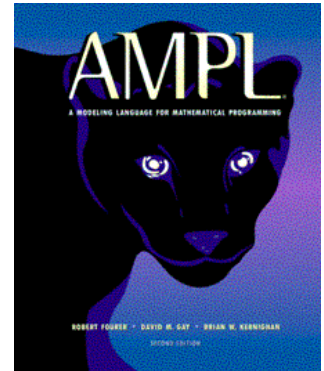


AMPL

A Modeling Language for Mathematical Programming

www.ampl.com



Robert Fourer

Department of Industrial Engineering & Management Sciences
Northwestern University

David M. Gay

AMPL Optimization LLC

Workshop on Modeling Languages in Mathematical Optimization

Gesellschaft für Operations Research e.V., 23-25 April 2003

The McDonald's Diet Problem

Foods:

QP	Quarter Pounder
FR	Fries, small
MD	McLean Deluxe
SM	Sausage McMuffin
BM	Big Mac
1M	1% Lowfat Milk
FF	Filet-O-Fish
OJ	Orange Juice
MC	McGrilled Chicken

Nutrients:

Prot	Protein
Iron	Iron
VitA	Vitamin A
Cals	Calories
VitC	Vitamin C
Carb	Carbohydrates
Calc	Calcium

McDonald's Diet Problem Data

	QP	MD	BM	FF	MC	FR	SM	1M	OJ	
Cost	1.8	2.2	1.8	1.4	2.3	0.8	1.3	0.6	0.7	Need:
Protein	28	24	25	14	31	3	15	9	1	55
Vitamin A	15	15	6	2	8	0	4	10	2	100
Vitamin C	6	10	2	0	15	15	0	4	120	100
Calcium	30	20	25	15	15	0	20	30	2	100
Iron	20	20	20	10	8	2	15	0	2	100
Calories	510	370	500	370	400	220	345	110	80	2000
Carbo	34	35	42	38	42	26	27	12	20	350

Formulation: Too General

Minimize cx

Subject to $Ax = b$

$x \geq 0$

Formulation: Too Specific

$$\begin{array}{l}
 \textit{Minimize} \quad 1.84 x_{QP} + 2.19 x_{MD} + 1.84 x_{BM} + 1.44 x_{FF} + 2.29 x_{MC} + 0.77 x_{FR} + 1.29 x_{SM} + 0.60 x_{IM} + 0.72 x_{OJ} \\
 \textit{Subject to} \quad 28 x_{QP} + 24 x_{MD} + 25 x_{BM} + 14 x_{FF} + 31 x_{MC} + 3 x_{FR} + 15 x_{SM} + 9 x_{IM} + 1 x_{OJ} \geq 55 \\
 \quad \quad \quad 15 x_{QP} + 15 x_{MD} + 6 x_{BM} + 2 x_{FF} + 8 x_{MC} + 0 x_{FR} + 4 x_{SM} + 10 x_{IM} + 2 x_{OJ} \geq 100 \\
 \quad \quad \quad 6 x_{QP} + 10 x_{MD} + 2 x_{BM} + 0 x_{FF} + 15 x_{MC} + 15 x_{FR} + 0 x_{SM} + 4 x_{IM} + 120 x_{OJ} \geq 100 \\
 \quad \quad \quad 30 x_{QP} + 20 x_{MD} + 25 x_{BM} + 15 x_{FF} + 15 x_{MC} + 0 x_{FR} + 20 x_{SM} + 30 x_{IM} + 2 x_{OJ} \geq 100 \\
 \quad \quad \quad 20 x_{QP} + 20 x_{MD} + 20 x_{BM} + 10 x_{FF} + 8 x_{MC} + 2 x_{FR} + 15 x_{SM} + 0 x_{IM} + 2 x_{OJ} \geq 100 \\
 \quad \quad \quad 510 x_{QP} + 370 x_{MD} + 500 x_{BM} + 370 x_{FF} + 400 x_{MC} + 220 x_{FR} + 345 x_{SM} + 110 x_{IM} + 80 x_{OJ} \geq 2000 \\
 \quad \quad \quad 34 x_{QP} + 35 x_{MD} + 42 x_{BM} + 38 x_{FF} + 42 x_{MC} + 26 x_{FR} + 27 x_{SM} + 12 x_{IM} + 20 x_{OJ} \geq 350
 \end{array}$$

Algebraic Model

Given \mathcal{F} , a set of foods
 \mathcal{N} , a set of nutrients

and $a_{ij} \geq 0$, the units of nutrient i in one serving of food j ,
for each $i \in \mathcal{N}$ and $j \in \mathcal{F}$

$b_i > 0$, the units of nutrient i required, for each $i \in \mathcal{N}$

$c_j > 0$, the cost per serving of food j , for each $j \in \mathcal{F}$

Define $x_j \geq 0$, the number of servings of food j to be purchased, for each $j \in \mathcal{F}$

Minimize $\sum_{j \in \mathcal{F}} c_j x_j$

Subject to $\sum_{j \in \mathcal{F}} a_{ij} x_j \geq b_i$, for each $i \in \mathcal{N}$

Algebraic Model in AMPL

```
set NUTR;    # nutrients
set FOOD;    # foods

param amt {NUTR,FOOD} >= 0;    # amount of nutrient in each food
param nutrLow {NUTR} >= 0;    # lower bound on nutrients in diet
param cost {FOOD} >= 0;    # cost of foods

var Buy {FOOD} >= 0 integer;    # amounts of foods to be purchased

minimize TotalCost: sum {j in FOOD} cost[j] * Buy[j];

subject to Need {i in NUTR}:
    sum {j in FOOD} amt[i,j] * Buy[j] >= nutrLow[i];
```

Data for the AMPL Model

```

param: FOOD:          cost :=
  "Quarter Pounder"  1.84    "Fries, small"      .77
  "McLean Deluxe"   2.19    "Sausage McMuffin"  1.29
  "Big Mac"         1.84    "1% Lowfat Milk"   .60
  "Filet-O-Fish"    1.44    "Orange Juice"     .72
  "McGrilled Chicken" 2.29 ;

param: NUTR: nutrLow :=
  Prot  55  Vita 100  VitC  100
  Calc 100  Iron 100  Cals 2000  Carb 350 ;

param amt (tr):          Cals  Carb  Prot  Vita  VitC  Calc  Iron
  :=
  "Quarter Pounder"     510   34   28   15   6   30   20
  "McLean Deluxe"       370   35   24   15  10   20   20
  "Big Mac"             500   42   25   6    2   25   20
  "Filet-O-Fish"        370   38   14   2    0   15   10
  "McGrilled Chicken"   400   42   31   8   15   15    8
  "Fries, small"        220   26   3    0   15    0    2
  "Sausage McMuffin"    345   27   15   4    0   20   15
  "1% Lowfat Milk"      110   12   9   10   4   30    0
  "Orange Juice"        80    20   1    2  120    2    2 ;

