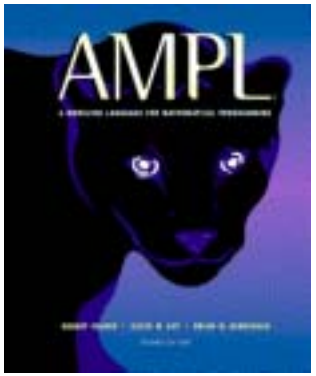
AMPL

New Features for Formulating Optimization Models



Robert Fourer

AMPL Optimization LLC, www.ampl.com

Department of Industrial Engineering & Management Sciences,
Northwestern University, Evanston, IL 60208-3119, USA

Workshop on Optimization Services in Europe

Gesellschaft für Operations Research e.V.
Physikzentrum, Bad Honnef, Germany, 14-15 October 2004

Development History

Research projects since 1985

Bell Laboratories Computing Sciences Research Center,
David Gay and *Brian Kernighan*

NU IE & MS Department,
National Science Foundation grants,
Robert Fourer

. . . all code after 1987 written by Gay

Lucent Technologies divestiture 1996

Lucent retains Bell Laboratories

Bell Laboratories retains AMPL

Commercialization History

Sold by licensed vendors since 1992

CPLEX Optimization, subsequently ILOG/CPLEX

4-6 much smaller companies, *including in Europe:*

- MOSEK (Denmark)
- OptiRisk Systems (UK)

AMPL Optimization LLC formed 2002

Lucent assigns

vendor agreements, trademark, web domain

Lucent retains

ownership of AMPL and gets a small royalty

. . . two years to negotiate!

Commercialization History *(cont'd)*

Current members of LLC

Fourer, professor at Northwestern

Kernighan, professor at Princeton

Gay, researcher at Sandia National Laboratory

Current situation

Sandia licenses the AMPL source code

. . . another year to negotiate!

AMPL Optimization LLC is gradually arranging to
sell solvers, provide marketing and maintenance

Marketing Strategy

Goals

Clearest *and* most powerful language

Tutorial but comprehensive textbook

Broad base of satisfied users and consultants

Automated benchmarking services

Moderate price

Advantages

Marketing and support can be decentralized

New AMPL company can be expanded gradually

Disadvantages

Not much known about the user base

Development of new features can be hard to coordinate

Market Position

Competition from . . .

Other modeling languages & systems

(AIMMS, MPL, GAMS, LPL)

Proprietary systems of established solver vendors

(ILOG/OPL Studio, Dash/MOSEL, LINGO)

Other software used as a modeling system

(Excel/Frontline, MATLAB/Tomlab)

Outline

Example

model, data, solution

Complementarity problems

Stochastic programming

Combinatorial optimization

The NEOS Server

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 Gate {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
```

Airline Fleet Assignment Data

```
set FLEETS := 72S 73S L10 ;
set CITIES := ATL CVG DFW ;

set TIMES := 1200a 1210a 1220a 1230a 1240a 1250a
             100a  110a  120a  130a  140a  150a
             200a  210a  220a  230a  240a  250a
             300a  310a  320a  330a  340a  350a

set FLEET_LEGS :=
        (72S,ATL,*,CVG,*)  630a  740a   830a  950a  1210p  130p
        (72S,ATL,*,CVG,*)  120p  240p   430p  600p   640p  810p
        (72S,ATL,*,CVG,*)  850p 1010p  1150p  100a
        (73S,ATL,*,CVG,*)  630a  740a   830a  950a  1210p  130p

param leg_cost :=
        [72S,ATL,*,CVG,*]  630a  740a 33   830a  950a 33  1210p  130p 33
        [72S,ATL,*,CVG,*]  120p  240p 33   430p  600p 33   640p  810p 33
        [72S,ATL,*,CVG,*]  850p 1010p 33  1150p  100a 33
        [73S,ATL,*,CVG,*]  630a  740a 30   830a  950a 30  1210p  130p 30

param fleet_size :=  72S 6  73S 6  L10 2 ;
```

Airline Fleet Assignment Solution

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

ampl: option solver kestrel;
ampl: option kestrel_options 'solver pcx';

ampl: option show_stats 1;

ampl: solve;

327 variables, all linear
258 constraints; 790 nonzeros
        211 linear network constraints
        47 general linear constraints
1 linear objective; 116 nonzeros.

Job has been submitted to Kestrel
Kestrel/NEOS Job number    : 458598
Kestrel/NEOS Job password  : lggrLQxk

Check the following URL for progress report :
     http://www-neos.mcs.anl.gov/neos/neos-cgi/
          check-status.cgi?job=458598&pass=lggrLQxk

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

Intermediate Solver Output: ...
```

Airline Fleet Assignment Solution *(cont'd)*

```
Executing algorithm...

Before Scaling: ScaleFactor = 0.0
Cholesky factor will have density  0.11448
  FOUND      5 TINY DIAGONALS; REPLACED WITH INF

Maximum Gondzio corrections = 0

Iter    Primal       Dual      (PriInf  DualInf)  log(mu) dgts   Merit
  0   1.0426e+04   1.7577e+03  (3.1e+00 8.9e-02)    0.77    0   1.8e+01
  FOUND      3 TINY DIAGONALS; REPLACED WITH INF
  1   6.0005e+03   1.8285e+03  (1.5e+00 1.5e-02)    0.30    0   8.7e+00
  FOUND      4 TINY DIAGONALS; REPLACED WITH INF
  2   2.4219e+03   1.9639e+03  (1.3e-01 8.4e-04)   -0.49    0   9.3e-01
  FOUND      3 TINY DIAGONALS; REPLACED WITH INF
  3   2.1674e+03   2.0302e+03  (4.2e-02 1.5e-04)   -1.12    1   2.8e-01
  FOUND      2 TINY DIAGONALS; REPLACED WITH INF
  4   2.0584e+03   2.0393e+03  (4.5e-03 3.9e-05)   -1.73    2   3.8e-02
  FOUND      4 TINY DIAGONALS; REPLACED WITH INF
  5   2.0442e+03   2.0439e+03  (5.5e-05 3.7e-07)   -3.66    3   4.6e-04
  FOUND      1 TINY DIAGONALS; REPLACED WITH INF
  6   2.0440e+03   2.0440e+03  (1.7e-10 8.5e-13)   -9.23    9   1.4e-09

--termination with OPTIMAL status

Finished call

Optimal solution found.
```

Airline Fleet Assignment Solution *(cont'd)*

```
ampl: option display_eps .00001, omit_zero_rows 1, display_1col 100000;

ampl: display {f in FLEETS}:
ampl?    {(f,c1,t1,c2,t2) in FLEET_LEGS} Fly[f,c1,t1,c2,t2];

Fly['72S',c1,t1,c2,t2] :=
CVG 110p  DFW 220p    1
CVG 640p  DFW 800p    1
CVG 850a  DFW 1010a   1
DFW 1050a CVG 200p    1
DFW 440p  CVG 800p    1
DFW 820p  CVG 1140p   1
;

Fly['73S',c1,t1,c2,t2] :=
ATL 1010a DFW 1110a   1
ATL 1010p DFW 1120p   1
ATL 1140p DFW 1250a   1
ATL 1150p CVG 100a    1
ATL 120p  CVG 240p    1
ATL 120p  DFW 230p    1
ATL 1210p CVG 130p    1
ATL 430p  CVG 600p    1
ATL 630a  CVG 740a    1   ...
```

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

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-variable *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

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)***

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 continuing to emerge

Stochastic Programs

Extensions within AMPL *(proposed)*

Allow random distributions for some problem data

Make distributions available to solvers

Extensions using AMPL *(substantially implemented)*

Add special expressions and conventions for stages & scenario trees

Compile to standard AMPL

Generate problem descriptions for various solvers

- SAMPL
- StAMPL

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

AMPL SP Extensions

SAMPL

Patrick Valente, Gautam Mitra, Mustapha Sadki

Brunel University, Uxbridge, Middlesex, UK

- P. Valente, G. Mitra, M. Sadki and R. Fourer, "Extending Algebraic Modelling Languages for Stochastic Programming" (2004).

Stages

Two-stage recourse model

```
suffix stage IN;
var x {Prod, Fact, t in Time, Scen} >=0,
                          suffix stage if t = 1 then 1 else 2;
var y {Prod, Fact, t in Time, Scen} >=0,
                          suffix stage if t = 1 then 1 else 2;
.......
```

Multi-stage recourse model

```
suffix stage IN;
var x {Prod, Fact, t in Time, Scen} >=0, suffix stage t;
var y {Prod, Fact, t in Time, Scen} >=0, suffix stage t;
.......
```

Scenarios

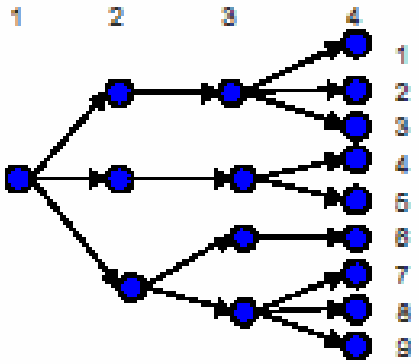
Scenario set

```
param NS > 1;
scenarioset Sc = 1..NS;
```

Scenario probabilities

```
probability param Pr {Sc} = 1 / card(Sc);   # uniform case
```

Scenario Trees: General Form



Bundle form

```
tree FourStageExample :=

bundles {

   (1,1),
   (2,1),(2,4),(2,6),
   (3,1),(3,4),(3,6),(3,7),
   (4,1),(4,2),(4,3),(4,4),(4,5),(4,6),(4,7)(4,8),(4,9)

};
```

Treelist form (if scenario paths never cross)

```
tree FourStageEXample := tlist {1,4,4,2,4,2,3,4,4};
```

Scenario Trees: Standard Forms

Uniform branching at each stage

```
multibranch (n1,n2,..., nST);
```

Uniform branching at all stages

```
nway (n);
```

Binary at all stages

```
binary;
```

Two-stage

```
twostage;
```

Random Parameters

Declaration

```
random param d {Time,Scen} >=0;  # demand
```

Compact data

```
random param dem :=
   1 1  10.0
   2 1   5.0
   2 3  15.0
   3 1   2.5
   3 2   7.5
   3 3   7.5
   3 4  22.5 ;
```

Expanded data

```
random param dem (tr):=
        1    2     3
   1   10    5   2.5
   2   10    5   7.5
   3   10   15   7.5
   4   10   15  22.5 ;
```

Chance Constraints

Random parameter in constraint

```
random param d {Prod,Deal,Time,Scen} >= 0  # demand
subj to satisfy_demand {j in Prod, k in Deal, t in Time, s in Scen}:
   sum {i in Fact} z[j,i,k,t,s] = d[j,k,t,s];
```

Conversion to a chance constraint

```
param beta := 0.9;
chance {j in Prod, k in Deal, t in Time,s in Scen}
   satisfy_demand[j,k,t,s] >= beta;
```

Further Issues

Communication with solvers

Need a standard form for communicating
stochastic programming instances to solvers

Existing "SMPS" form is outdated and inadequate
(and not entirely standard)

Communication with scenario generators

Independent generators:
consistency with model must be ensured somehow

Integrated generator:
modeling language calls generator as needed

Integration in a modeling environment

SPInE: scenario generation, modeling, solving, results analysis

StAMPL

Leo Lopes

PhD, Northwestern University;
Assistant Professor, University of Arizona

- R. Fourer and L. Lopes, "StAMPL: A Filtration-Oriented Modeling Tool for Stochastic Programming" (2003).

