AMPL "Hands-On" Session

Robert Fourer

Department of Industrial Engineering & Management Sciences
Northwestern University

Institute for Mathematics and Its Applications

Annual Program: Optimization

Supply Chain and Logistics Optimization Tutorial

September 9-13, 2002

The McDonald's Diet Problem

Foods:		*Nutrients:*	
QP	*Quarter Pounder*	*Prot*	*Protein*
FR	*Fries, small*	*Iron*	*Iron*
MD	*McLean Deluxe*	*VitA*	*Vitamin A*
SM	*Sausage McMuffin*	*Cals*	*Calories*
BM	*Big Mac*	*VitC*	*Vitamin C*
1M	*1% Lowfat Milk*	*Carb*	*Carbohydrates*
FF	*Filet-O-Fish*	*Calc*	*Calcium*
OJ	*Orange Juice*		
MC	*McGrilled Chicken*		

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

Minimize	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_{1M}	+ 0.72 x_{OJ}	
Subject to	28 x_{QP}	+ 24 x_{MD}	+ 25 x_{BM}	+ 14 x_{FF}	+ 31 x_{MC}	+ 3 x_{FR}	+ 15 x_{SM}	+ 9 x_{1M}	+ 1 x_{OJ}	≥ 55
	15 x_{QP}	+ 15 x_{MD}	+ 6 x_{BM}	+ 2 x_{FF}	+ 8 x_{MC}	+ 0 x_{FR}	+ 4 x_{SM}	+ 10 x_{1M}	+ 2 x_{OJ}	≥ 100
	6 x_{QP}	+ 10 x_{MD}	+ 2 x_{BM}	+ 0 x_{FF}	+ 15 x_{MC}	+ 15 x_{FR}	+ 0 x_{SM}	+ 4 x_{1M}	+ 120 x_{OJ}	≥ 100
	30 x_{QP}	+ 20 x_{MD}	+ 25 x_{BM}	+ 15 x_{FF}	+ 15 x_{MC}	+ 0 x_{FR}	+ 20 x_{SM}	+ 30 x_{1M}	+ 2 x_{OJ}	≥ 100
	20 x_{QP}	+ 20 x_{MD}	+ 20 x_{BM}	+ 10 x_{FF}	+ 8 x_{MC}	+ 2 x_{FR}	+ 15 x_{SM}	+ 0 x_{1M}	+ 2 x_{OJ}	≥ 100
	510 x_{QP}	+ 370 x_{MD}	+ 500 x_{BM}	+ 370 x_{FF}	+ 400 x_{MC}	+ 220 x_{FR}	+ 345 x_{SM}	+ 110 x_{1M}	+ 80 x_{OJ}	≥ 2000
	34 x_{QP}	+ 35 x_{MD}	+ 42 x_{BM}	+ 38 x_{FF}	+ 42 x_{MC}	+ 26 x_{FR}	+ 27 x_{SM}	+ 12 x_{1M}	+ 20 x_{OJ}	≥ 350

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 $\Sigma_{j \in \mathcal{F}} \; c_j \, x_j$

Subject to $\Sigma_{j \in \mathcal{F}} \; a_{ij} \, x_j \; \geq \; b_i$, 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 way that people do
- Support varied solvers

Features

Sets and Indexing

Simple sets

Sets of objects
Sets of numbers
Ordered sets

Compound sets

Sets of pairs
Sets of n-tuples
Indexed collections of sets

Computing sets

By enumerating set members
By operating on sets: union, intersection, . . .

Sets & Indexing

Sets of Objects

Transportation

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

param supply {ORIG} >= 0;   # at origins
param demand {DEST} >= 0;   # at destinations
   check: sum {i in ORIG} supply[i] = sum {j in DEST} demand[j];

param cost {ORIG,DEST} >= 0;  # ship cost/unit
var Trans {ORIG,DEST} >= 0;   # units shipped

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

subject to Supply {i in ORIG}:
   sum {j in DEST} Trans[i,j] = supply[i];
subject to Demand {j in DEST}:
   sum {i in ORIG} Trans[i,j] = demand[j];
```

Sets & Indexing

Sets of Numbers

Multi-period production