```


Continuous-Variable Solution

```
ampl: model mcdiet1.mod;
```

```
ampl: data mcdiet1.dat;
```

```
ampl: solve;
```

```
MINOS 5.5: ignoring integrality of 9 variables
```

```
MINOS 5.5: optimal solution found.
```

```
7 iterations, objective 14.8557377
```

```
ampl: display Buy;
```

```
Buy [*] :=
```

```
  1% Lowfat Milk      3.42213
```

```
    Big Mac          0
```

```
    Filet-O-Fish     0
```

```
    Fries, small     6.14754
```

```
McGrilled Chicken    0
```

```
  McLean Deluxe      0
```

```
    Orange Juice     0
```

```
    Quarter Pounder  4.38525
```

```
    Sausage McMuffin  0
```

Integer-Variable Solution

```
ampl: option solver cplex;
```

```
ampl: solve;
```

```
CPLEX 7.0.0: optimal integer solution; objective 15.05
```

```
41 MIP simplex iterations
```

```
23 branch-and-bound nodes
```

```
ampl: display Buy;
```

```
Buy [*] :=
```

1% Lowfat Milk	4
Big Mac	0
Filet-O-Fish	1
Fries, small	5
McGrilled Chicken	0
McLean Deluxe	0
Orange Juice	0
Quarter Pounder	4
Sausage McMuffin	0

Same for 63 Foods, 12 Nutrients

```
ampl: reset data;
ampl: data mcdiet2.dat;

ampl: option solver minos;
ampl: solve;
MINOS 5.5: ignoring integrality of 63 variables
MINOS 5.5: optimal solution found.
16 iterations, objective -1.786806582e-14

ampl: option omit_zero_rows 1;
ampl: display Buy;
Buy [*] :=
           Bacon Bits      55
           Barbeque Sauce  50
           Hot Mustard Sauce 50
```

Essential Modeling Language Features

Sets and indexing

Simple sets

Compound sets

Computed sets

Objectives and constraints

Linear, piecewise-linear

Nonlinear

Integer, network

... and many more features

Express problems the various ways that people do

Support varied solvers

Example: 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  
  
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
```

Airline Fleet Assignment (*cont'd*)

```
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 & departure times at that city,  
  # with arrival time adjusted for the turn requirement  
  
param leg_cost {FLEET_LEGS} >= 0;  
param fleet_size {FLEETS} >= 0;
```

Airline Fleet Assignment (*cont'd*)

```
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
```

Airline Fleet Assignment (*cont'd*)

```
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];

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


Extended and Forthcoming AMPL Features

Programming iterative schemes

Loops over sets, **if-then-else** tests

Switching between subproblems

Debugging

Recognizing other types of models

Complementarity problems

General combinatorial problems (*to come*)

Stochastic programs (*to come*)

Communicating with other systems

Relational database access

Internet optimization services

Suffixes for solver-specific

directives, results & diagnostic information

Iterative Schemes

Flow of control

Looping

If-then-else

Named subproblems

Defining subproblems

Switching subproblems

Debugging

Expanding constraints

Single-stepping

Flow of Control

```
model diet.mod;
data diet2a.dat;

set NALOG default {};
param NAobj {NALOG};
param NAdual {NALOG};

for {theta in 52000 .. 70000 by 1000} {
    let n_max["NA"] := theta;
    solve;
    let NALOG := NALOG union {theta};
    let NAobj[theta] := total_cost;
    let NAdual[theta] := diet["NA"].dual;
    if diet["NA"].dual > -.000001 then break;
}
```

Iterative Schemes

Flow of Control: *Sample Output*

```
AMPL: commands diet.run;
AMPL: display NAobj, NAdual;
:      NAobj      NAdual      :=
52000  113.428    -0.0021977
53000  111.23     -0.0021977
54000  109.42     -0.00178981
55000  107.63     -0.00178981
56000  105.84     -0.00178981
57000  104.05     -0.00178981
58000  102.26     -0.00178981
59000  101.082    -0.000155229
60000  101.013    0                ;
```

Iterative Schemes

Subproblems

Cutting-stock optimization with given patterns

```
param roll_width > 0;
set WIDTHS;
param orders {WIDTHS} > 0;
param nPAT integer >= 0;
set PATTERNS := 1..nPAT;
param nbr {WIDTHS,PATTERNS} integer >= 0;
var Cut {PATTERNS} integer >= 0;
minimize Number: sum {j in PATTERNS} Cut[j];
subj to Fill {i in WIDTHS}:
    sum {j in PATTERNS} nbr[i,j] * Cut[j] >= orders[i];
```

New pattern generation

```
param price {WIDTHS};
var Use {WIDTHS} integer >= 0;
minimize Reduced_Cost:
    1 - sum {i in WIDTHS} price[i] * Use[i];
subj to Width_Limit:
    sum {i in WIDTHS} i * Use[i] <= roll_width;
```

Naming Defined Subproblems

*AMPL “script” to set up for
Gilmore-Gomory column generation method*

```
### DEFINE CUTTING PROBLEM
problem Cutting_Opt: Cut, Number, Fill;
option relax_integrality 1;

### DEFINE PATTERN-GENERATING PROBLEM
problem Pattern_Gen: Use, Reduced_Cost, Width_Limit;
option relax_integrality 0;

### SET UP INITIAL CUTTING PATTERNS
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;
};
```

... each problem has its own option environment

Iterative Schemes

Switching Subproblems

*AMPL “script” for main loop
of Gilmore-Gomory column generation*

```
repeat {  
  solve Cutting_Opt;  
  display Cut;  
  let {i in WIDTHS} price[i] := Fill[i].dual;  
  solve Pattern_Gen;  
  display price,Use;  
  if Reduced_Cost >= -0.00001 then break;  
  else {  
    let nPAT := nPAT + 1;  
    let {i in WIDTHS} nbr[i,nPAT] := Use[i];  
  };  
};  
option Cutting_Opt.relax_integrality 0;  
solve Cutting_Opt;
```

Iterative Schemes

Debugging

Single-stepping through the cutting-stock script

```
AMPL: model cut.mod; data cut.dat;
AMPL: option single_step 1;
AMPL: commands cut.run;
cut.run:10(172) for ...
<2>AMPL: next
cut.run:16(318) option ...
<2>AMPL: display nbr;
nbr [*,*]
:    1    2    3    4    5 :=
20   5    0    0    0    0
45   0    2    0    0    0
50   0    0    2    0    0
55   0    0    0    2    0
75   0    0    0    0    1 ;
<2>AMPL: step
cut.run:19(365) repeat ...
<2>AMPL: step
cut.run:23(454) solve ...
```


Iterative Schemes

Debugging *(cont'd)*

Expanding the cutting-stock constraints

```
ampl: display nbr;

:      1    2    3    4    5    6    7    8 :=
20     5    0    0    0    0    1    1    3
45     0    2    0    0    0    2    0    0
50     0    0    2    0    0    0    0    1
55     0    0    0    2    0    0    0    0
75     0    0    0    0    1    0    1    0 ;

ampl: expand Fill;
s.t. Fill[20]:
      5*Cut[1] + Cut[6] + Cut[7] + 3*Cut[8] >= 48;
s.t. Fill[45]:
      2*Cut[2] + 2*Cut[6] >= 35;
s.t. Fill[50]:
      2*Cut[3] + Cut[8] >= 24;
s.t. Fill[55]:
      2*Cut[4] >= 10;
s.t. Fill[75]:
      Cut[5] + Cut[7] >= 8;
```

... can also view after reduction by “presolve” routines

Complementarity Problems

Definition

Collections of complementarity conditions:

- Two inequalities must hold,
at least one of them with equality

Applications

Equilibrium problems in economics and engineering

Optimality conditions for nonlinear programs,
bi-level linear programs, bimatrix games, . . .