Main Features

Non-technical

- Fewer indexes
- Modular
- Fewer conditions

Technical

- No non-anticipativity constraints
- Recourse and technology matrices are apparent at first inspection

```
definestage 1;

set INSTR;
var Buy{INSTR} >= 0;

param initial_wealth;

subject to InvestAll:
     sum{i in INSTR} Buy[i] = initial_wealth;

#####################################################
definestage 2..(stages()-1);

set INSTR;
var Buy{INSTR} >= 0;

param return{INSTR};

subject to ReinvestAll:
     sum{i in INSTR} parent().Buy[i]*return[i] =
     sum{i in INSTR} Buy[i];

#####################################################
definestage stages();

set INSTR;
var Shortage >= 0;
var Overage >= 0;

param shortage_penalty;
param overage_reward;
     check: shortage_penalty > overage_reward;
param return{INSTR};
param goal;

maximize Final_Wealth:
     overage_reward*Overage -
shortage_penalty*Shortage;

subject to ReinvestAll:
     sum{i in INSTR} parent().Buy[i]*return[i]
     + Shortage - Overage = goal;
```

Syntax: Stage Definition

definestage *statement*

Similar to the AMPL **`problem`** statement

```
definestage 1;

set INSTR;
var Buy{INSTR} >= 0;

param initial_wealth;

subject to InvestAll:
     sum{i in INSTR} Buy[i] = initial_wealth;

##################################################
definestage 2..(stages()-1);

set INSTR;
var Buy{INSTR} >= 0;

param return{INSTR};

subject to ReinvestAll:
     sum{i in INSTR} parent().Buy[i]*return[i] =
     sum{i in INSTR} Buy[i];

##################################################
definestage stages();

set INSTR;
var Shortage >= 0;
var Overage >= 0;

param shortage_penalty;
param overage_reward;
     check: shortage_penalty > overage_reward;
param return{INSTR};
param goal;

maximize Final_Wealth:
     overage_reward*Overage -
shortage_penalty*Shortage;

subject to ReinvestAll:
     sum{i in INSTR} parent().Buy[i]*return[i]
     + Shortage - Overage = goal;
```

Syntax: Connecting Problems

`parent()` *function*

Returns a model object

All components of the parent model can be accessed using the `'.'` (dot) operator

```
definestage 1;

set INSTR;
var Buy{INSTR} >= 0;

param initial_wealth;

subject to InvestAll:
     sum{i in INSTR} Buy[i] = initial_wealth;

##################################################
definestage 2..(stages()-1);

set INSTR;
var Buy{INSTR} >= 0;

param return{INSTR};

subject to ReinvestAll:
     sum{i in INSTR} parent().Buy[i]*return[i] =
     sum{i in INSTR} Buy[i];

##################################################
definestage stages();

set INSTR;
var Shortage >= 0;
var Overage >= 0;

param shortage_penalty;
param overage_reward;
     check: shortage_penalty > overage_reward;
param return{INSTR};
param goal;

maximize Final_Wealth:
     overage_reward*Overage -
shortage_penalty*Shortage;

subject to ReinvestAll:
     sum{i in INSTR} parent().Buy[i]*return[i]
     + Shortage - Overage = goal;
```

Syntax: Stage Information

`stage()` *and* **`stages()`** *functions*

Return the number of the current stage and the total stages

Uses:

- Defining stages
- Discounting
- Multi-period
- constraints

```
definestage 1;

set INSTR;
var Buy{INSTR} >= 0;

param initial_wealth;

subject to InvestAll:
     sum{i in INSTR} Buy[i] = initial_wealth;

########################################################
definestage 2..(stages()-1);

set INSTR;
var Buy{INSTR} >= 0;

param return{INSTR};

subject to ReinvestAll:
     sum{i in INSTR} parent().Buy[i]*return[i] =
     sum{i in INSTR} Buy[i];

########################################################
definestage stages();

set INSTR;
var Shortage >= 0;
var Overage >= 0;

param shortage_penalty;
param overage_reward;
     check: shortage_penalty > overage_reward;
param return{INSTR};
param goal;

maximize Final_Wealth:
     overage_reward*Overage -
shortage_penalty*Shortage;

subject to ReinvestAll:
     sum{i in INSTR} parent().Buy[i]*return[i]
     + Shortage - Overage = goal;
```

Scenario Trees

Issues

Trees don't fit well with AML paradigms

Generating scenario trees is very specialized

Usually involves developing special routines

Often demands specialized software

Conclusion

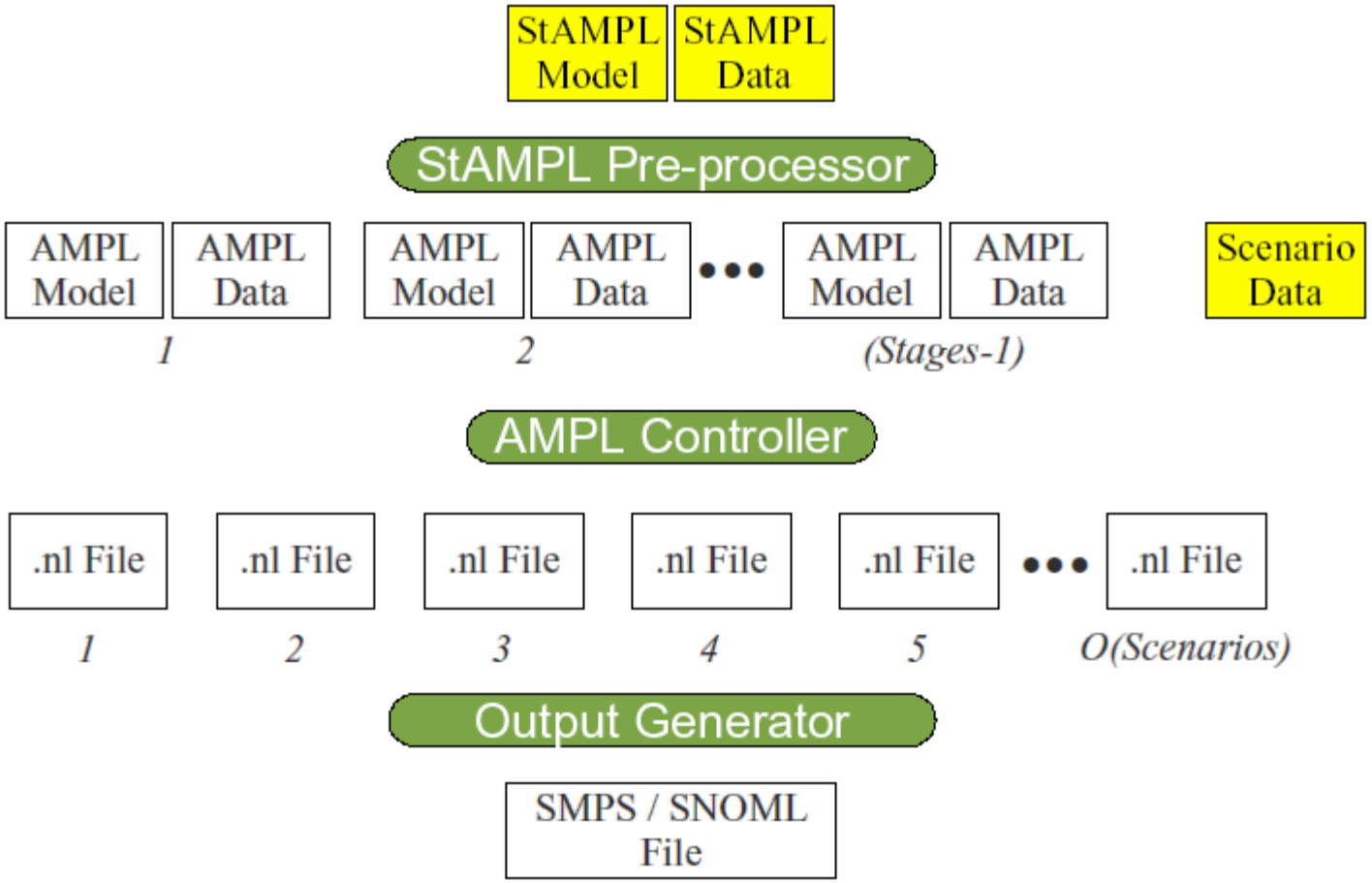
Write a general-purpose language library (C++)

- *The library generates an intermediary file*

Export the library to each necessary environment using

- *SWIG*
- *XML-RPC, .NET*
- *System-specific libraries*

How It Works


StAMPL Model
StAMPL Data
StAMPL Pre-processor
AMPL Model
AMPL Data
AMPL Model
AMPL Data
AMPL Model
AMPL Data
Scenario Data
1
2
(Stages-1)
AMPL Controller
.nl File
.nl File
.nl File
.nl File
.nl File
.nl File
1
2
3
4
5
O(Scenarios)
Output Generator
SMPS / SNOML File

Combinatorial Optimization

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

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

```
duePen[j] * (dueTime[j] - 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

```
setupCost[i,j] * 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

```
setupCost[JobForSlot[k],JobForSlot[k+1]]
```

Example: Production Constraints

Integer programming formulation

For each job **i**, **ComplTime[i] ≤ dueTime[i]**

For each job pair **(i,j)**,

```
ComplTime[i] + setupTime[i,j] + procTime[j] ≤
ComplTime[j] + BIG * (1 - Seq[i,j])
```

More natural formulation

For each slot **k**,

```
ComplTime[JobForSlot[k]] = min (
  dueTime[JobForSlot[k]],
  ComplTime[JobForSlot[k+1]]
    - procTime[JobForSlot[k+1]]
    - setupTime[JobForSlot[k],JobForSlot[k+1]] )
```

Example: Sequencing Constraints

Integer programming formulation

For each job **i**,

```
sum {j in JOBS} Seq[i,j] = 1
```

For each job **i**,

```
sum {j in JOBS} Seq[j,i] = 1
```

More natural formulation

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

Representing "Range" Constraints

General format

lower-bound ≤ *linear-expr* + *nonlinear-expr* ≤ *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

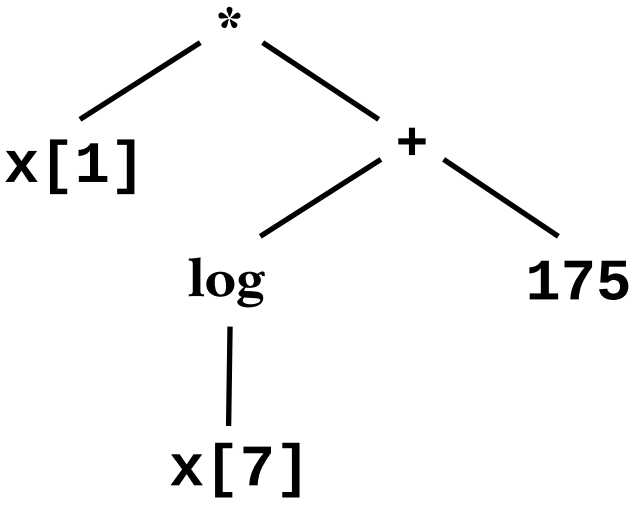
Binary, unary operators

Iterated summation, min, max

Piecewise-linear terms

If-then-else terms

. . . single array of variables



Combinatorial

"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]);
}
```

Combinatorial

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

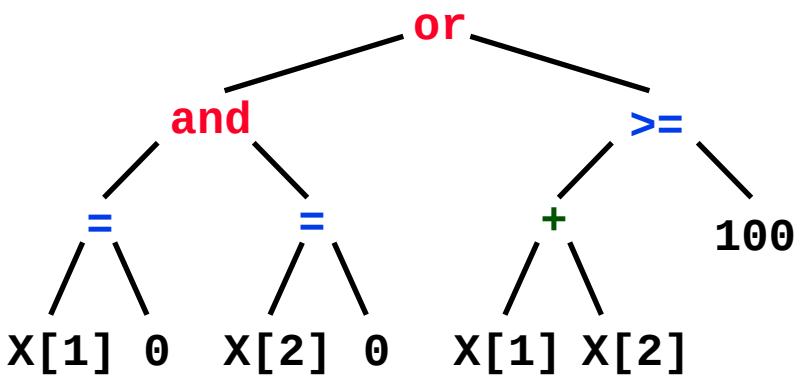
```
switch(opnum) {

 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];

 .......