```
set PROD;     # products
param T > 0;  # number of weeks

param rate {PROD} > 0;
param inv0 {PROD} >= 0;
param avail {1..T} >= 0;
param market {PROD,1..T} >= 0;

param prodcost {PROD} >= 0;
param invcost {PROD} >= 0;
param revenue {PROD,1..T} >= 0;

var Make {PROD,1..T} >= 0;
var Inv {PROD,0..T} >= 0;
var Sell {p in PROD, t in 1..T} >= 0, <= market[p,t];

maximize Total_Profit:
   sum {p in PROD, t in 1..T}
     (revenue[p,t] * Sell[p,t]
        - prodcost[p] * Make[p,t] - invcost[p] * Inv[p,t]);

subj to Time {t in 1..T}:
   sum {p in PROD} (1/rate[p]) * Make[p,t] <= avail[t];

subj to Initial {p in PROD}: Inv[p,0] = inv0[p];

subj to Balance {p in PROD, t in 1..T}:
   Make[p,t] + Inv[p,t-1] = Sell[p,t] + Inv[p,t];
```


Sets & Indexing

Subsets

Diet

```
set FOOD;  # foods

param cost {FOOD} > 0;
param f_min {FOOD} >= 0;
param f_max {j in FOOD} >= f_min[j];

set NUTR;  # nutrients

set MINREQ within NUTR;  # nutrients with minimum requirements
set MAXREQ within NUTR;  # nutrients with maximum requirements

param n_min {MINREQ} >= 0;
param n_max {MAXREQ} >= 0;

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_Min {i in MINREQ}:
   sum {j in FOOD} amt[i,j] * Buy[j] >= n_min[i];

subject to Diet_Max {i in MAXREQ}:
   sum {j in FOOD} amt[i,j] * Buy[j] <= n_max[i];
```

Sets & Indexing

Ordered Sets

Multi-period production

```
set PROD;
set WEEKS ordered;
param rate {PROD} > 0;
param inv0 {PROD} >= 0;
param avail {WEEKS} >= 0;
param market {PROD,WEEKS} >= 0;
. . . . .
var Make {PROD,WEEKS} >= 0;
var Inv {PROD,WEEKS} >= 0;
var Sell {p in PROD, t in WEEKS} >= 0, <= market[p,t];
. . . . .
subj to balance {p in PROD, t in WEEKS}:
   Make[p,t] + (if t = first(WEEKS) then inv0[p] else Inv[p,prev(t)])
      = Sell[p,t] + Inv[p,t];
```

```
set PROD := bands coils ;
set WEEKS := 27sep93 04oct93 11oct93 18oct93 ; . . . . .
```

. . . also options to declare induced orderings

Sets & Indexing

Sets of Pairs

Transportation

```
set ORIG;   # origins
set DEST;   # destinations
set LINKS within {ORIG,DEST};

param supply {ORIG} >= 0;
param demand {DEST} >= 0;

param cost {LINKS} >= 0;
var Trans {LINKS} >= 0;

minimize total_cost:
  sum {(i,j) in LINKS} cost[i,j] * Trans[i,j];

subject to Supply {i in ORIG}:
  sum {(i,j) in LINKS} Trans[i,j] = supply[i];
subject to Demand {j in DEST}:
  sum {(i,j) in LINKS} Trans[i,j] = demand[j];
```

Sets & Indexing

Sets of Pairs: Data

```
param: ORIG: supply :=
    GARY 1400  CLEV 2600  PITT 2900 ;
param: DEST:  demand :=
    FRA  900   DET 1200   LAN  600   WIN  400
    STL 1700   FRE 1100   LAF 1000 ;
param: LINKS:  cost :=
    GARY DET 14  GARY LAN 11  GARY STL 16
    GARY LAF  8  CLEV FRA 27  CLEV DET  9
    CLEV LAN 12  CLEV WIN  9  CLEV STL 26
    CLEV LAF 17  PITT FRA 24  PITT WIN 13
    PITT STL 28  PITT FRE 99 ;
```

```
ampl: display {i in ORIG}: {(i,j) in LINKS};
set {('GARY',j) in LINKS} := DET LAN STL LAF;
set {('CLEV',j) in LINKS} := FRA DET LAN WIN STL LAF;
set {('PITT',j) in LINKS} := FRA WIN STL FRE;
ampl: display cost;
:    DET  FRA  FRE  LAF  LAN  STL  WIN :=
CLEV   9   27    .   17   12   26    9
GARY  14    .    .    8   11   16    .
PITT   .   24   99    .    .   28   13 ;
```


Sets & Indexing

Sets of *n*-tuples

Airline Fleet Assignment