Classical Linear Complementarity

Economic equilibrium

```
set PROD;    # products
set ACT;     # activities

param cost {ACT} > 0;    # cost per unit of each activity
param demand {PROD} >= 0; # units of demand for each product
param io {PROD,ACT} >= 0; # units of each product from
                          # 1 unit of each activity

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] >= demand[i];

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

*... complementary slackness conditions
for an equivalent linear program*

Complementarity

Mixed Linear Complementarity

Economic equilibrium with bounded variables

```
set PROD;    # products
set ACT;     # activities

param cost {ACT} > 0;    # cost per unit
param demand {PROD} >= 0;    # units of demand

param io {PROD,ACT} >= 0;    # units of product per unit of activity

param level_min {ACT} > 0;    # min allowed level for each activity
param level_max {ACT} > 0;    # max allowed level for each activity

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] >= demand[i];

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

*... complementarity conditions
for optimality of an equivalent bounded linear program*

Nonlinear Complementarity

Economic equilibrium with price-dependent demands

```
set PROD;    # products
set ACT;     # activities

param cost {ACT} > 0;    # cost per unit
param demand {PROD} >= 0;    # units of demand

param io {PROD,ACT} >= 0;    # units of product per unit of activity
param demzero {PROD} > 0;    # intercept and slope of the demand
param demrate {PROD} >= 0;    # as a function of price

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];
```

... not equivalent to a linear program

Complementarity

Operands to complements: always 2 inequalities

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 $expr = 0$

One equality

equality complements expr

expr complements equality

The equality must hold (*included for completeness*)

Complementarity

Solvers

“Square” systems

of variables =
of complementarity constraints +
of equality constraints

Transformation to a simpler canonical form required

MPECs

Mathematical programs with equilibrium constraints

No restriction on numbers of variables & constraints

Objective functions permitted

... solvers beginning to emerge

General Combinatorial Problems

Formulations

More natural for modelers than integer programs

Independent of solvers

Compatible with existing modeling languages

Solution methods

Theoretically optimal

Based on tree search (like branch & bound)

Sensitive to details of search strategy

Combinatorial

Example: Job Sequencing with Setups

Given

A set of jobs, with
production times, due times and earliness penalties

One machine that processes one job at a time

Setup costs and times between jobs

Precedence relations between certain jobs

Choose

A sequence for the jobs

Minimizing

Setup costs plus earliness penalties

C. Jordan & A. Drexl, A Comparison of Constraint and Mixed Integer Programming Solvers for Batch Sequencing with Sequence Dependent Setups.
ORSA Journal on Computing 7 (1995) 160–165.

Example: Variables and Costs

Either way

`ComplTime[j]` is the completion time of job `j`

Earliness penalty is the sum over jobs `j` of

$$\text{duePen}[j] * (\text{dueTime}[j] - \text{ComplTime}[j])$$

Integer programming formulation

`Seq[i,j]` = 1 iff `i` immediately precedes `j`

Setup cost is the sum over job pairs `(i,j)` of

$$\text{setupCost}[i,j] * \text{Seq}[i,j]$$

More natural formulation

`JobForSlot[k]` is the job in the `k`th slot in sequence

Setup cost is the sum over slots `k` of

$$\text{setupCost}[\text{JobForSlot}[k], \text{JobForSlot}[k+1]]$$

Example: Production Constraints

Integer programming formulation

For each job i , $\text{CompTime}[i] \leq \text{dueTime}[i]$

For each job pair (i, j) ,

$$\begin{aligned} \text{CompTime}[i] + \text{setupTime}[i, j] + \text{procTime}[j] &\leq \\ \text{CompTime}[j] + \text{BIG} * (1 - \text{Seq}[i, j]) & \end{aligned}$$

More natural formulation

For each slot k ,

$$\begin{aligned} \text{CompTime}[\text{JobForSlot}[k]] &= \min (\\ &\text{dueTime}[\text{JobForSlot}[k]], \\ &\text{CompTime}[\text{JobForSlot}[k+1]] \\ &\quad - \text{procTime}[\text{JobForSlot}[k+1]] \\ &\quad - \text{setupTime}[\text{JobForSlot}[k], \text{JobForSlot}[k+1]]) \end{aligned}$$

Example: Sequencing Constraints

Integer programming formulation

For each job i ,

$$\text{sum } \{j \text{ in JOBS}\} \text{Seq}[i,j] = 1$$

For each job i ,

$$\text{sum } \{j \text{ in JOBS}\} \text{Seq}[j,i] = 1$$

More natural formulation

`all_different` { k in SLOTS} JobForSlot[k]

Representing “Range” Constraints

- *General format*

$$\text{lower-bound} \leq \text{linear-expr} + \text{nonlinear-expr} \leq \text{upper-bound}$$

Arrays of *lower-bound* and *upper-bound* values

Coefficient lists for *linear-expr*

Expression tree for *nonlinear-expr*

- *Expression tree nodes*

Variables, constants

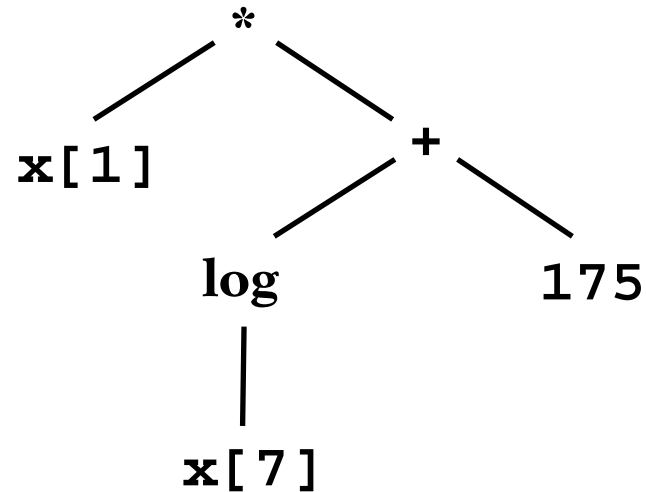
Binary, unary operators

Iterated summation, min, max

Piecewise-linear terms

If-then-else terms

... *single array of variables*



“Walking the Tree”

Example: AMPL interface to ILOG Concert

Definition of variables

```
IloNumVarArray Var(env, n_var);  
for (j = 0; j < n_var - n_var_int; j++)  
    Var[j] = IloNumVar(env, loVarBnd[j], upVarBnd[j], ILOFLOAT);  
for (j = n_var - n_var_int; j < n_var; j++)  
    Var[j] = IloNumVar(env, loVarBnd[j], upVarBnd[j], ILOINT);
```

Tree Walk (*cont'd*)

Top-level processing of constraints

```
IloRangeArray Con(env, n_con);  
for (i = 0; i < n_con; i++) {  
    IloExpr conExpr(env);  
    if (i < nlc)  
        conExpr += build_expr (con_de[i].e);  
    for (cg = Cgrad[i]; cg; cg = cg->next)  
        conExpr += (cg -> coef) * Var[cg -> varno];  
    Con[i] = (loConBnd[i] <= conExpr <= upConBnd[i]);  
}
```