}
```

Combinatorial

Logical Constraints

Simple forms

constraint **and** *constraint*
constraint **or** *constraint*
not *constraint*




```
(X[1] = 0 and X[2] = 0) or X[1] + X[2] >= 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 Logical Constraint 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)

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

A general-purpose optimization server

- Over 45 solvers in all
  - Linear, linear network, linear integer
  - Nonlinear, nonlinear integer, nondifferentiable & global
  - Stochastic, semidefinite, semi-infinite, complementarity
- Commercial as well as experimental solvers
- Central scheduler with distributed solver sites

A research project

- Currently free of charge
- Supported through the Optimization Technology Center of Northwestern University & Argonne National Laboratory

. . . 4677 submissions last week
. . . as many as 6094 submissions in a week

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

Varied submission options

E-mail – **Web forms** – **Direct function call**

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

. . . server processes submissions of new solvers, too

Outline

Server basics

- Use through a web browser
- ***Calls from within a modeling environment: Kestrel***
- ***Benchmarking interface***

Frequently asked questions

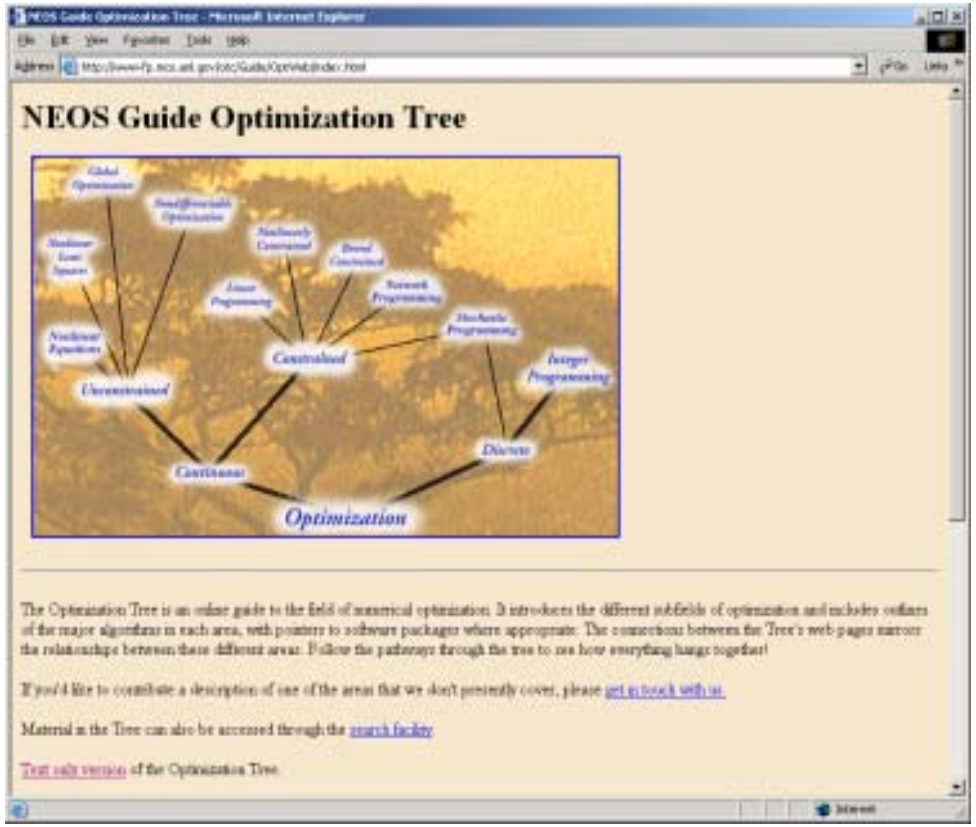
- Who uses it? Where from? How much?
- What solvers are there?
  - Who supplies them? Who hosts them?
  - Which are most used?
- How is it supported? Who answers user questions?

Recent and forthcoming developments . . .

Learn About Your Problem

The NEOS Guide

- Optimization tree: Problem types
- Optimization software guide: Individual solvers
- Frequently asked questions: Varied listings & advice