```
set FLEETS;
set CITIES;
set TIMES circular;
set LEGS within
   {c1 in CITIES, c2 in CITIES,
    t1 in TIMES,  t2 in TIMES: c1 <> c2 and t1 <> t2};

set HOOKS within
   {(c1,c2,t1,t2) in LEGS,
    (c3,c4,t3,t4) in LEGS: c2 = c3};

param cost {FLEETS,LEGS} >= 0;
set FL_LEGS :=
   {f in FLEETS, (c1,c2,t1,t2) in LEGS: cost[f,c1,c2,t1,t2] < BIG};
```

Sets & Indexing

Indexed Collections of Sets

Multicommodity Transportation

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

set orig {PROD} within ORIG;
set dest {PROD} within DEST;
set links {p in PROD} := orig[p] cross dest[p];

param supply {p in PROD, orig[p]} >= 0;
param demand {p in PROD, dest[p]} >= 0;

param limit {ORIG,DEST} >= 0;

param cost {p in PROD, links[p]} >= 0;
var Trans {p in PROD, links[p]} >= 0;

minimize total_cost:
  sum {p in PROD, (i,j) in links[p]} cost[p,i,j] * Trans[p,i,j];

subj to Supply {p in PROD, i in orig[p]}:
  sum {j in dest[p]} Trans[p,i,j] = supply[p,i];
subj to Demand {p in PROD, j in dest[p]}:
  sum {i in orig[p]} Trans[p,i,j] = demand[p,j];
subj to Multi {i in ORIG, j in DEST}:
  sum {p in PROD: (i,j) in links[p]} Trans[p,i,j] <= limit[i,j];
```


Sets & Indexing

Computed Sets

Airline Fleet Assignment

```
set SERV_CITIES {f in FLEETS} :=
   union {(c1,c2,t1,t2) in 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 {(c,c2,t1,t2) in LEGS} t1 union
   setof {(c1,c,t1,t2) in LEGS} next(t2,TIMES,turn[f,c]);
         # 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
```

. . . *also enumerate set members:* `setof`, `{ . . . }`

. . . *operate on sets:* `union`, `int`, `diff`, `symdiff`

Features

Objectives and Constraints

(Linear)

Nonlinear

Integer

Network

Objectives & Constraints

Nonlinear

Established features

Nonlinear expressions in the variables

Derivatives supplied to algorithms

Fast "automatic differentiation"

Refinements

Nondifferentiable expressions

Substitution of lengthy or common subexpressions

User-supplied functions

Detection of partially separable structure

2nd derivatives

Nonlinear

Nonlinear Objective

Traffic Flow

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

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

   check: sum {i in ORIG} supply[i] = sum {j in DEST} demand[j];

param rate {ORIG,DEST} >= 0;   # base shipment costs per unit
param limit {ORIG,DEST} > 0;   # limit on units shipped

var Trans {i in ORIG, j in DEST}
   >= 1e-10, <= .9999 * limit[i,j], := limit[i,j]/2;
                               # actual units to be shipped

minimize Total_Cost:
   sum {i in ORIG, j in DEST}
      rate[i,j] * Trans[i,j]^0.8 / (1 - Trans[i,j]/limit[i,j]);

subject to Supply {i in ORIG}:
   sum {j in DEST} Trans[i,j] = supply[i];

subject to Demand {j in DEST}:
   sum {i in ORIG} Trans[i,j] = demand[j];
```


Nonlinear

Nonlinear Constraints

Largest "Small" Polygon

```
param N integer > 0;
param pi := 4*atan(1.);

var rho {i in 1..N} <= 1, >= 0, := 4*i*(N+1-i) / (N+1)**2;
          # polar radius (distance to fixed vertex)
var the {i in 1..N} >= 0, := pi*i/N;
          # polar angle (measured from fixed direction)

maximize area:
   0.5 * sum{i in 2..N} rho[i] * rho[i-1] * sin(the[i]-the[i-1]);

subject to cd {i in 1..N, j in i+1 .. N}:
   rho[i]**2 + rho[j]**2 - 2*rho[i]*rho[j]*cos(the[j]-the[i]) <= 1;

subject to ac {i in 2..N}: the[i] >= the[i-1];
subject to fix_theta: the[N] = pi;
subject to fix_rho: rho[N] = 0;
```

Nonlinear

Nonlinear Solvers

```
ampl: model nltranse.mod
ampl: data nltrans.dat;
ampl: option solver minos;
ampl: solve;
MINOS 5.5: optimal solution found.
40 iterations, objective 355438.2006
Nonlin evals: obj = 78, grad = 77.

ampl: option reset_initial_guesses 1;
ampl: option solver snopt;
ampl: solve;
SNOPT 6.1-1:
  Primal feasible solution; could not satisfy dual feasibility.
444 iterations, objective 354276.7272
Nonlin evals: obj = 213, grad = 212.
ampl:
```


Nonlinear

Derivative Computations

Nonlinear expressions

- Automatic differentiation
- Function & gradient interface routines

2nd derivatives

- Dense or sparse storage

Detection of separability

- Partially separable structure
- Group partially separable structure

Use of separability

- Faster 2nd derivative computations
- Faster optimization methods

Nonlinear

Automatic Differentiation

Computations

Forward sweep: compute $\phi(x)$,
save info on $\partial f(x)/\partial o$ for each operation o

Backward sweep: recur to compute $\nabla f(x)$

Complexity

Small multiple of *time* for $\phi(x)$ alone

Potentially large multiple of *space*

Nonlinear

Nondifferentiable Expressions

Interval of instability