Tree Walk (*cont'd*)

Tree-walk function for expressions

```
IloExpr build_expr (expr *e)
{
    expr **ep;
    IloInt opnum;
    IloExpr partSum;

    opnum = (int) e->op;
    switch(opnum) {
        case PLUS_opno: ...
        case MINUS_opno: ...
        .....
    }
}
```


Tree Walk (cont'd)

Tree-walk cases for expression nodes

```
switch(opnum) {  
  case PLUS_opno:  
    return build_expr (e->L.e) + build_expr (e->R.e);  
  case SUMLIST_opno:  
    partSum = IloExpr(env);  
    for (ep = e->L.ep; ep < e->R.ep; *ep++)  
      partSum += build_expr (*ep);  
    return partSum;  
  case LOG_opno:  
    return IloLog (build_expr (e->L.e));  
  case CONST_opno:  
    return IloExpr (env, ((expr_n*)e)->v);  
  case VAR_opno:  
    return Var[e->a];  
  .....  
}
```

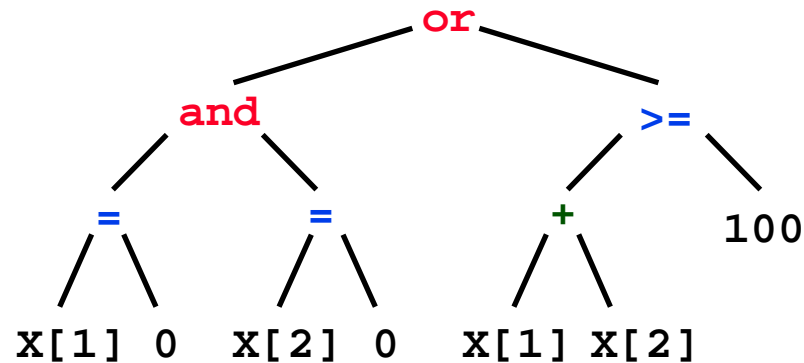
Logical Constraints

Simple forms

constraint **and** constraint

constraint **or** constraint

not constraint



$$(x[1] = 0 \text{ and } x[2] = 0) \text{ or } x[1] + x[2] \geq 100$$

Representation

Expression tree for entire constraint

Constraint nodes whose children are constraint nodes

Constraint nodes whose children are expression nodes

Tree Walk (*cont'd*)

Tree-walk function for constraints

```
IloConstraint build_constr (expr *e)
{
    expr **ep;
    IloInt opnum;

    opnum = (int) e->op;
    switch(opnum) {
        .....
    }
}
```

Tree Walk (*cont'd*)

Tree-walk cases for constraint nodes

```
switch(opnum) {  
  case OR_opno:  
    return build_constr (e->L.e) || build_constr (e->R.e);  
  case AND_opno:  
    return build_constr (e->L.e) && build_constr (e->R.e);  
  case GE_opno:  
    return build_expr (e->L.e) >= build_expr (e->R.e);  
  case EQ_opno:  
    return build_expr (e->L.e) == build_expr (e->R.e);  
  .....  
}
```

Further Tree-Walk Cases

Constraint types

Counting expressions and constraints

Structure (global) constraints

Variables in subscripts

Solver inputs

C++ types and operators (ILOG Concert)

Unindexed algebraic input format (BARON)

Codelist of 4-tuples (GlobSol)

Compact, flexible NOP format (GLOPT)

Stochastic Programs

Extensions within AMPL

Allow random distributions for some problem data

Make distributions available to solvers

Extensions using AMPL

Add special expressions and conventions for stages & scenario trees

Compile to standard AMPL

Generate problem descriptions for various solvers

... various schemes being independently developed

Random Entities

Distributions set in the model

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

Distributions assigned as data

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

Parameters or Variables?

Modeled like “random” parameters

Specify distributions in place of fixed data values

Instantiate the same model with different distributions

Processed like “defined” variables

Save a symbolic definition rather than a specific sample

Record in expression tree passed to solver driver

Evaluate (sample) as directed by solver

New Expression Types

Discrete distributions

`Discrete (1/3, 20, 1/3, 50, 1/3, 175)`

`Discrete ({s in SCEN} (prob[s],demand[s]))`

Stochastic objectives

Default: expected value of objective

Explicit: using functions **Expected_Value** and **Variance**

Further Concerns

Modeling

Recourse variables indicated by user-defined **.stage** suffix

Chance constraints defined by

new function **Probability** (*logical-expression*)

Processing

For **Discrete**, **Uniform**, and other (half-) bounded distributions,
AMPL's presolve phase may eliminate constraints.

Jacobian entries indicate

which constraints involve which random entities

Relational Database Access

Principles

Model stays strictly independent of data

New **table** statement links model to relational tables

New **read table**, **write table** statements control data transfer

Open interface supports adding new links

Extensions

Reading and writing the same table

Indexed collections of tables or columns

Reading from SQL queries

Example: Diet Model

```
set FOOD;  
param cost {FOOD} > 0;  
param f_min {FOOD} >= 0;  
param f_max {j in FOOD} >= f_min[j];  
  
set NUTR;  
param n_min {NUTR} >= 0;  
param n_max {i in NUTR} >= n_min[i];  
  
param amt {NUTR,FOOD} >= 0;  
  
var Buy {j in FOOD} >= f_min[j], <= f_max[j];  
  
minimize Total_Cost:  
    sum {j in FOOD} cost[j] * Buy[j];  
  
subject to Diet {i in NUTR}:  
    n_min[i] <= sum {j in FOOD} amt[i,j] * Buy[j] <= n_max[i];
```

Database

Example: Diet Data (in MS Access)

The screenshot shows the Microsoft Access interface for a database named 'dietIN'. The main window displays three tables: 'Amounts', 'Foods', and 'Nutrs'. The 'Foods' table is currently selected and shown in Datasheet View. The 'Nutrs' table is also shown in Datasheet View. The 'Amounts' table is shown in a separate window, also in Datasheet View.

foods	cost	f_min	f_max
BEEF	3.19	2	10
CHK	2.59	2	10
FISH	2.29	2	10
HAM	2.89	2	10
MCH	1.89	2	10
MTL	1.99	2	10
SPG	1.99	2	10
TUR	2.49	2	10
*			

nutrients	n_min	n_max
A	700	20000
B1	700	20000
B2	700	20000
C	700	20000
CAL	16000	24000
NA	0	50000
*		

foods	nutrs	amt
BEEF	A	60
BEEF	B1	10
BEEF	B2	15
BEEF	C	20
BEEF	NA	938
BEEF	CAL	295
CHK	A	8
CHK	B1	20
CHK	B2	20
CHK	C	0
CHK	NA	2180
CHK	CAL	770
FISH	A	8
FISH	B1	15
FISH	B2	10
FISH	C	10
FISH	NA	945
FISH	CAL	440
HAM	A	40
HAM	B1	35
HAM	B2	10
HAM	C	40
HAM	NA	278
HAM	CAL	430
MCH	A	15
MCH	B1	15
MCH	B2	15
MCH	C	35
MCH	NA	1182
MCH	CAL	315
MTL	A	70
MTL	B1	15
MTL	B2	10
MTL	C	10
MTL	NA	945
MTL	CAL	440
TUR	A	8
TUR	B1	20
TUR	B2	20
TUR	C	0
TUR	NA	2180
TUR	CAL	770
*		

