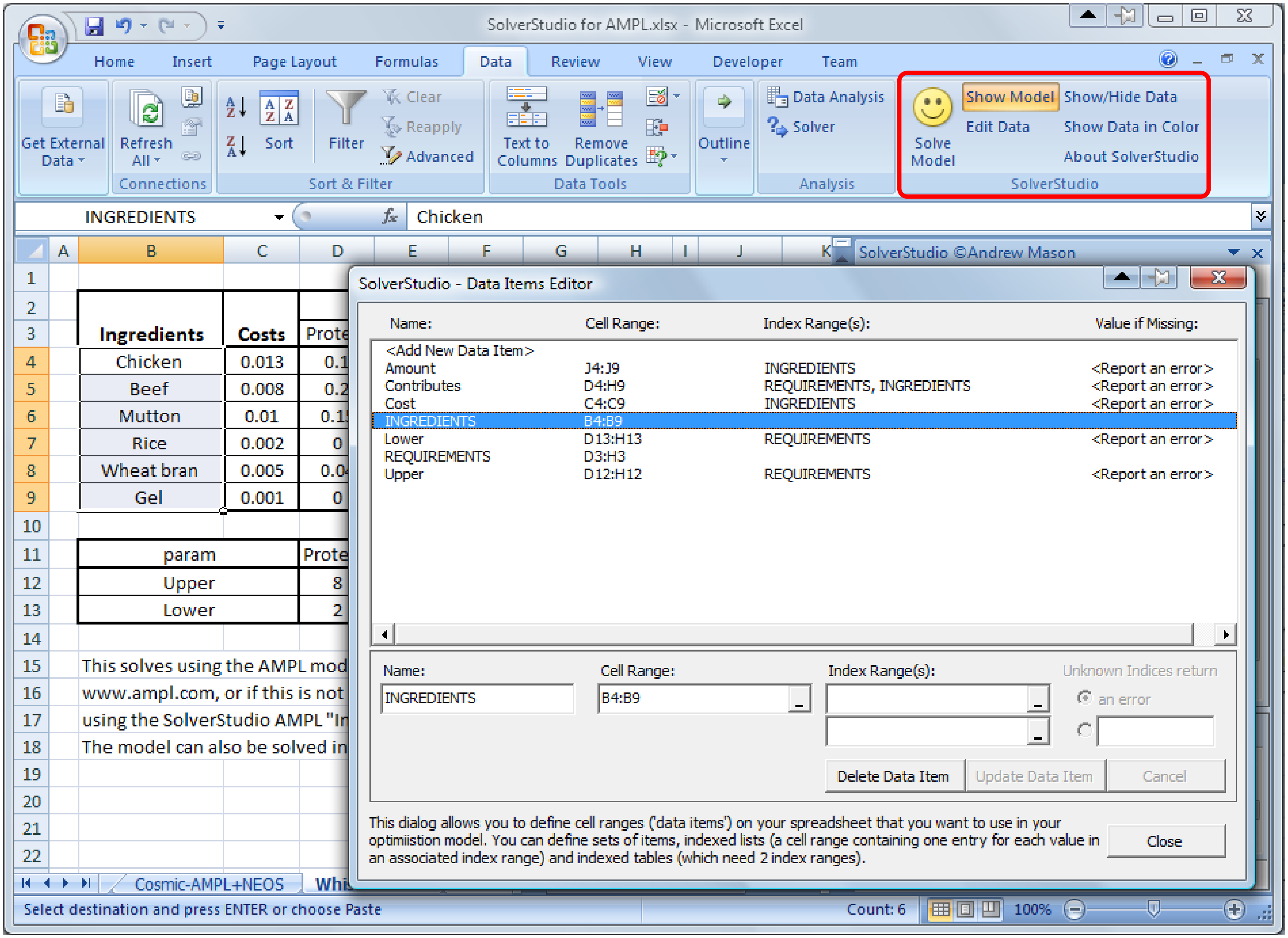
Solver Studio for Excel

SolverStudio



- Build & solve AMPL models using Excel
- Solve AMPL models in the cloud using NEOS.
- Free download: http://solverstudio.org

SolverStudio



- Easily define params & sets on spreadsheet
- Automatic data exchange with model
- Free download from http://solverstudio.org