```
if abs(x[j]) > delta
   then log(1+x[j]) / x[j]
   else 1 - x[j]/2
```

Composite

```
head {(i,j) in pipes}:
   H[i,j] = cv^2 * corr[i,j] *
      (if Q[i,j] >= 0 then Q[i,j]^2 else -Q[i,j]^2);
```

. . . appropriate derivatives do get computed

Nonlinear

Lengthy Subexpressions

Definition by constraints

```
var T >= 0;  # temperature
var I >= 0;  # thickness of insulation
var P;   # pressure
var CI;  # insulation cost
var CV;  # vessel cost
var CR;  # recondensation cost

minimize Total_Cost:  CI + CV + CR;

subj to P_def: P = exp(-3950/(T+460)+11.86);
subj to CI_def: CI = 400 * I^0.9;
subj to CV_def: CV = 1000 + 22 * (P-14.7)^1.2;
subj to CR_def: CR = 144 * (80-T) / I;
```


Nonlinear

Lengthy Subexpressions

Definition by declarations

```
var T >= 0;
var I >= 0;

var P = exp(-3950/(T+460)+11.86);
var CI = 400 * I^0.9;
var CV = 1000 + 22 * (P-14.7)^1.2;
var CR = 144 * (80-T) / I;

minimize Total_Cost:  CI + CV + CR;
```

Nonlinear

Common Subexpressions

Definition by constraints

```
equil6: K6*sqrt(n2)*sqrt(n4) = sqrt(n1)*n6*sqrt(p/nT);
equil7: K7*sqrt(n1)*sqrt(n2) = sqrt(n4)*n7*sqrt(p/nT);
equil8: K8*n1 = n4*n8*(p/nT);
equil9: K9*n1*sqrt(n3) = n4*n9*sqrt(p/nT);
equil10: K10*n1^2 = n4^2*n10*(p/nT);
total_defn: nT = n1+n2+n3+n4+n5+n6+n7+n8+n9+n10;
```

. . . substitution can be switched on or off

Definition by declarations

```
var nT = n1+n2+n3+n4+n5+n6+n7+n8+n9+n10;
equil6: K6*sqrt(n2)*sqrt(n4) = sqrt(n1)*n6*sqrt(p/nT);
equil7: ...
```

. . . more flexible and selective

Nonlinear

Substitutions for Subexpressions

without substitution

```
ampl: model ammonia.mod;
ampl: solve;

6 variables:
        3 nonlinear variables
        3 linear variables
4 constraints, all nonlinear; 9 linear nonzeros
1 linear objective; 3 nonzeros.

MINOS 5.5: optimal solution found.
33 iterations, objective 5339.252824
```

with substitution

```
ampl: model ammonia.mod;
ampl: option substout 1;
ampl: solve;

Substitution eliminates 4 variables.

Adjusted problem:
2 variables, all nonlinear
0 constraints
1 nonlinear objective; 2 linear nonzeros.

MINOS 5.5: optimal solution found.
4 iterations, objective 5339.252824
```

Nonlinear

User-Defined Functions

Declaration

Specification of calling protocols:

`function` *name* (*domain-list*) *options* **`;`**

domain-list specifies sets in which arguments must lie

return value may be numeric, symbolic, "random"

Definition

Through user's C program

Refinements

Evaluate where needed

- in model
- while solving

Link dynamically

- DLLs for Windows PCs
- shared libraries for Unix systems

Robert Fourer, IMA Tutorials, 9 Sept 2002: AMPL "Hands-On" Session

Objectives & Constraints

Integer

Established features

Integer variables

Refinements

Piecewise-linear functions of variables

Network modeling extensions

. . . combinatorial extensions to be mentioned later

Robert Fourer, IMA Tutorials, 9 Sept 2002: AMPL "Hands-On" Session

Integer

Integer Variables

Declare like ordinary variables

```
var Ship >= 0, integer;
var Use integer >= 0, <= 1;
var Use binary;
```

Translation is automatic

Bounds on integer variables rounded

Integer information sent to solver

. . . some logical conditions
(special ordered sets) can be recognized

Integer

Multicommodity Model

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

param supply {ORIG,PROD} >= 0;
param demand {DEST,PROD} >= 0;
param limit {ORIG,DEST} >= 0;

param minload >= 0;
param maxserve integer > 0;

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

param fcost {ORIG,DEST} >= 0;
var Use {ORIG,DEST} binary;

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];
```