Database

Example: Diet Script (in AMPL)

```
model diet.mod;

table dietFoods IN "ODBC" "diet.mdb" "Foods":
    FOOD <- [foods], cost, f_min, f_max;
table dietNutrs IN "ODBC" "diet.mdb" "Nutrs":
    NUTR <- [nutrients], n_min, n_max;
table dietAmts IN "ODBC" "diet.mdb" "Amounts":
    [nutrs, foods], amt;

read table dietFoods;
read table dietNutrs;
read table dietAmts;

solve;

table dietResults OUT "ODBC" "diet.mdb" "Scen3": [foodlist],
    Buy, Buy.rc ~ BuyRC, {j in FOOD} Buy[j]/f_max[j] ~ BuyFrac;

write table dietResults;
```

Database

Example: Diet Results (in MS Access)

The screenshot displays the Microsoft Access interface. The main window is titled 'diet : Database' and shows a list of tables: Amounts, Foods, Nutrs, and Scen3. Below this, a window titled 'Scen3 : Table' is open, showing a datasheet view of the Scen3 table. The table has four columns: foodlist, Buy, BuyRC, and BuyFrac. The data is as follows:

foodlist	Buy	BuyRC	BuyFrac
BEEF	5.36	0.00	53.6%
CHK	2.00	1.19	20.0%
FISH	2.00	1.14	20.0%
HAM	10.00	-0.30	100.0%
MCH	10.00	-0.55	100.0%
MTL	10.00	-1.33	100.0%
SPG	9.31	0.00	93.1%
TUR	2.00	2.73	20.0%
*			

The status bar at the bottom of the window indicates 'Record: 7 of 8' and 'Datasheet View'.

Internet Optimization Services

Optimization Modeling-Language Servers

Single-language servers

General server projects related to optimization

NEOS Server

Client-Server Alternatives

General-purpose and special-purpose clients

Callable NEOS

Kestrel / AMPL

GAMS / AMPL via Kestrel

Metacomputing clients

MW, iMW, Condor / AMPL

Kestrel / AMPL for “parallel” submissions

Internet

The NEOS Server

`www-neos.mcs.anl.gov/neos/`

Over three dozen solvers, for

Linear programming

Linear network optimization

Linear integer programming

Nonlinear programming

Nonlinear integer programming

Nondifferentiable & global optimization

Stochastic linear programming

Complementarity problems

Semidefinite programming

Internet

NEOS Server Design

Flexible architecture

Central controller and scheduler machine

Distributed solver sites

Numerous formats

Low-level formats: MPS, SIF, SDPA

Programming languages: C/ADOL-C, Fortran/ADIFOR

High-level modeling languages: AMPL, GAMS, MP-MODEL

Varied submission options

E-mail

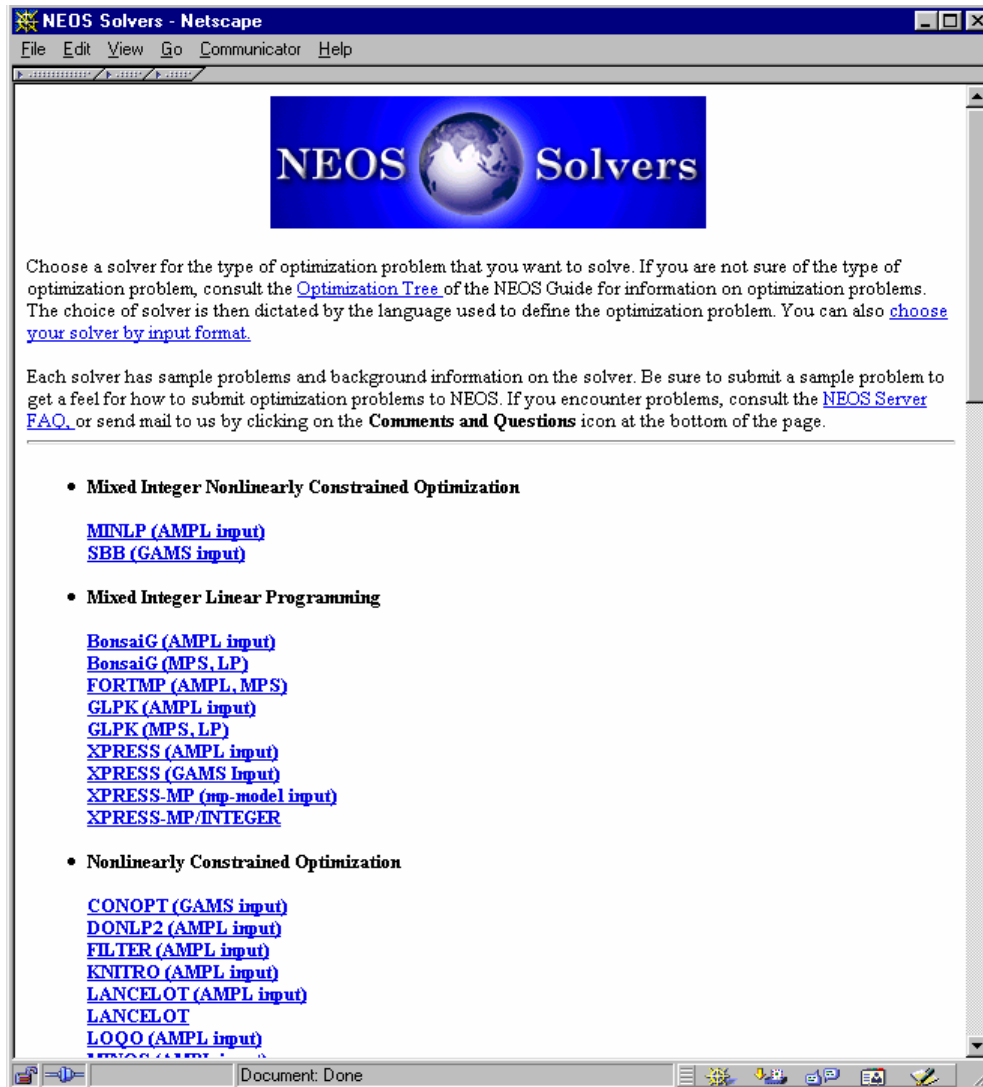
Web forms

TCP/IP socket-based submission tool: Java or tcl/tk

... handled 3195 submissions last week

... can accept submissions of new solvers, too

NEOS Solver Listing



NEOS Solvers

Choose a solver for the type of optimization problem that you want to solve. If you are not sure of the type of optimization problem, consult the [Optimization Tree](#) of the NEOS Guide for information on optimization problems. The choice of solver is then dictated by the language used to define the optimization problem. You can also [choose your solver by input format](#).

Each solver has sample problems and background information on the solver. Be sure to submit a sample problem to get a feel for how to submit optimization problems to NEOS. If you encounter problems, consult the [NEOS Server FAQ](#), or send mail to us by clicking on the **Comments and Questions** icon at the bottom of the page.