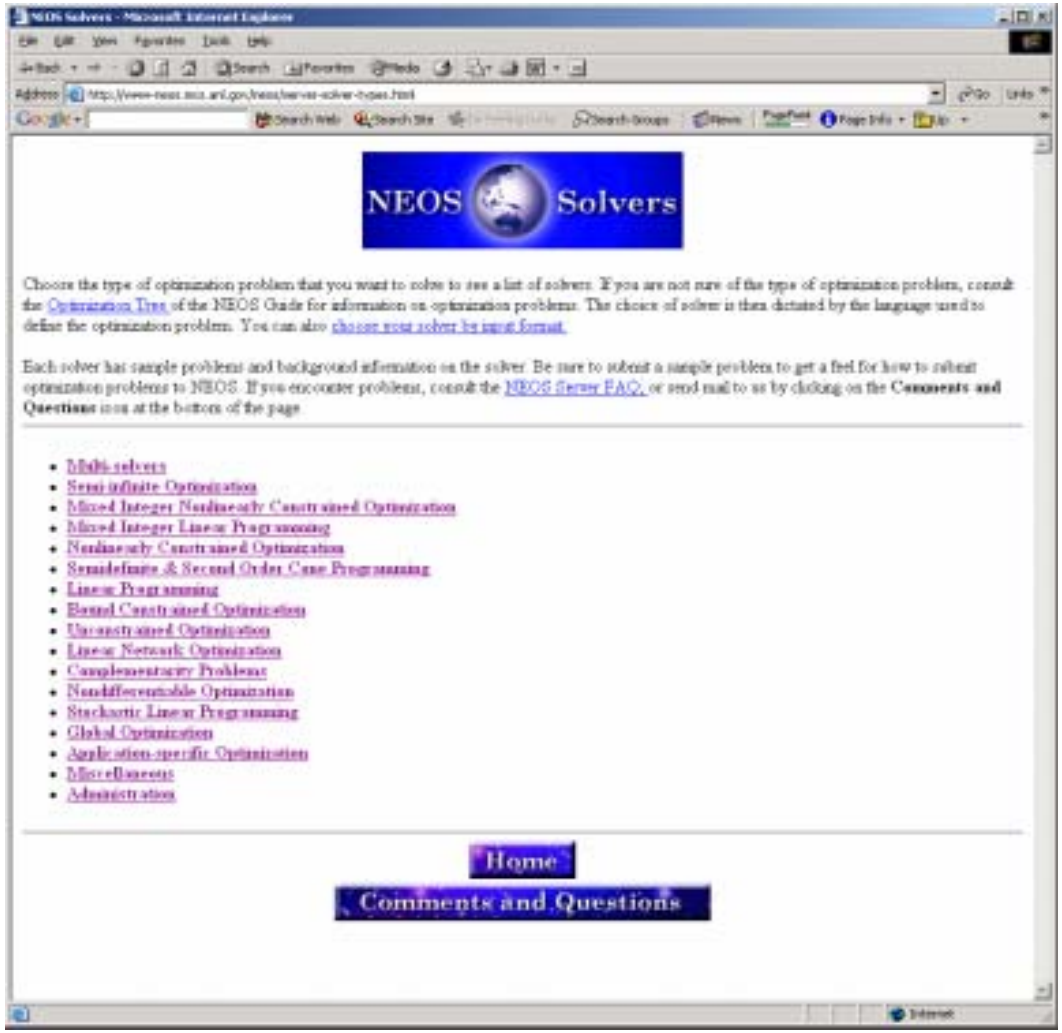


Investigate Solvers

NEOS Server home page

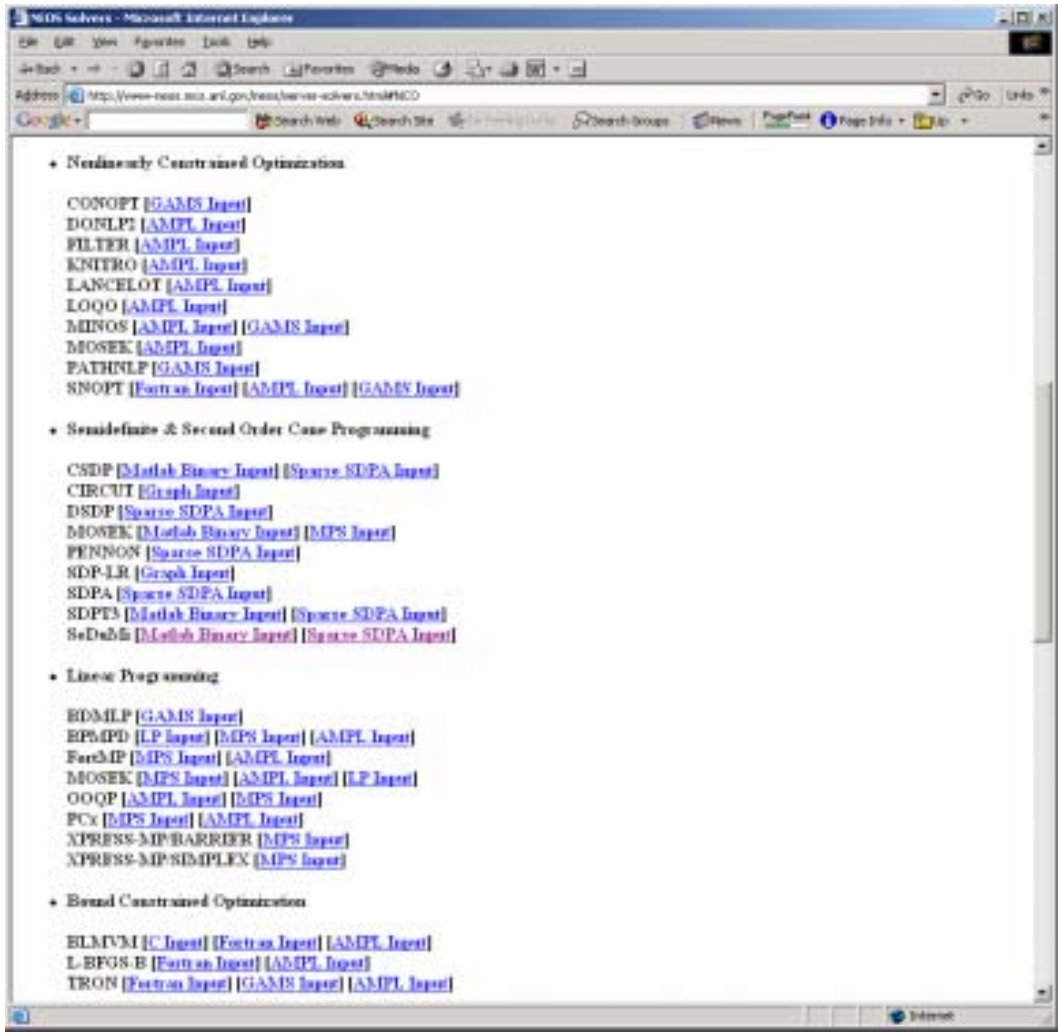
Investigate Solvers

NEOS Server solver type listing



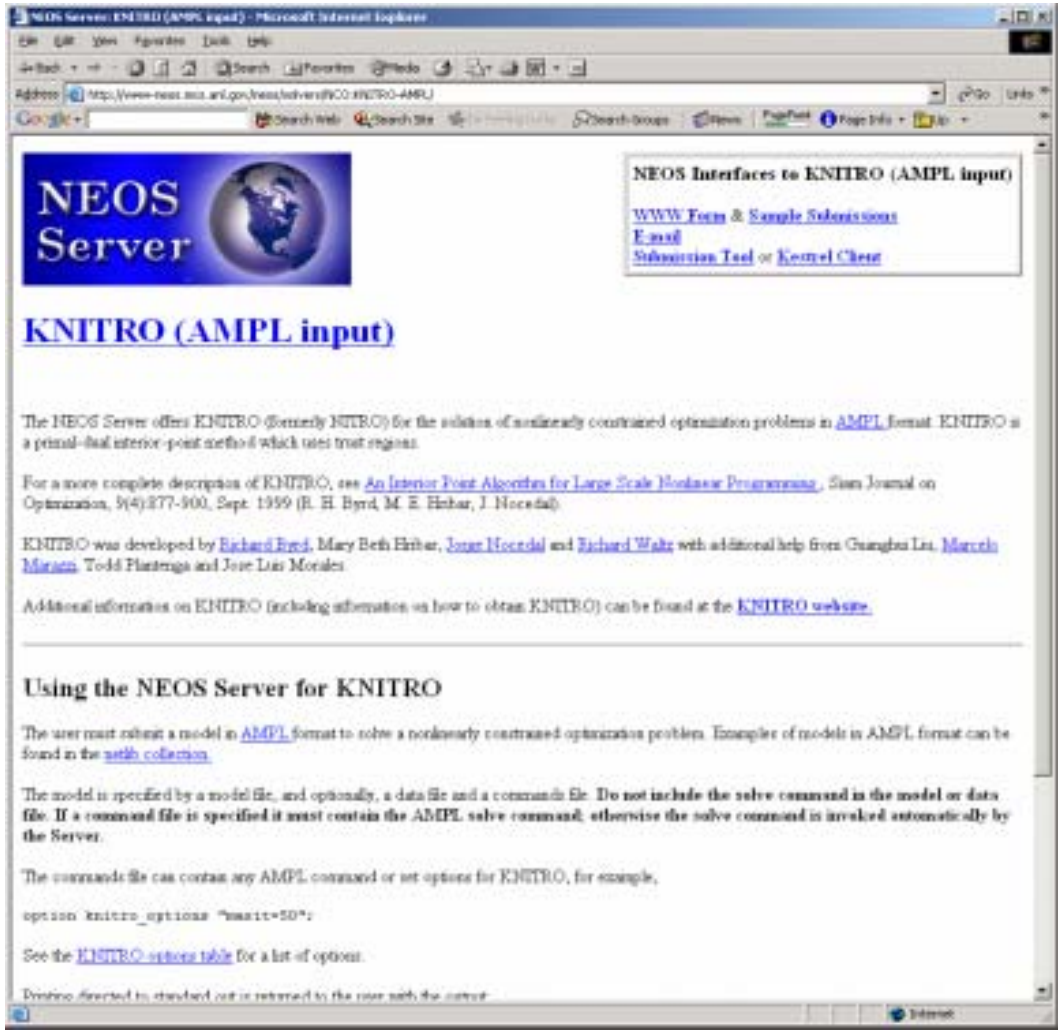
Investigate Solvers

NEOS Server solver listing



Investigate Solvers

Individual solver listing



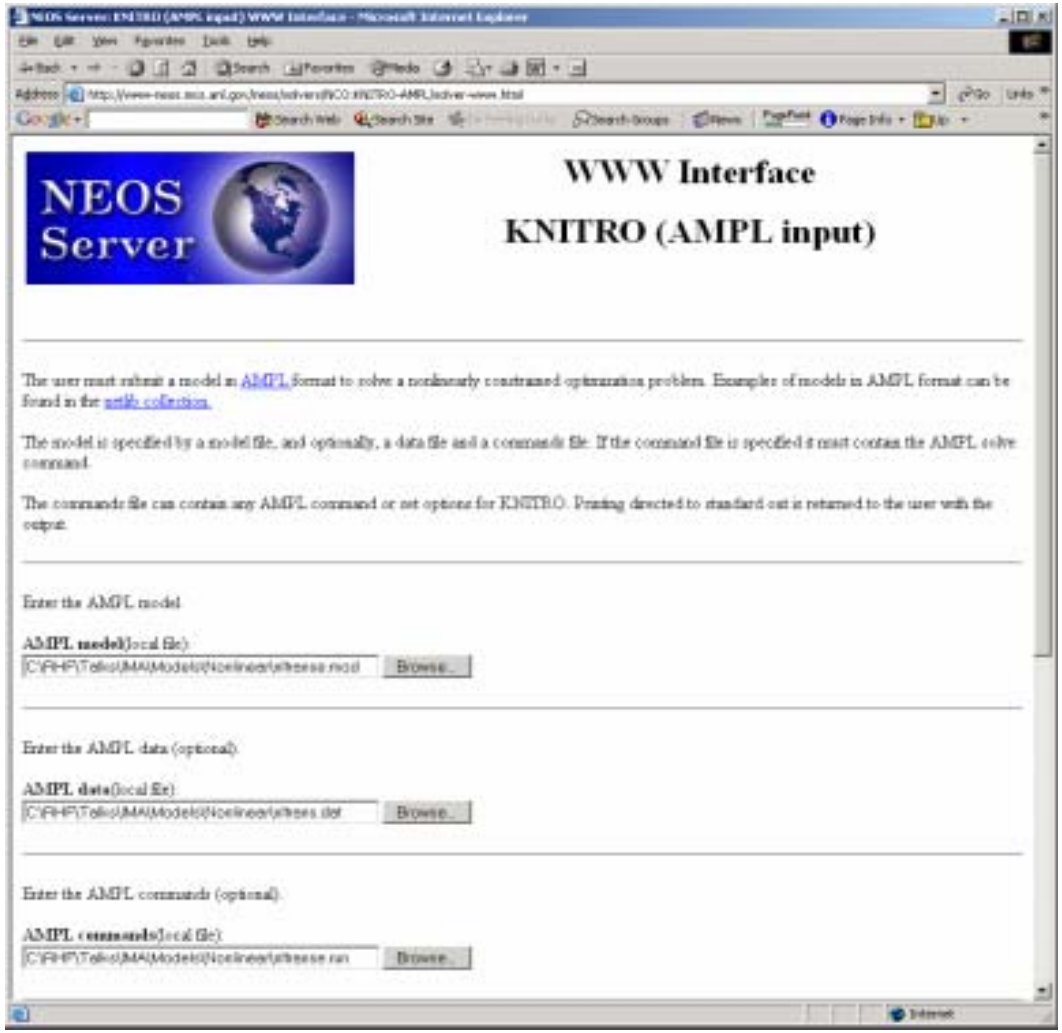
Try a Solver: Web Interface

Sample submission form

- "Comments and Questions" button on every page

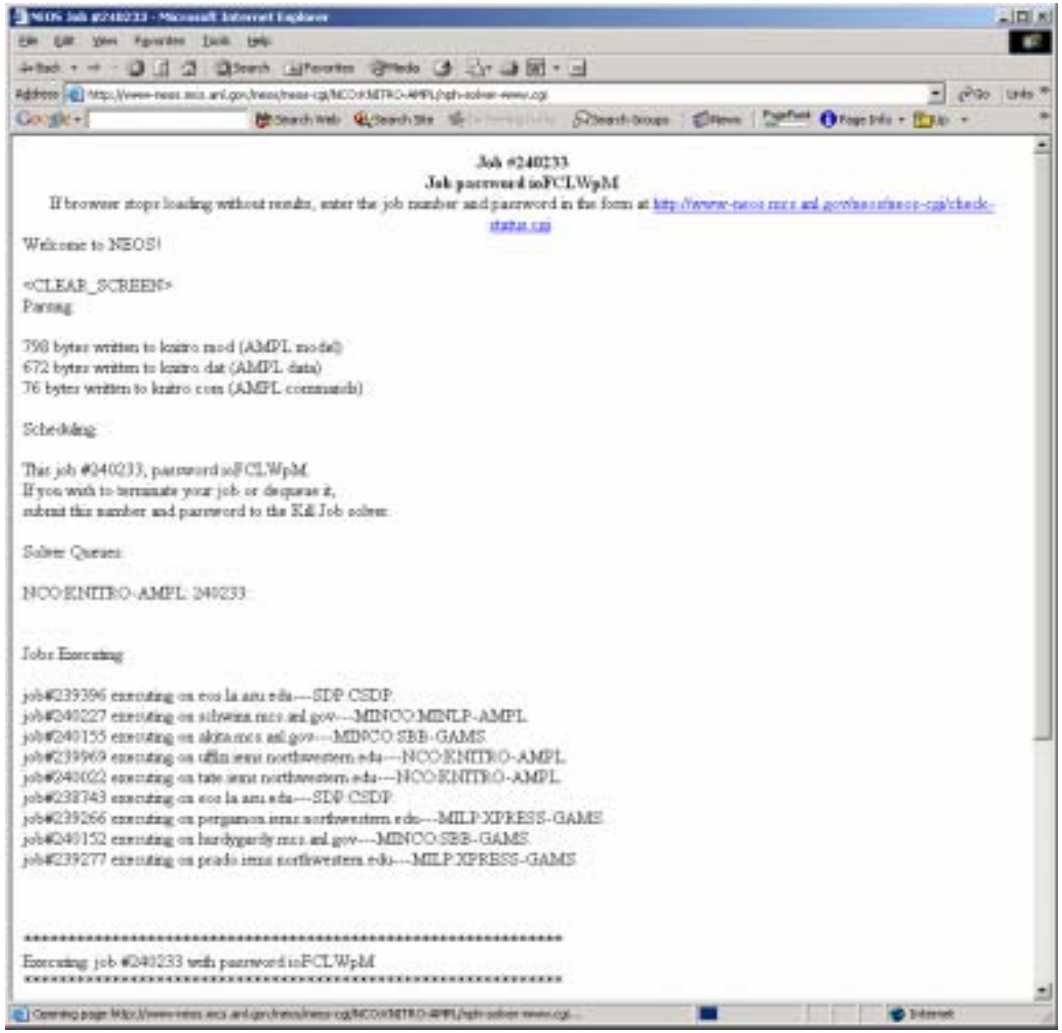
Web Interface

Submission form for your problem



Web Interface

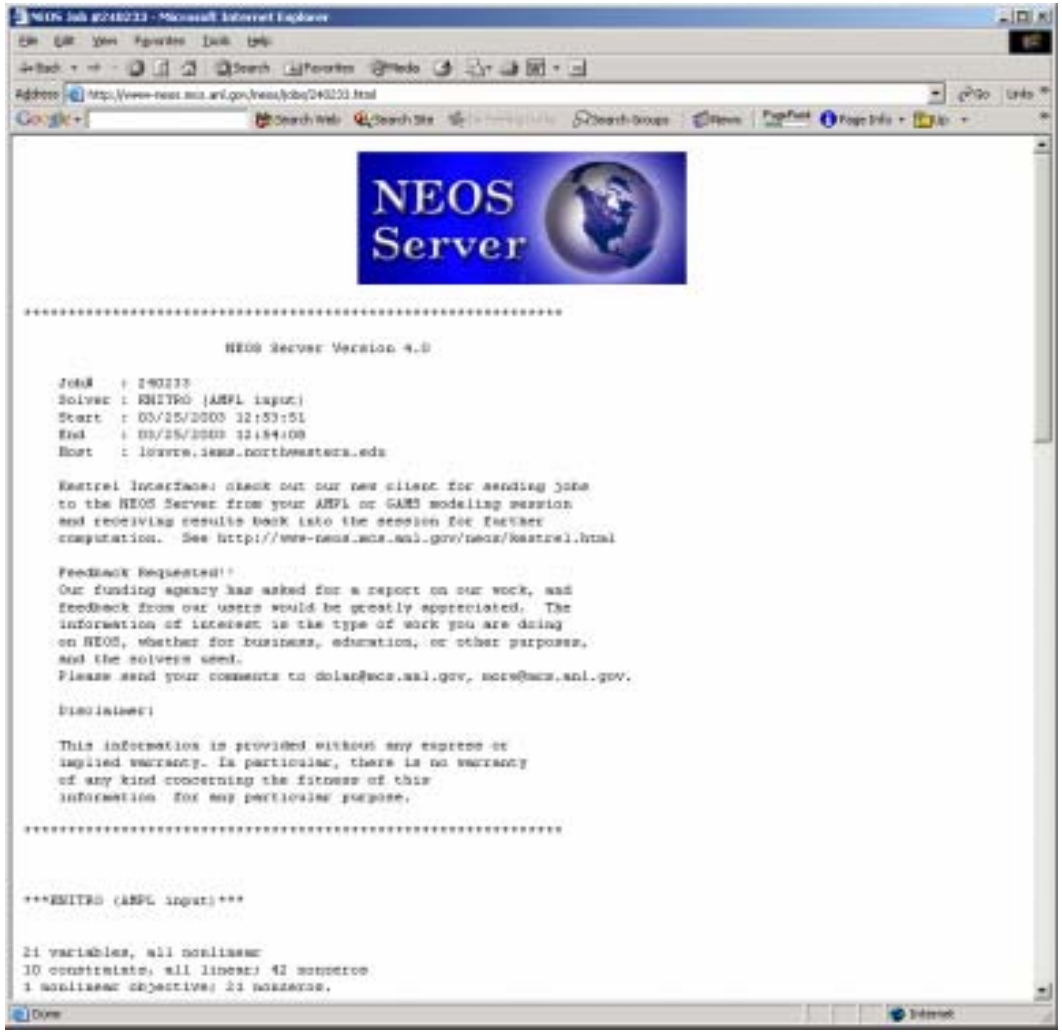
Start of your run



Using NEOS

Web Interface

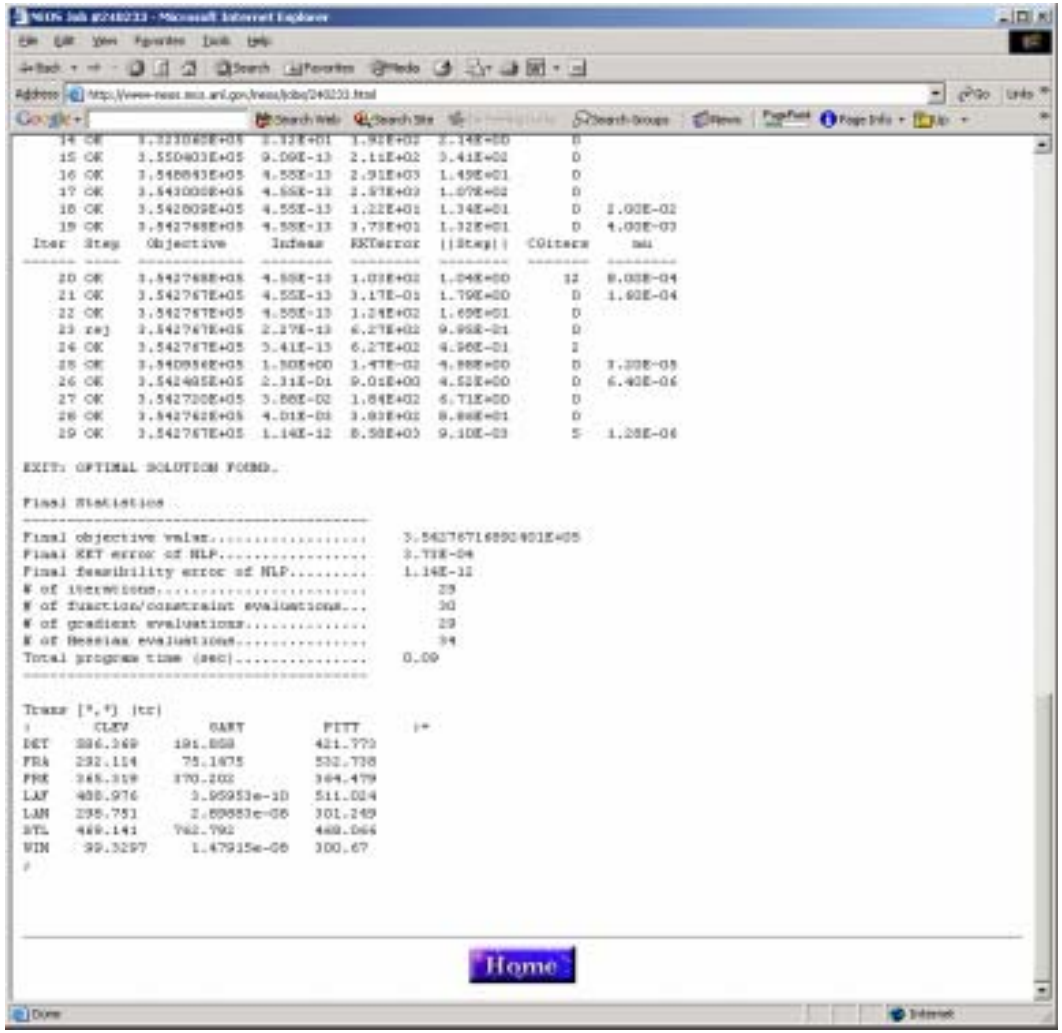
Beginning of your solution listing



Using NEOS

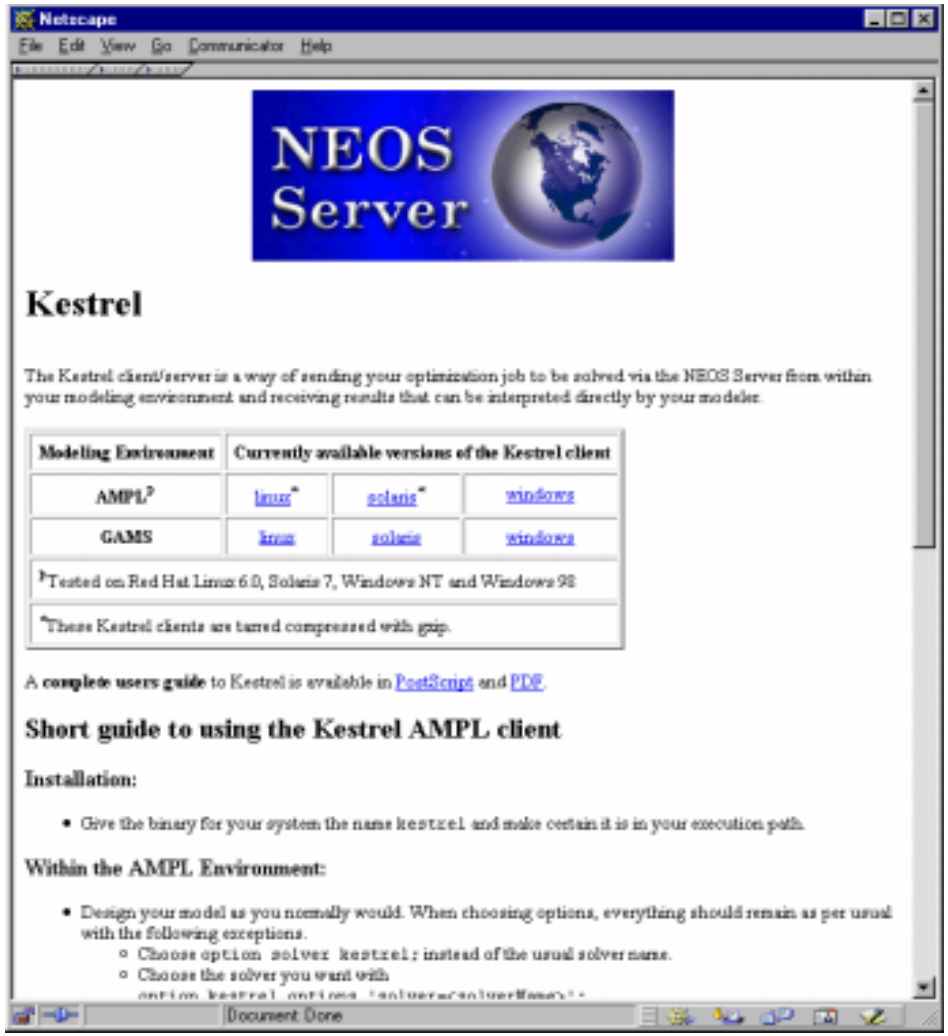
Web Interface

End of your solution listing



Try a Solver: Kestrel Interface

Kestrel client download page



Kestrel Interface

Applying a local solver to an AMPL model

```
AMPL Version 20040202 (MS VC++ 6.0)

ampl: model gs2000b.mod; data gs2000b.dat;
ampl: option solver minos;
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.

MINOS 5.5:

Sorry, the student edition is limited to 300 variables and
300 constraints.  You have 4290 variables and 733 constraints.

exit code 1
<BREAK>
```

Kestrel Interface

Applying a NEOS solver to an AMPL model . . .

```
ampl: option solver kestrel;
ampl: option kestrel_options 'solver=loqo';
ampl: option loqo_options 'minlocfil outlev=1';
ampl: solve;

Job has been submitted to Kestrel