Integer

Multicommodity Constraints

```
subj to Supply {i in ORIG, p in PROD}:
  sum {j in DEST} Trans[i,j,p] = supply[i,p];
subj to Max_Serve {i in ORIG}:
  sum {j in DEST} Use[i,j] <= maxserve;
subj to Demand {j in DEST, p in PROD}:
  sum {i in ORIG} Trans[i,j,p] = demand[j,p];
subj to Multi {i in ORIG, j in DEST}:
  sum {p in PROD} Trans[i,j,p] <= limit[i,j] * Use[i,j];
subj to Min_Ship {i in ORIG, j in DEST}:
  sum {p in PROD} Trans[i,j,p] >= minload * Use[i,j];
```

```
set ORIG := GARY CLEV PITT ;
set DEST := FRA DET LAN WIN STL FRE LAF ;
set PROD := bands coils plate ;
param minload := 375 ;
param maxserve := 5 ;
param supply (tr):  GARY   CLEV   PITT :=
            bands    400    700    800
            coils    800   1600   1800
            plate    200    300    300 ;
param demand (tr): .....
```


Integer

Multicommodity Optimization

```
ampl: model multmip3.mod;
ampl: data multmip3.dat;
ampl: option solver cplex;
ampl: option cplex_options °mipdisplay=1°;
ampl: solve;

CPLEX 8.0.0: mipdisplay 1
MIP emphasis: balance optimality and feasibility
Node log . . .
Best integer = 2.371250e+005  Node = 35  Best node = 2.344660e+005
Best integer = 2.356250e+005  Node = 40  Best node = 2.352110e+005
Cover cuts applied:  2
Implied bound cuts applied:  16
Flow cuts applied:  11
Gomory fractional cuts applied: 5

CPLEX 8.0.0: optimal integer solution; objective 235625
547 MIP simplex iterations
41 branch-and-bound nodes
```

Integer

Multicommodity Results

```
ampl: option omit_zero_rows 1;
ampl: display Trans;
Trans [CLEV,*,*]
:   bands coils plate  :=
DET     0   525   100
FRA   275    50    50
FRE     0   450   100
LAN   100   400     0
STL   325   175    50
 [GARY,*,*]
:   bands coils plate  :=
LAF     0   400     0
STL   325   150   150
WIN    75   250    50
 [PITT,*,*]
:   bands coils plate  :=
DET   300   225     0
FRA    25   450    50
FRE   225   400     0
LAF   250   100   250
STL     0   625     0
```


Objectives & Constraints

Network

Network flow linear program

Define nodes, arcs, data, variables:

```
set N;                    # nodes
set A within N cross N;   # arcs
param b {N};              # inflow or outflow
param c {A} >= 0;         # flow cost
param u {A} >= 0;         # flow upper bound
var Flow {(i,j) in A} >= 0, <= u[i,j];
```

Flow balance at nodes, *explicit:*

```
subject to Balance {i in N}:
   sum {j in N: (i,j) in A} Flow[i,j] -
   sum {j in N: (j,i) in A} Flow[j,i] = b[i];
```

Flow balance at nodes, *implicit:*

```
subject to Balance {i in N}:
   sum {(i,j) in A} Flow[i,j] -
   sum {(j,i) in A} Flow[j,i] = b[i];
```

. . . why not declare nodes and arcs directly?

Network

Node Declarations

Transshipment

```
node Balance {i in N};
```

Source or sink

```
node Balance {i in N}: net_out = b[i];
```

Various kinds

```
node RT: rt_min <= net_out <= rt_max;
node OT: ot_min <= net_out <= ot_max;

node P_RT {fact};
node P_OT {fact};

node M {prod,fact};
node D {prod,dctr};

node W {p in prod, w in whse}:
   net_in = dem[p,w];
```


Network

Arc Declarations

Pure network

```
arc Flow {(i,j) in A}:
   from Balance[i], to Balance[j],
   >= 0, <= u[i,j], obj total_cost c[i,j];
```

Generalized network

```
arc Manu_RT {p in prd, f in fact: rpc[p,f] <> 0}:
   >= 0,
   from P_RT[f],
   to M[p,f] (dp[f]*hd[f]/pt[p,f]),
   obj cost (rpc[p,f]*dp[f]*hd[f]/pt[p,f]);
```