- **Mixed Integer Nonlinearly Constrained Optimization**
 - [MINLP \(AMPL input\)](#)
 - [SBB \(GAMS input\)](#)
- **Mixed Integer Linear Programming**
 - [BonsaiG \(AMPL input\)](#)
 - [BonsaiG \(MPS, LP\)](#)
 - [FORTMP \(AMPL, MPS\)](#)
 - [GLPK \(AMPL input\)](#)
 - [GLPK \(MPS, LP\)](#)
 - [XPRESS \(AMPL input\)](#)
 - [XPRESS \(GAMS Input\)](#)
 - [XPRESS-MP \(mp-model input\)](#)
 - [XPRESS-MP/INTEGER](#)
- **Nonlinearly Constrained Optimization**
 - [CONOPT \(GAMS input\)](#)
 - [DONLP2 \(AMPL input\)](#)
 - [FILTER \(AMPL input\)](#)
 - [KNITRO \(AMPL input\)](#)
 - [LANCELOT \(AMPL input\)](#)
 - [LANCELOT](#)
 - [LOQO \(AMPL input\)](#)
 - [MPS \(AMPL input\)](#)

Internet

Callable NEOS

General mechanism

CORBA-based connection

between local program and NEOS Server

API under development

to support varied specialized applications

Kestrel: a callable NEOS “solver” for AMPL (& GAMS)

`www-neos.mcs.anl.gov/neos/kestrel.html`

`www.ampl.com/ampl/REMOTE/`

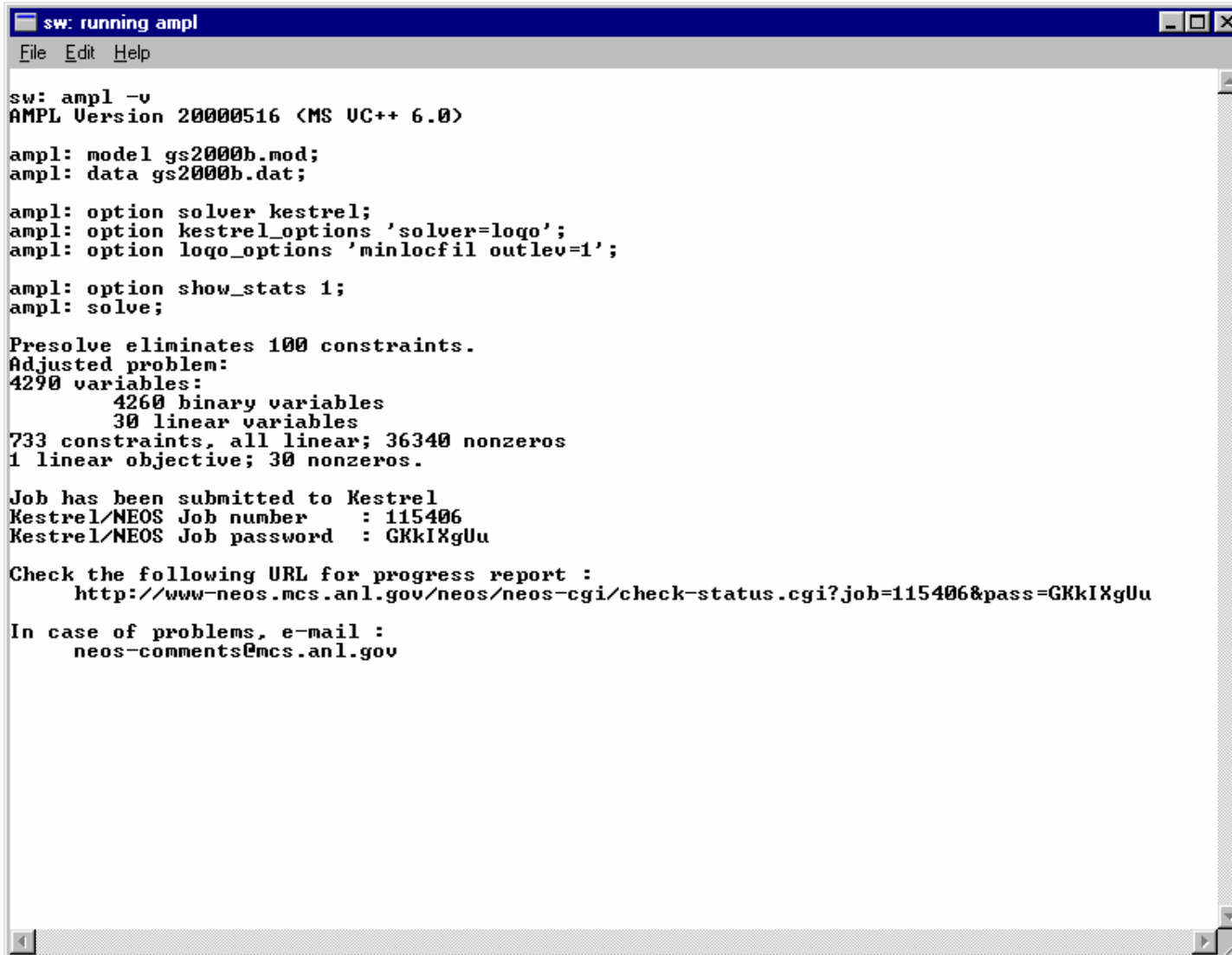
`www.gams.com/contrib/kestrel.htm`

Installs like a solver but provides a NEOS gateway

Combines convenient local modeling environment
with access to remote solvers

Internet

Kestrel Example



```
sw: running ampl
File Edit Help

sw: ampl -v
AMPL Version 20000516 (MS UC++ 6.0)

ampl: model gs2000b.mod;
ampl: data gs2000b.dat;

ampl: option solver kestrel;
ampl: option kestrel_options 'solver=logo';
ampl: option logo_options 'minlocfil outlev=1';

ampl: option show_stats 1;
ampl: solve;

Presolve eliminates 100 constraints.
Adjusted problem:
4290 variables:
    4260 binary variables
    30 linear variables
733 constraints, all linear; 36340 nonzeros
1 linear objective; 30 nonzeros.

Job has been submitted to Kestrel
Kestrel/NEOS Job number   : 115406
Kestrel/NEOS Job password : GKkIXgUu

Check the following URL for progress report :
  http://www-neos.mcs.anl.gov/neos/neos-cgi/check-status.cgi?job=115406&pass=GKkIXgUu

In case of problems, e-mail :
  neos-comments@mcs.anl.gov
```