Kestrel/NEOS Job number    : 368607
Kestrel/NEOS Job password  : OxBpVYMb

Check the following URL for progress report :

http://www-neos.mcs.anl.gov/neos/
neos-cgi/check-status.cgi?job=368607&pass=OxBpVYMb

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

Kestrel Interface

. . . and receiving a solution from NEOS . . .

```
LOQO 6.06: minlocfil

 1    0.000000e+00   2.1e-01   -4.266000e+05   3.1e+02
 2    2.840664e+03   9.8e-04   -4.206215e+05   1.5e+01
 3    2.796511e+03   4.8e-04   -3.078341e+05   6.2e-01
 4    1.769055e+03   4.2e-04   -2.948373e+04   0.0e+00            DF
 5    3.024840e+02   5.2e-05   -3.871922e+03   0.0e+00            DF
 6    3.705130e+01   5.9e-06   -2.158849e+02   0.0e+00            DF
 7    2.220340e+01   3.1e-06   -1.070050e+01   0.0e+00            DF
 8    1.685976e+01   1.4e-06    2.596295e+00   0.0e+00            DF
 9    1.534094e+01   6.7e-07    9.491761e+00   0.0e+00        PF  DF
10    1.445050e+01   2.1e-07    1.284805e+01   0.0e+00    1   PF  DF
11    1.405725e+01   8.3e-09    1.333832e+01   0.0e+00    1   PF  DF
12    1.400313e+01   4.9e-10    1.396657e+01   0.0e+00    3   PF  DF
13    1.400016e+01   2.5e-11    1.399833e+01   0.0e+00    4   PF  DF
14    1.400001e+01   1.2e-12    1.399992e+01   0.0e+00    5   PF  DF

LOQO 6.06: optimal solution (14 QP iterations, 14 evaluations)

primal objective 14.00000783
  dual objective 13.99991642
```

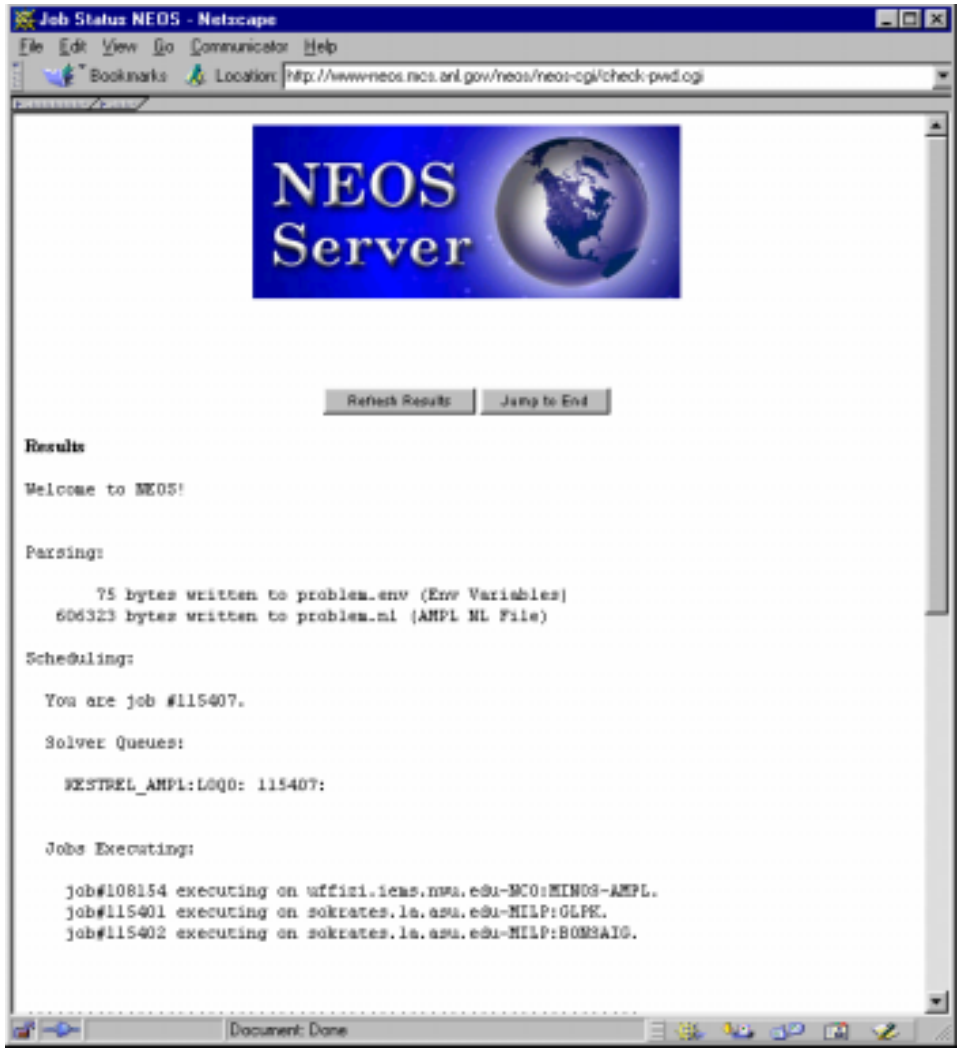
Kestrel Interface

. . . where it can be browsed interactively