. . . objective must be previously declared

Networks

Multicommodity Flows

Side constraints

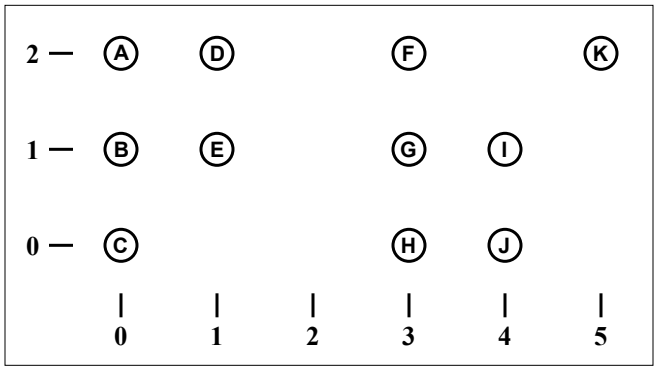
```
node Balance {i in N, p in P}: net_out = b[i,p];
arc Flow {(i,j) in A, p in P}:
     from Balance[i,p], to Balance[j,p],
     >= 0, <= u[i,j,p],
     obj total_cost c[i,j,p];
subject to Mult {(i,j) in A}:
     sum {p in P} Flow[i,j,p] <= u_mult[i,j];
```

Side variables

```
var Feed {f in F, i in N} >= 0, <= u_feed[f,i];
node Balance {i in N, p in P}: net_out =
     b[i,p] + sum {f in F} a[p,f] * Feed[f,i];
arc Flow {(i,j) in A, p in P}:
     from Balance[i,p], to Balance[j,p],
     >= 0, <= u[i,j,p], obj total_cost c[i,j,p];
```


Networks

Traveling Salesman Problem



Traveling
Implied Integer Program

```
set CITIES ordered;
param hpos {CITIES} >= 0;
param vpos {CITIES} >= 0
set PAIRS := {i in CITIES, j in CITIES: ord(i) < ord(j)};
param dist {(i,j) in PAIRS}
    := sqrt((hpos[j]-hpos[i])^2 + (vpos[j]-vpos[i])^2);

# ================================

var Travel {PAIRS} integer >= 0, <= 1;

minimize Tour_Length:
   sum {(i,j) in PAIRS} dist[i,j] * Travel[i,j];

subject to Visit_All {i in CITIES}:
   sum {(i,j) in PAIRS} Travel[i,j] +
   sum {(j,i) in PAIRS} Travel[j,i] = 2;
```


Traveling
Data for the Integer Program

```
param: CITIES: hpos vpos :=
         A      0    2
         B      0    1
         C      0    0
         D      1    2
         E      1    1
         F      3    2
         G      3    1
         H      3    0
         I      4    1
         J      4    0
         K      5    2  ;
```

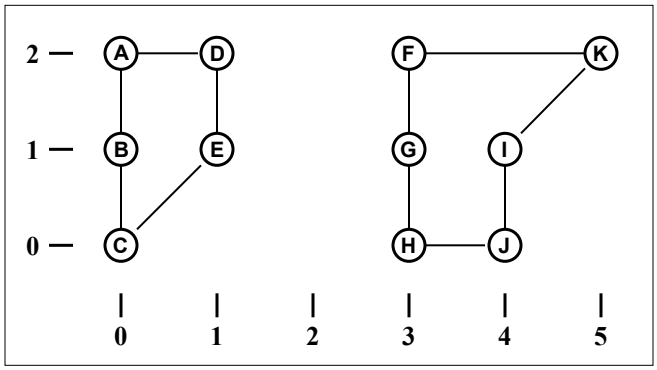
Traveling

Solution *(as a linear program)*

```
ampl: model travel.mod
ampl: data travel.dat
ampl: solve;
MINOS 5.5: ignoring integrality of 55 variables
MINOS 5.5: optimal solution found.
37 iterations, objective 12.82842712
ampl: option display_1col 1000;
ampl: option omit_zero_rows 1;
ampl: option display_eps .000001;
ampl: display Travel;
Travel :=
A B   1
A D   1
B C   1
C E   1
D E   1
F G   1
F K   1
G H   1
H J   1
I J   1
I K   1 ;
```


Networks

Solution *(showing 2 subtours)*



. . . total distance = 12.828

Traveling

Equivalent Integer Program

```
. . . . . .
param nSub >= 0;
set SUB {1..nSub} within CITIES;

subject to Subtour_Elimination {k in 1..nSub}:
   sum {i in SUB[k], j in CITIES diff SUB[k]} Travel[i,j] +
   sum {i in CITIES diff SUB[k], j in SUB[k]} Travel[i,j] >= 2;
```

and data for one subtour:

```
. . . . . .
param nSub := 1 ;
set SUB[1] := A B C D E ;
```