Kestrel Example (*cont'd*)

```

sw: running ampl
File Edit Help
Check the following URL for progress report :
  http://www-neos.mcs.anl.gov/neos/neos-cgi/check-status.cgi?job=115406&pass=GKkIXgUu

In case of problems, e-mail :
  neos-comments@mcs.anl.gov

Intermediate Solver Output:
Checking the AMPL files
Executing algorithm...

LOQO 6.00: minlocfil
outlev=1

It's a QP.
ignoring integrality of 4260 variables

  1   0.000000e+00   2.1e+02   -4.263593e+05   1.7e+03
  2   2.839512e+03   1.1e+01   -4.206438e+05   8.8e+01
  3   2.803962e+03   5.8e-01   -3.084425e+05   3.7e+00
  4   1.804909e+03   7.0e-02   -2.965997e+04   1.4e-13           DF
  5   3.154594e+02   1.1e-02   -3.913235e+03   1.7e-13           DF
  6   3.771029e+01   1.2e-03   -2.201994e+02   4.6e-14           DF
  7   2.235023e+01   6.4e-04   -1.072541e+01   3.6e-14           DF
  8   1.700808e+01   3.1e-04   2.429905e+00   2.9e-14           DF
  9   1.536949e+01   1.4e-04   9.410120e+00   4.6e-14           DF
 10   1.446494e+01   4.4e-05   1.271394e+01   3.8e-14           1           DF
 11   1.405838e+01   2.4e-06   1.326864e+01   3.0e-14           1           DF
 12   1.400320e+01   1.4e-07   1.396308e+01   3.5e-14           3           PF DF
 13   1.400016e+01   7.3e-09   1.399815e+01   3.9e-14           4           PF DF
 14   1.400001e+01   3.6e-10   1.399991e+01   3.5e-14           5           PF DF
 15   1.400000e+01   1.8e-11   1.400000e+01   4.1e-14           6           PF DF
 16   1.400000e+01   9.1e-13   1.400000e+01   3.5e-14           8           PF DF

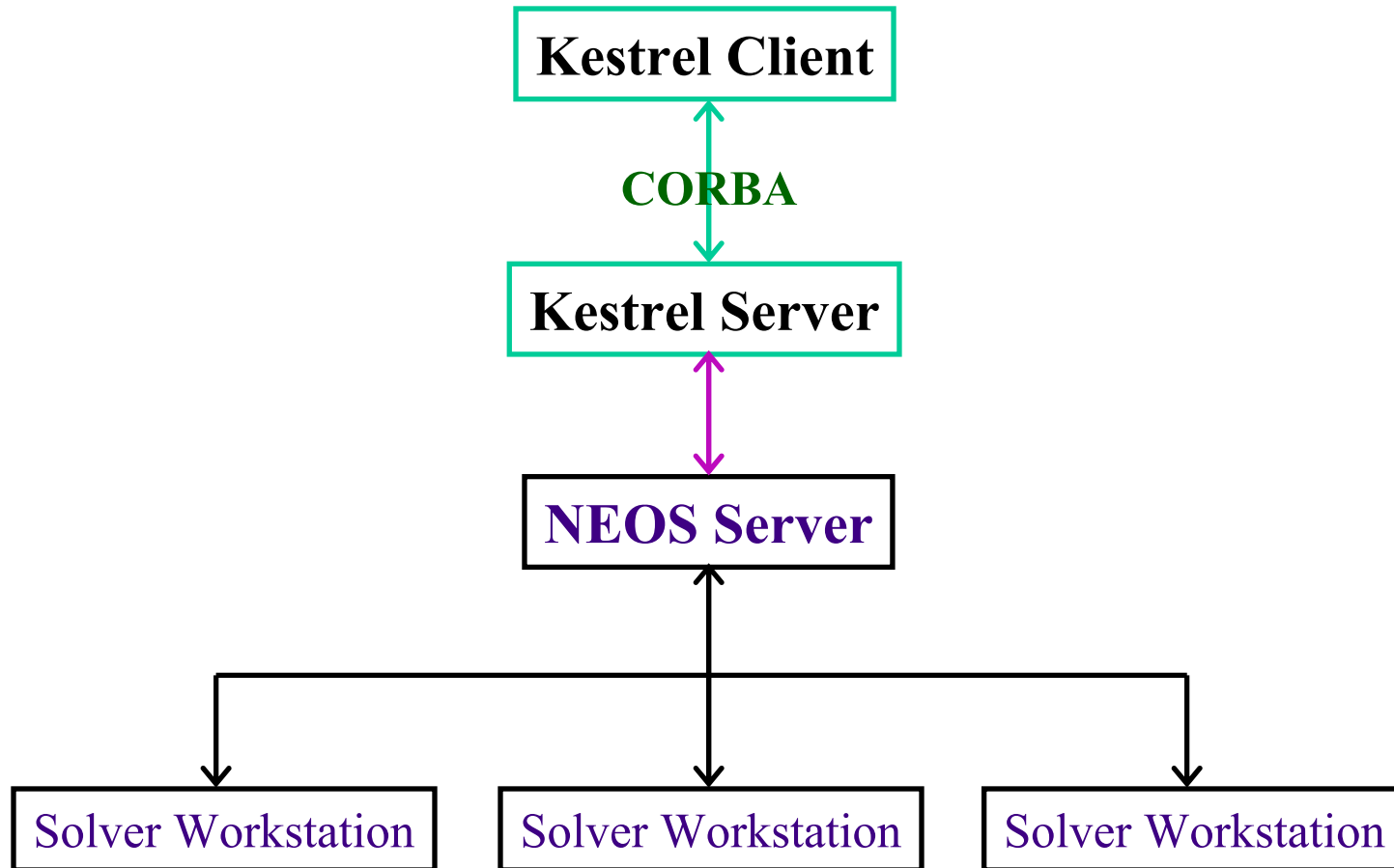
Finished call

LOQO 6.00: optimal solution <16 QP iterations, 31 evaluations>
primal objective 14.00000002
dual objective 13.99999977

ampl: display MinNotDom, MaxNotDom;
:
:           MinNotDom MaxNotDom      :=
Office Americas      3           4
:
ampl:

```

Outline of Kestrel Operations



Internet

Other Kestrel Enhancements

Connections to results

Web connection to intermediate results

Kestrel client access to recent results

after connection to Kestrel server has been broken

. . . job number and password required

Flexibility

NEOS Server now accepts binary data from any input source

Jobs on many solver workstations may be killed by user

Kestrel “Parallel” Processing

Invoke Kestrel within a loop

`kestrelsub` submits multiple solve requests

NEOS Server distributes requests to multiple workstations
(queueing some if necessary)

`kestrelret` retrieves requests in the order they were sent

Example from a decomposition script

```
for {p in PROD} {  
    problem SubI[p];  
    commands kestrelsub;  
};  
  
for {p in PROD} {  
    problem SubI[p];  
    commands kestrelret;  
    display Artif_Reduced_Cost[p];  
    ...  
};
```

Solver-Specific Directives & Results

System-defined “dot suffixes” (original)

Built into AMPL: **Trans.ub**, **Balance.slack**, etc.

User-defined suffixes

Define your own suffixes for use in AMPL scripts

Define solver-specific suffixes to send directives
relating to individual variables, constraints, objectives

Solver-defined suffixes

Define new suffixes
to return additional, solver-specific values

... support post-solution analyses

Suffixes

User-Defined

```
ampl: model multmip1.mod;
ampl: data multmip1.dat;
ampl: solve;
CPLEX 5.0: optimal integer solution;
  objective 229850
470 simplex iterations
112 branch-and-bound nodes
```

WITHOUT

```
ampl: model multmip1.mod;
ampl: data multmip1.dat;
ampl: suffix priority integer, > 0, < 5000;
ampl: let {i in ORIG, j in DEST}
ampl? Use[i,j].priority := sum {p in PROD} demand[j,p];
ampl: solve;
CPLEX 5.0: optimal integer solution;
  objective 229850
253 simplex iterations
49 branch-and-bound nodes
```

WITH

Definition Command

```
suffix suffix-name  
type-phraseopt restriction-phraseopt ... inout-phraseopt i
```

type-phrase, restriction-phrase

Similar to **param** declarations

inout-phrase

Specifies treatment of suffix
when a solver is invoked (by **solve**) . . .