```
ampl: option display_eps .000001;

ampl: display MinType, MaxType;
:                                                  MinType  MaxType :=
Division Shipping                                    0        1
Division Logistics_and_Supply_Chain                  1        2
Division Information_Technology                      1        1
Division Production                                  0        1
Division Production_Scheduling                       1        2
Division Production_Scheduling_Research              1        2
Division Operations_Management                       0        1
Division Finance                                     3        4
Division Support                                     0        1
Office   Americas                                    7        8
Office   EMEA                                        1        2
Office   Far_East                                    1        2
Gender   F                                           3        4
Gender   M                                           7        8
;
```

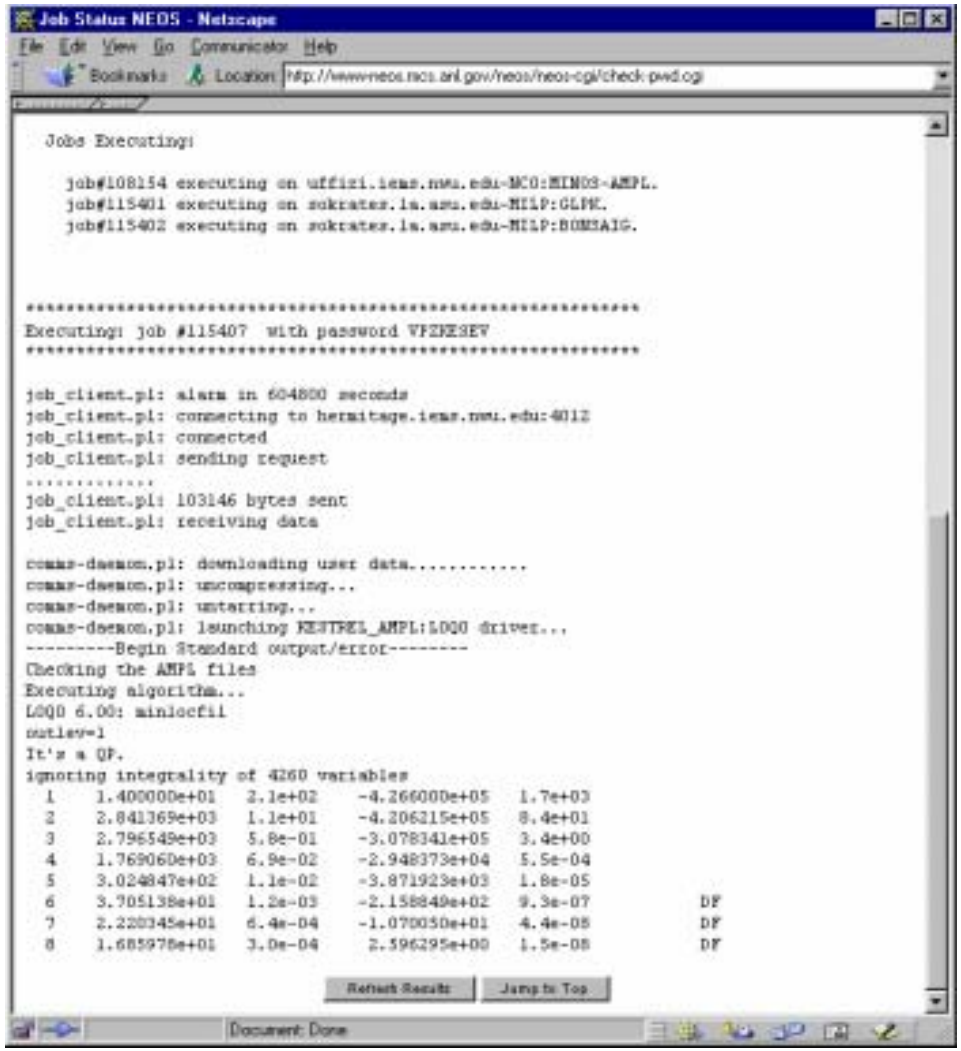
Kestrel Interface

Web form for checking your run's status



Kestrel Interface

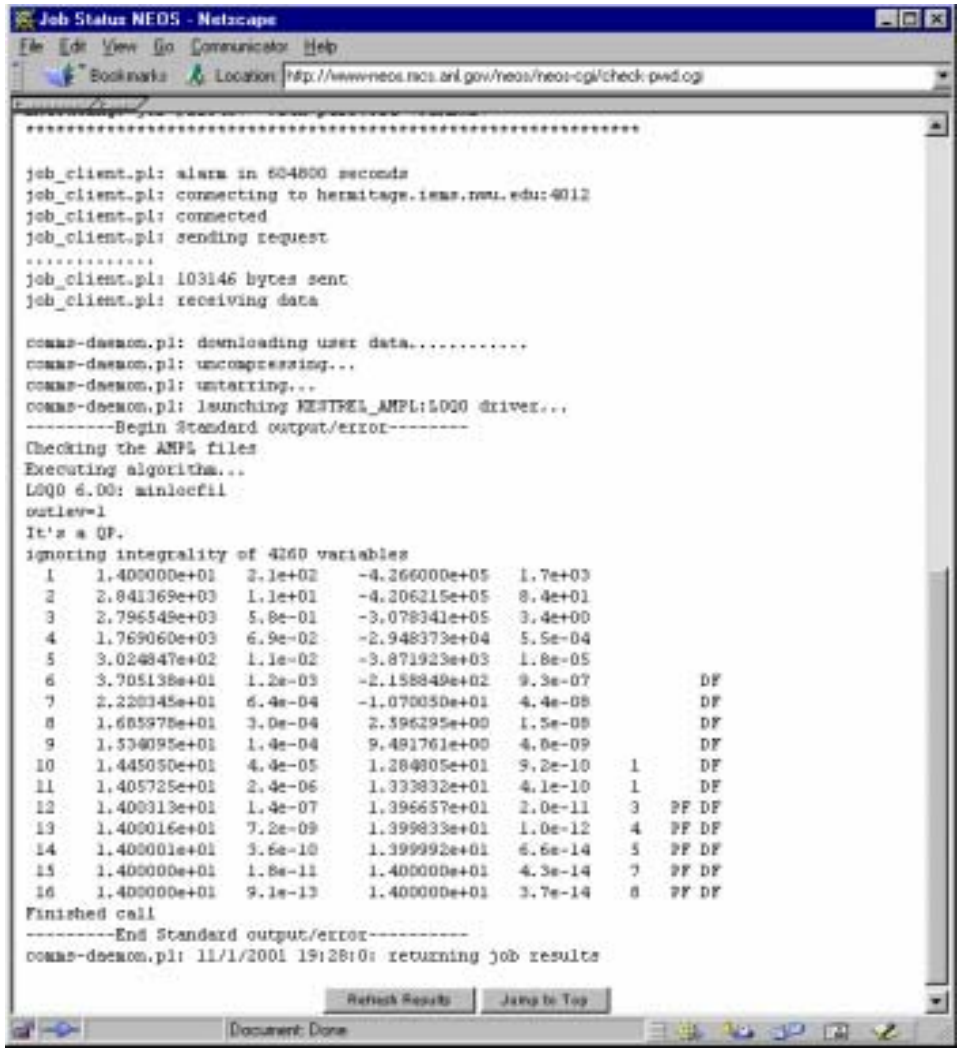
Intermediate status listing



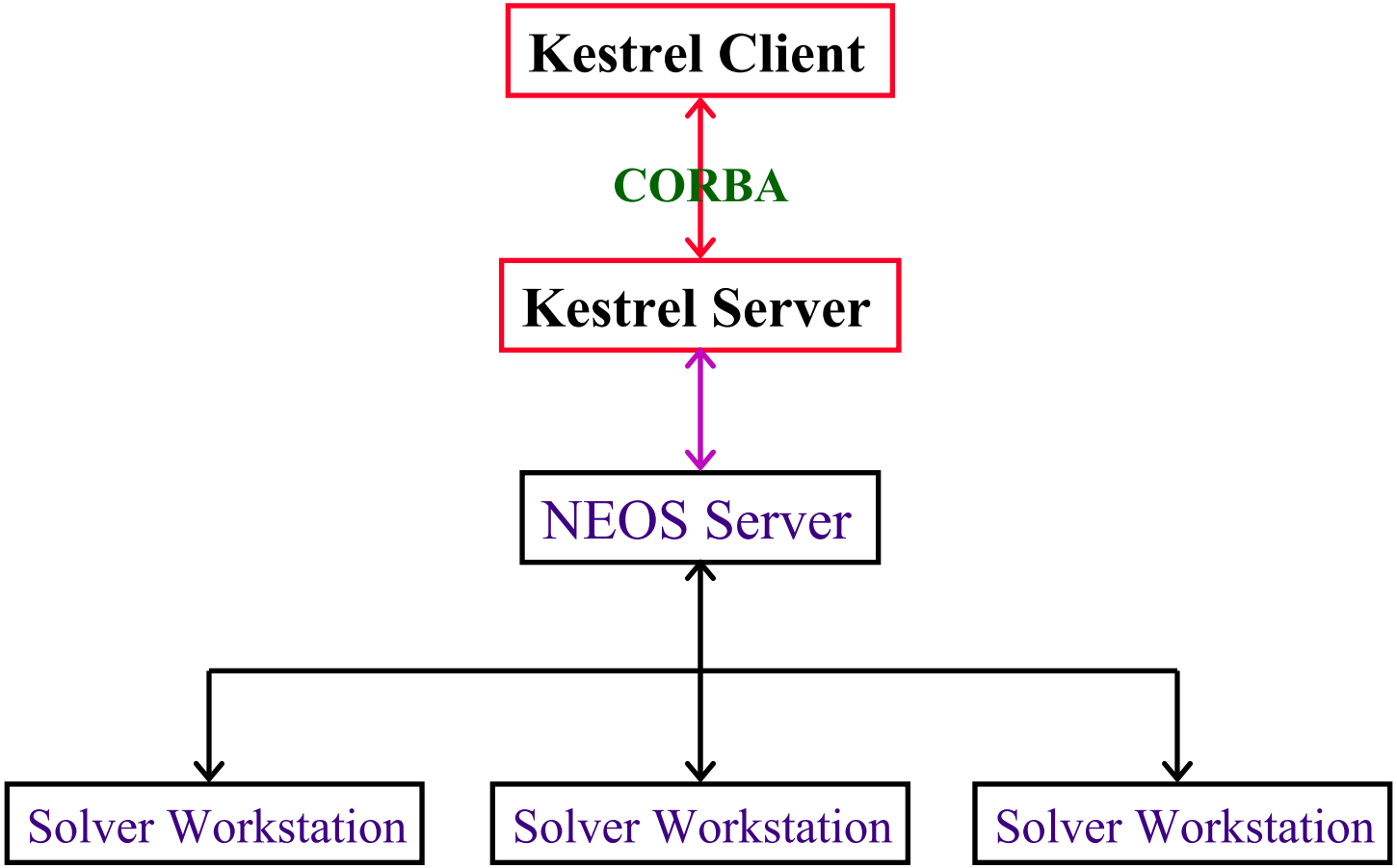
Using NEOS

Kestrel Interface

Final result listing



Outline of Operation



The CORBA Connection

Motivation

Make function calls over the Internet

Implementation

- Describe interface that remote will present to local, using Interface Definition Language
- Translate to "stub" and "skeleton" code for local and remote applications
- Implement C++ methods to realize the interface

Operation

- Local program identifies remote program by supplying Interoperable Object Reference
- Includes host name, port number, object key

. . . competitors: .NET, DCE, XML-RPC

The Kestrel Client

First call to server

- Instantiate Kestrel solver object
- Different derived classes for AMPL & GAMS

Second call to server

- Invoke a method of the Kestrel solver object to pass the problem and other information
- Retrieve NEOS job number and password

Third (blocking) call to server

- Invoke a method of the Kestrel solver object to request results
- *Wait for return of results*
- Return results to AMPL or GAMS

The Kestrel Server

Interaction with NEOS Server

- Concatenate received files, separated by delimiters
- Place file (uniquely named) in NEOS Server directory
- Wait for NEOS server to find file and start processing
- Retrieve job number and password from directory

. . . similar to protocol for web, socket input

. . . requires shared file system

Design considerations

- Servers must share a file system
- File locking greatly reduces overhead of checking for NEOS Server to process a job
- Each combination of a Kestrel client and a solver has its own entry in the NEOS Server data bank

Kestrel

Other 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

GAMS / AMPL

Motivation

Numerous long-established GAMS models & applications

Numerous solvers supported by AMPL but not GAMS

NEOS Server has access to both GAMS and AMPL

Operation

GAMS input file

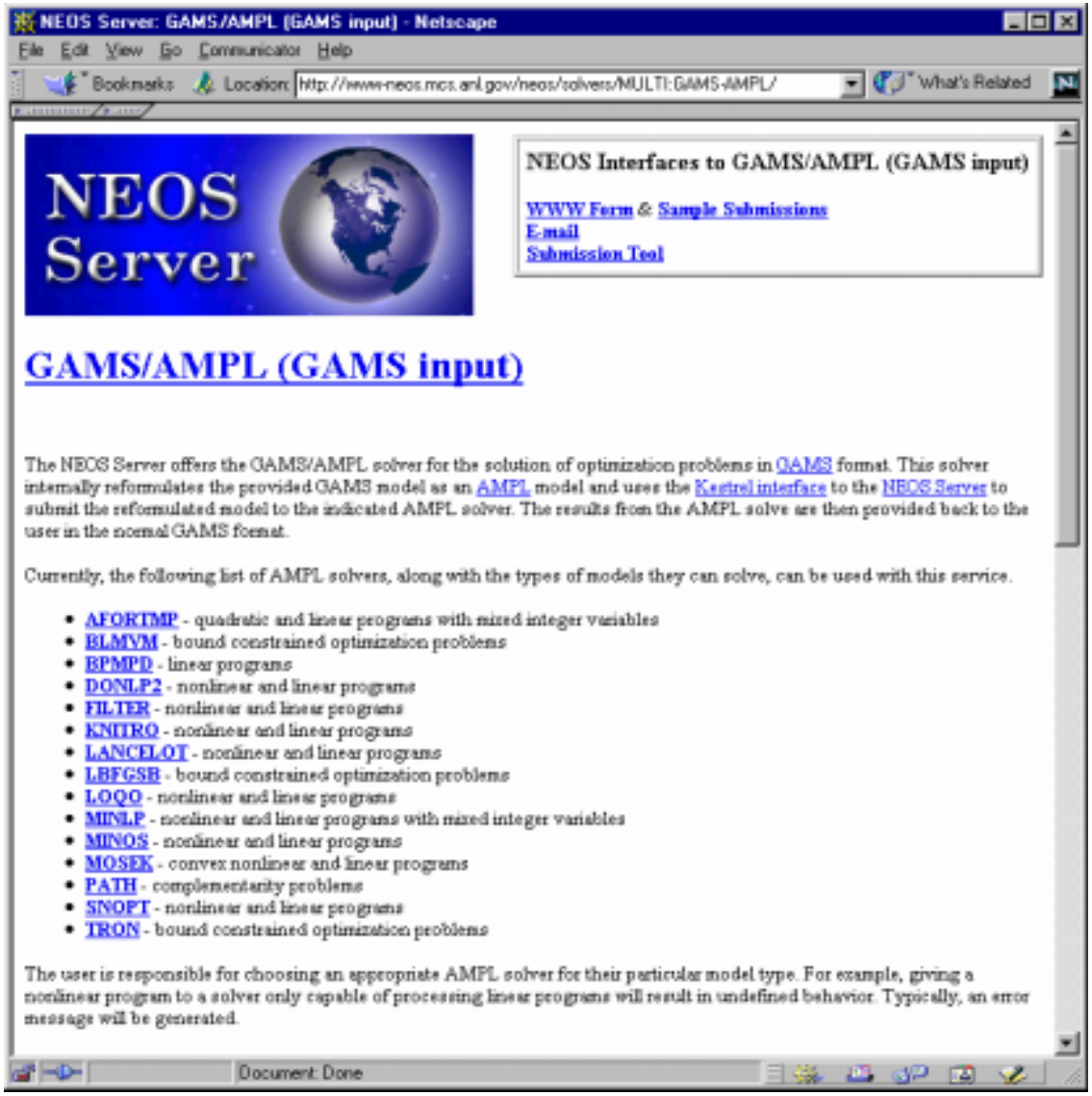
. . . translated to ***"scalar"*** AMPL by GAMS/CONVERT tool

. . . sent to solver by Kestrel / AMPL execution script

Results returned in GAMS listing

Kestrel

GAMS / AMPL *(cont'd)*



Kestrel

Multiple Submission & Retrieval

Loop through submissions

- use provided command file