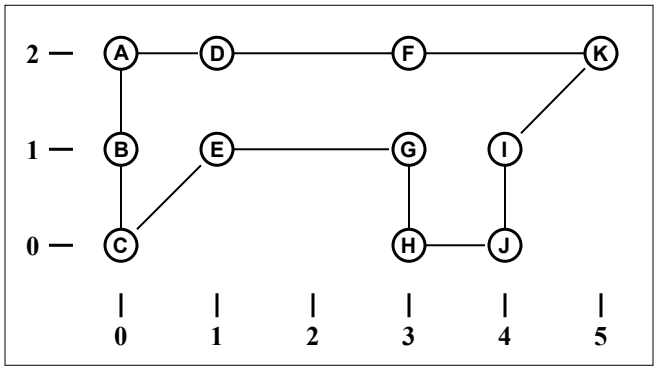

Traveling

Solution *(as a linear program)*

```
ampl: reset;
ampl: model travel2.mod
ampl: data travel2.dat
ampl: solve;
MINOS 5.5: ignoring integrality of 55 variables
MINOS 5.5: optimal solution found.
39 iterations, objective 14.82842712
ampl: display Travel;
Travel :=
A B   1
A D   1
B C   1
C E   1
D F   1
E G   1
F K   1
G H   1
H J   1
I J   1
I K   1 ;
```

Networks

Solution *(optimal)*



. . . total distance = 14.828

Very Important Modeling Language Features

Recognizing other types of models

- Complementarity problems
- General combinatorial problems ***(see Thursday's session)***
- Semidefinite programs ***(to come)***
- Stochastic programs ***(to come, but harder)***

Exchanging information with solvers

- Solver-specific directives
- Solver-specific results & diagnostic information

Programming iterative schemes

- Loops over sets
- `if-then-else` tests
- Switching between subproblems

Very Important Features *(cont'd)*

Communicating with more solvers

Open solver interfaces ***(as in AMPL)***
Internet solver services
Solver input/output standards ***(to come)***

Communicating with other software

Database access
Callable interfaces to modeling systems ***(see Thursday's session)***

Features

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, . . .

Complementarity

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

Complementarity

Mixed 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 just now beginning to emerge