Suffixes

Input-Output Options

	Pass values to solver when invoked?	Read values from solver on termination?
OUT	yes	no
IN	no	yes
INOUT	yes	yes
LOCAL	no	no

Examples

OUT: integer variable priorities, time periods

IN: sensitivity, infeasibility information

INOUT (*default*): simplex basis statuses

LOCAL: values special to your application

Suffixes

Solver-Defined

```
AMPL: model diet.mod;
AMPL: data diet2.dat;

AMPL: option solver cplex;

AMPL: solve;
CPLEX 5.0: infeasible problem
7 iterations (7 in phase I)

AMPL: option cplex_options 'iisfind=1';

AMPL: solve;
CPLEX 5.0: iisfind=1
CPLEX 5.0: infeasible problem
0 iterations

Returning iis of 7 variables and 2 constraints.

suffix iis symbolic OUT;

option iis_table '\
0      non      not in the iis\
1      low      at lower bound\
2      fix      fixed\
3      upp      at upper bound\
';
```

(continued)

Solver-Defined Key Tables

Infeasibility analysis (cont'd)

`suffix suff symbolic` induces AMPL option `suff_table`

Solver returns integers, but

strings from table are used and displayed

```
AMPL: option iis_table;
option iis_table '\
0      non      not in the iis\
1      low      at lower bound\
2      fix      fixed\
3      upp      at upper bound\
';

AMPL: display {i in 1.._ncons: _con[i].iis <> "non"}
AMPL?      (_conname[i],_con[i].iis);

:      _conname[i]  _con[i].iis :=
3      "diet['B2']"  low
5      "diet['NA']"  upp
```

Suffixes

Solver-Defined

Example: Sensitivity ranges

```
ampl: model net3.mod;
ampl: data net3.dat;

ampl: option solver cplex;
ampl: option cplex_options 'sensitivity';
ampl: solve;

CPLEX 5.0: optimal solution; objective 1819
3 iterations (2 in phase I)

suffix up OUT;
suffix down OUT;
suffix current OUT;

ampl: display _varname, _var.down, _var.current, _var.up;

: _varname          _var.down _var.current _var.up :=
1 "PD_Ship['NE']"   -1e+20      2.5         3
2 "PD_Ship['SE']"     3          3.5        1e+20
3 "DW_Ship['NE','BOS']" -1e+20     1.7        1e+20
4 "DW_Ship['NE','EWR']" -1e+20     0.7         1.8
5 "DW_Ship['NE','BWI']" 0.2        1.3         1.8
...
```


Suffixes

Solver-Defined

Example: Ray of unboundedness

From Benders decomposition of a location-transportation problem

```
problem Master: Build, Max_Ship_Cost, Total_Cost, Cut_Defn;
problem Sub: Supply_Price, Demand_Price, Dual_Ship_Cost, Dual_Ship;
suffix unbdd OUT;

let nCUT := 0;
let Max_Ship_Cost := 0;
let {i in ORIG} build[i] := 1;
param GAP default Infinity;

repeat {
  solve Sub;
  if Dual_Ship_Cost <= Max_Ship_Cost + 0.00001
    then break;
  if Sub.result = "unbounded" then { ...
```

(continued)

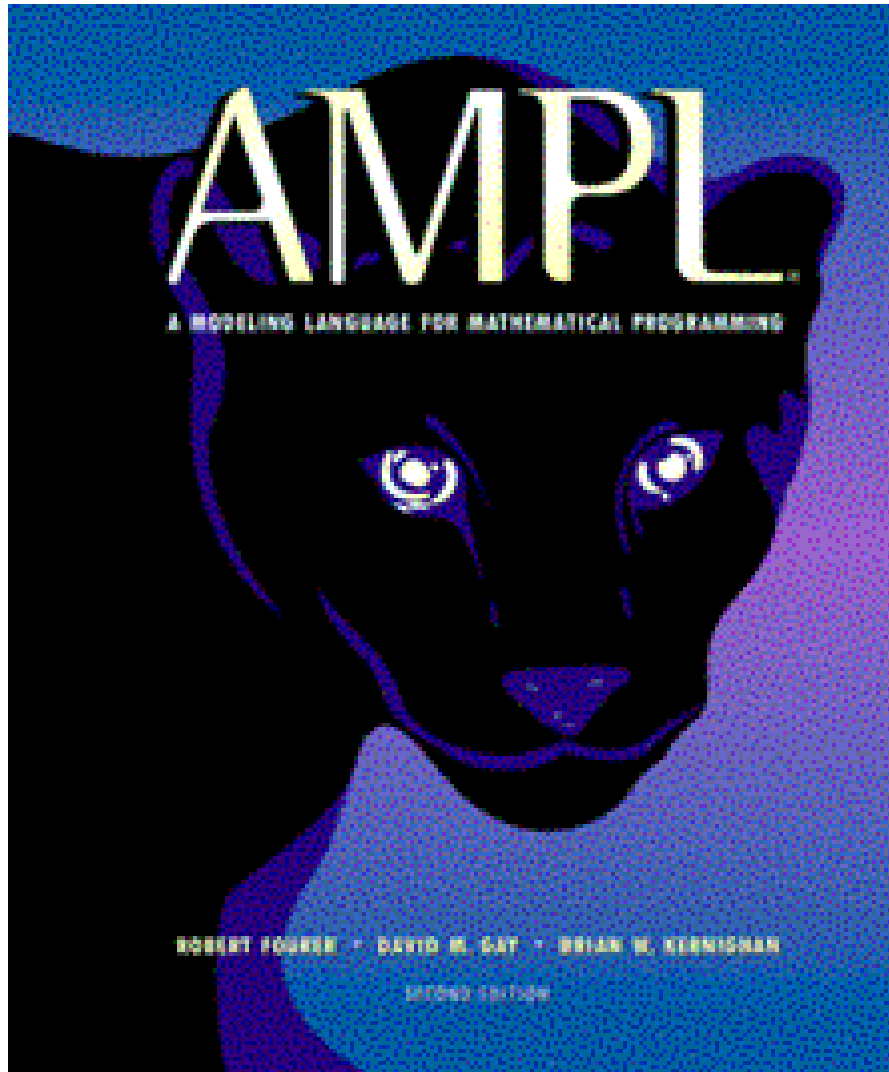
Suffixes

Solver-Defined

Example: Ray of unboundedness (cont'd)

```
if Sub.result = "unbounded" then {
  let nCUT := nCUT + 1;
  let cut_type[nCUT] := "ray";
  let {i in ORIG} supply_price[i,nCUT] := Supply_Price[i].unbdd;
  let {j in DEST} demand_price[j,nCUT] := Demand_Price[j].unbdd;
}
else { ...
}
solve Master;
let {i in ORIG} build[i] := Build[i];
};
```

AMPL Book **2nd Edition** Now Available



2nd Edition Features

New chapters

Database access

Command scripts

Modeling commands

Interactions with solvers

Display commands

Complementarity problems

... all extensions previously only described roughly at web site

Updates and improvements

Existing chapters extensively revised

Updated reference manual provided as appendix

... and at half the recent price of the 1st edition!

... see www.aml.com/BOOK/