```
for {p in PROD} {
   problem SubI[p];
   commands kestrelsub;
};
```

Loop through retrievals

- in the same order

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

Multiple Submission/Retrieval *(cont'd)*

kestrelsub

```
option␣ampl_id␣(_pid);
write␣bkestproblem;
shell␣␣kestrel␣submit␣kestproblem␣;
```

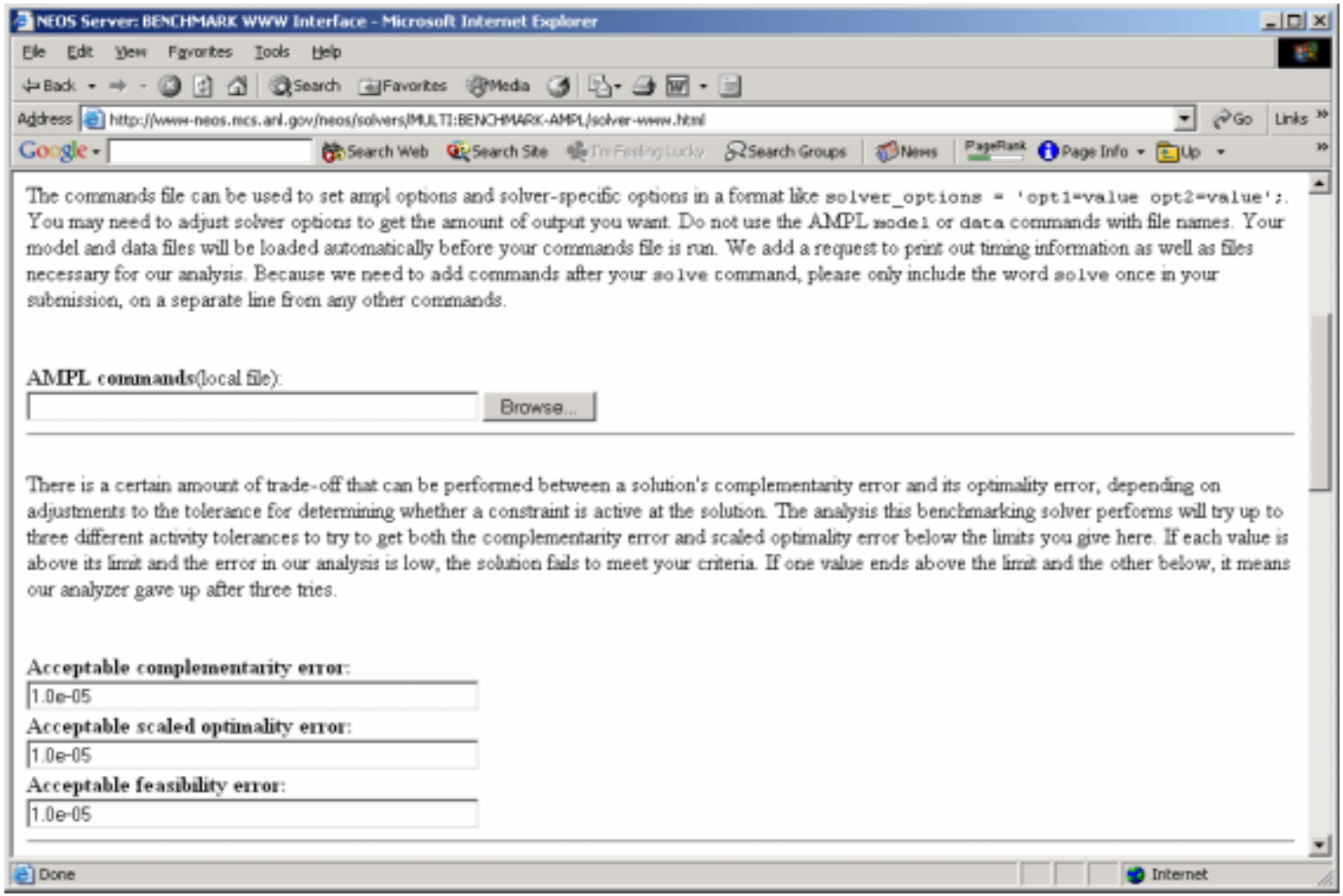
kestrelret