Features

Iterative Schemes

Flow of control

Looping
If-then-else

Named subproblems

Defining subproblems
Switching subproblems

Debugging

Expanding constraints
Single-stepping

Iterative Schemes

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

Named 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;
```


Iterative Schemes

Defining 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

Features

Internet Solver Services

Optimization Modeling-Language Servers

Single-langauge servers

General server projects related to optimization

NEOS Server

Client-Server Alternatives

General-purpose clients

Clients specially for NEOS

Callable NEOS

Kestrel / AMPL, Kestrel / GAMS

GAMS / AMPL via Kestrel

Metacomputing clients

MW, iMW

Condor / AMPL, Condor / GAMS

Kestrel / AMPL, Kestrel / GAMS

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 modeing languages: AMPL, GAMS, MP-MODEL

Varied submission options

- E-mail
- Web forms
- TCP/IP socket-based submission tool: Java or tcl/tk

. . . handled 2785 submissions last week (16 Sept 2002)

. . . can accept submissions of new solvers, too

Internet

NEOS Solver Listing

Internet

Callable NEOS

General mechanism

CORBA-based connection
between local program and NEOS Server

API under development
to support varied specialized applications

Kestrel: ***An application to AMPL*** *(also GAMS)*

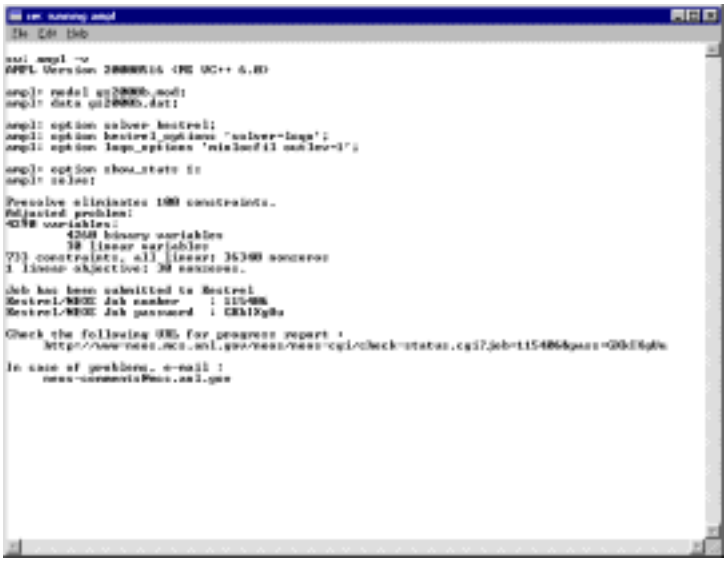
`www-neos.mcs.anl.gov/neos/kestrel.html`
`www.ampl.com/ampl/REMOTE/`
`www.gams.com/contrib/kestrel.htm`

Installs as a new "solver" that provides a NEOS gateway

Combines convenient local modeling environment
with access to remote solvers

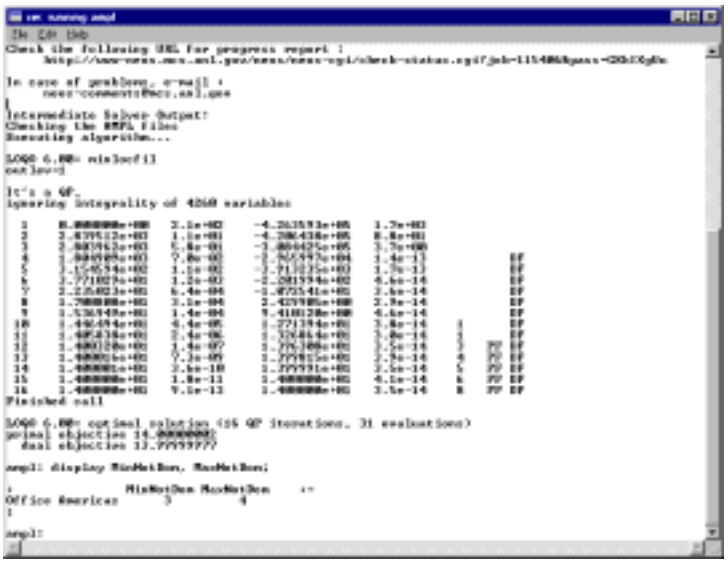
Internet

Kestrel Example



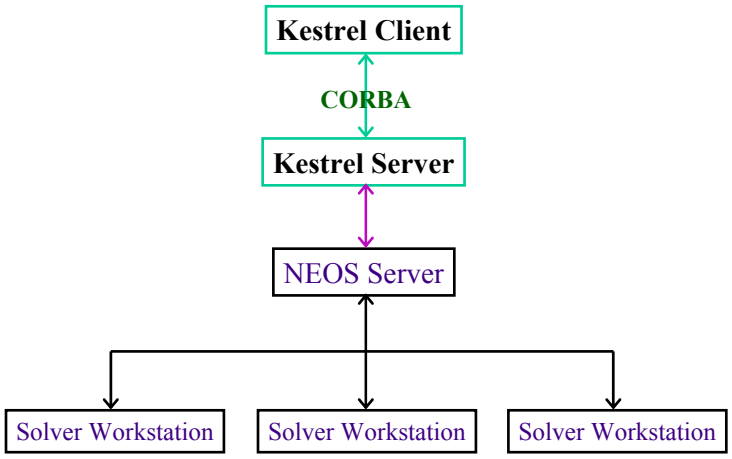
Internet

Kestrel Example *(cont'd)*



Internet

Outline of Kestrel Operations



Robert Fourer, IMA Tutorials, 9 Sept 2002: AMPL "Hands-On" Session

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

Robert Fourer, IMA Tutorials, 9 Sept 2002: AMPL "Hands-On" Session

Internet

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];
   ...
};
```


Features

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

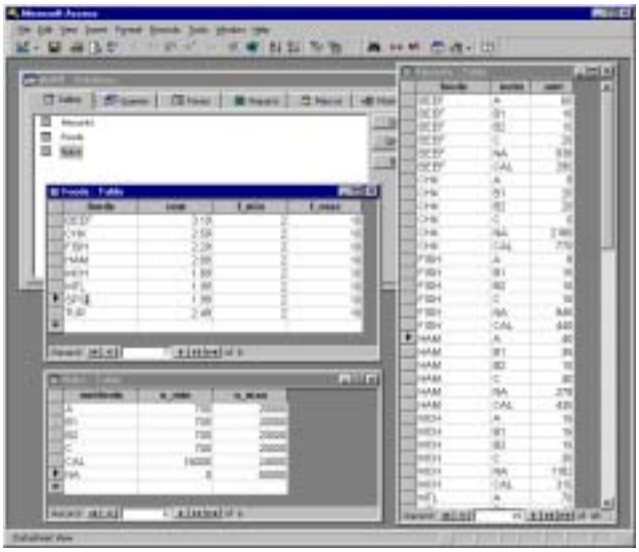
Database

Example: Diet Model

```
set NUTR;
set FOOD;

param cost {FOOD} > 0;
param f_min {FOOD} >= 0;
param f_max {j in FOOD} >= f_min[j];

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)*



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)*