```
option␣ampl_id␣(_pid);
shell␣␣kestrel␣retrieve␣kestresult␣;
solution␣kestresult.sol;
```

. . . shared temporary file has name based on _pid

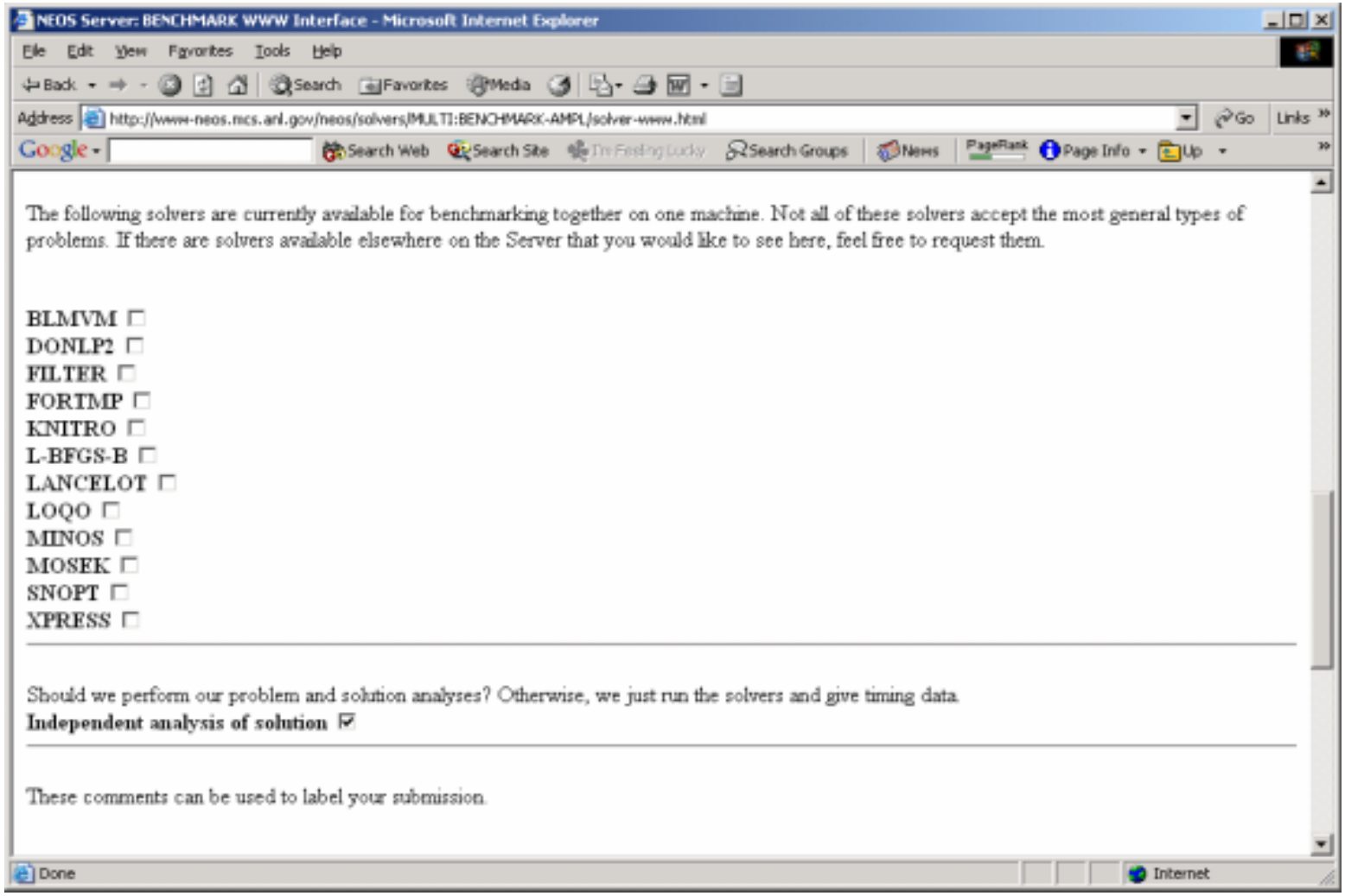
Benchmarking Interface

Benchmarking web page (instructions)



Benchmarking Interface *(cont'd)*

Web page (solver choice)



Verifying Results

Comparable running environments

- Same computer and operating system
- User's choice of solver parameters
- User's choice of tolerances for
  feasibility, optimality, complementarity

Independent assessment of solutions

- Based only on solution returned

E.D. Dolan, J.J. Moré and T.S. Munson,
"Optimality Measures for Performance Profiles"
(available from www.optimization-online.org)

Verifying Results *(cont'd)*

Benchmark verification results

```
Solver lbfgsb.

feasibility error                          0.000000e+00
complementarity error                      0.000000e+00
optimality error                           1.923416e-07
scaled optimality error                    3.827304e-06

Solver solution optimality and complementarity found
acceptable.

Solver loqo.

feasibility error                          0.000000e+00
complementarity error                      7.554012e-05
optimality error                           6.588373e-06
scaled optimality error                    1.311233e-04

Solver solution not acceptable by this analysis because the
scaled optimality error is greater than your limit of 1.0e-05
and the complementarity error is greater than your limit of
1.0e-05.
```

Frequently Asked Questions

Who uses NEOS?

- Where are its users from?
- How much is it used?

What kinds of solvers does NEOS offer?

- Who supplies them?
- Which are most heavily used?
- Where are they hosted?

How is NEOS supported?

- Who answers user questions?

Who Uses NEOS? *(a sample)*

- We are using NEOS services for duty-scheduling for ground handling activities in a regional airport environment.
- We used NEOS to solve nonlinear optimization problems associated with models of physical properties in chemistry.
- Our company is working with various projects concerning R&D of internal combustion engines for cars and brakes for heavy vehicles.
- We are working on bi-dimensional modeling of earth's conductivity distribution.
- I am dealing with ultimate limit-state analyses of large dams by means of a non-standard approach (“direct method”); this requires solving problems of linear and non-linear programming. The NEOS server is an extraordinary tool to perform parametric tests on small models, in order to choose the best suited solver.
- I have used NEOS with LOQO solver to optimize an interpolator. . . . My domain is digital receivers where the receiver clock is not changed to match the transmitter clock.

Who Uses NEOS? *(more)*

- I have been able to build and solve a prototype combinatorial auction MIP model using AMPL and NEOS in a fraction of the time it would have required me to do this had I needed to requisition a solver and install it locally.
- Our idea is trying to design antennas by using the computer. . . . We have tried various solvers on NEOS to see if this is possible at all.
- I am using the LOQO solver and code written in AMPL to perform numerical optimization of a spinor Bose-Einstein condensate.
- We are using the NEOS Server for solving linear and nonlinear complementarity problems in engineering mechanics and in robotics.
- I have been working on a system for protein structure prediction. . . . I had need to incorporate a nonlinear solver to handle packing of sidechain atoms in the protein.

Who Uses NEOS? *(academic)*

- I am regularly suggesting my students to use NEOS as soon as their projects in AMPL cannot be solved with the student edition. **So they debug their AMPL models locally . . . and then they run their real-life projects thanks to NEOS.**
- I didn't even know what nonlinear programming was and after I discovered what it was, it became clear how enormous a task it would be to solve the problems assigned to me. . . . I had extremely complicated objective functions, both convex and nonconvex, an armload of variables, and an armload of convex, nonconvex, equality and inequality constraints, but when I sent off the information via the web submission form, within seconds I received extremely accurate and consistent results. **The results were used for verifying a certain theory in my professor's research** and so accuracy was extremely important.
- NEOS has been a very valuable tool in the two graduate optimization courses that I teach. **NEOS allows students to see a broader variety of solvers than we have available . . .**

  . . . *more at* `www-neos.mcs.anl.gov/neos/stories.html`

Where are They From?

2004 through 24 April:
Identifiable domain and >= 20 submissions

(com)	3502
(edu)	13610
(gov)	2510
(net)	12310
(mil)	90

Northwestern	1338
Argonne	2194

Belgium(be)	4918
Brazil(br)	2558
Canada(ca)	4514
Switzerland(ch)	1098
Chile(cl)	158
Colombia(co)	218
Cyprus(cy)	20
Czech Republic(cz)	554
Germany(de)	1834
Spain(es)	1548
Finland(fi)	270
Micronesia(fm)	36
France(fr)	2730
Greece(gr)	776
Hong Kong(hk)	...

NEOS Users

What Countries are They From?

2004 through 24 April

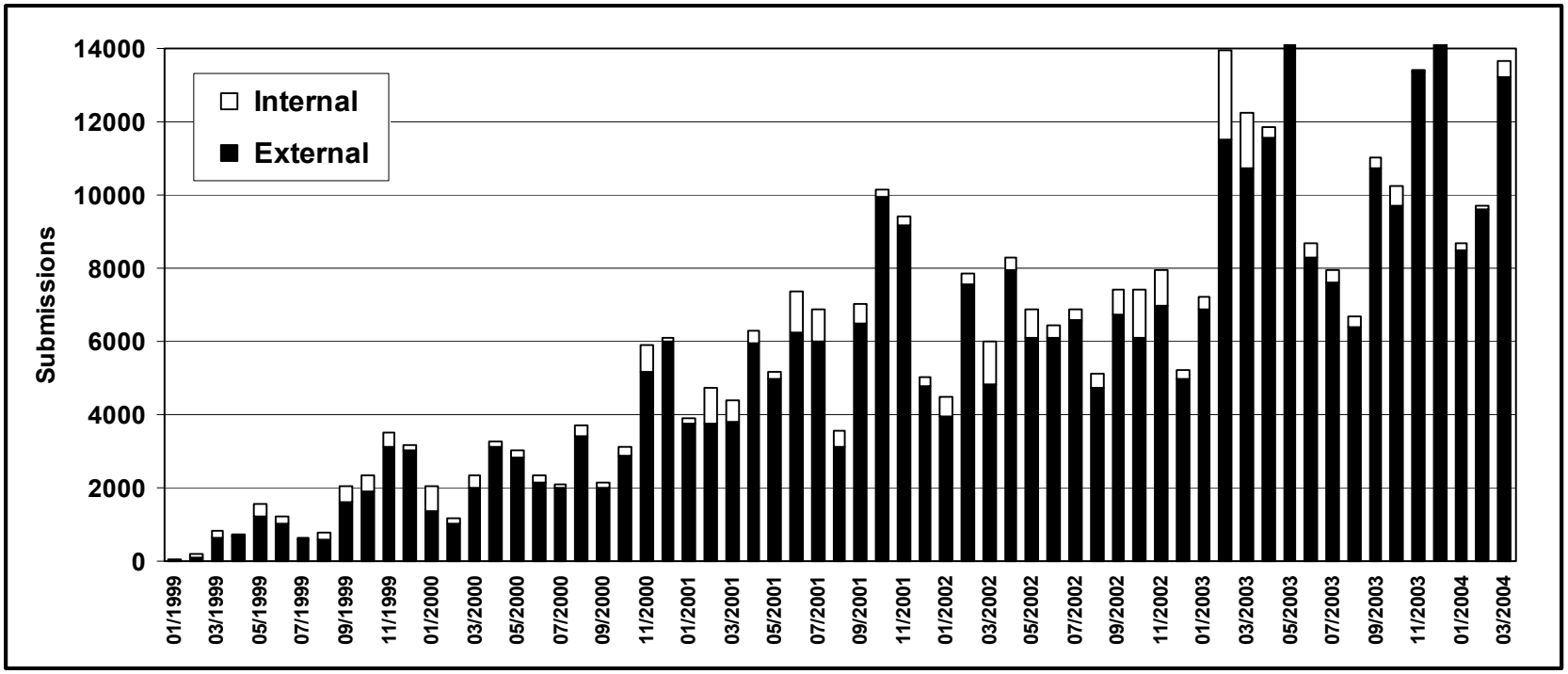
Belgium(be)	4918
Brazil(br)	2558
Canada(ca)	4514
Switzerland(ch)	1098
Chile(cl)	158
Colombia(co)	218
Cyprus(cy)	20
Czech Republic(cz)	554
Germany(de)	1834
Spain(es)	1548
Finland(fi)	270
Micronesia(fm)	36
France(fr)	2730
Greece(gr)	776
Hong Kong(hk)	466
Hungary(hu)	182

Israel(il)	1022
Italy(it)	1646
Japan(jp)	32
Luxembourg(lu)	40
Mexico(mx)	252
Malaysia(my)	912
Netherlands(nl)	4254
Norway(no)	82
New Zealand(nz)	20
Poland(pl)	302
Sweden(se)	3562
Singapore(sg)	236
USSR (former)(su)	834
Turkey(tr)	526
Taiwan(tw)	62
United Kingdom(uk)	4870
Venezuela(ve)	2042

NEOS Users

How Much Do They Use It?

Submissions by month, 1/1999 through 3/2004



. . . 15 / hour over past year

. . . 20 / hour in peak months

What Solvers Does NEOS Offer?

For familiar problem types

- Linear programming
- Linear network optimization
- Linear integer programming
- Nonlinear programming
- Stochastic linear programming
- Complementarity problems

For emerging problem types

- Nondifferentiable optimization
- Semi-infinite optimization
- Global optimization
- Nonlinear integer programming
- Semidefinite & 2nd-order cone programming

. . . virtually every published semidefinite programming code

Who Supplies Them?

Some commercial solver vendors

- Xpress, FortMP (mixed integer)
- CONOPT, KNITRO, MOSEK (nonlinear)

Universities and their researchers

- BonsaiG (mixed integer)
- DONLP2, FILTER, LANCELOT, LOQO, MINOS, SNOPT (nonlinear)

Open-Source Enthusiasts

- GLPK (mixed integer)

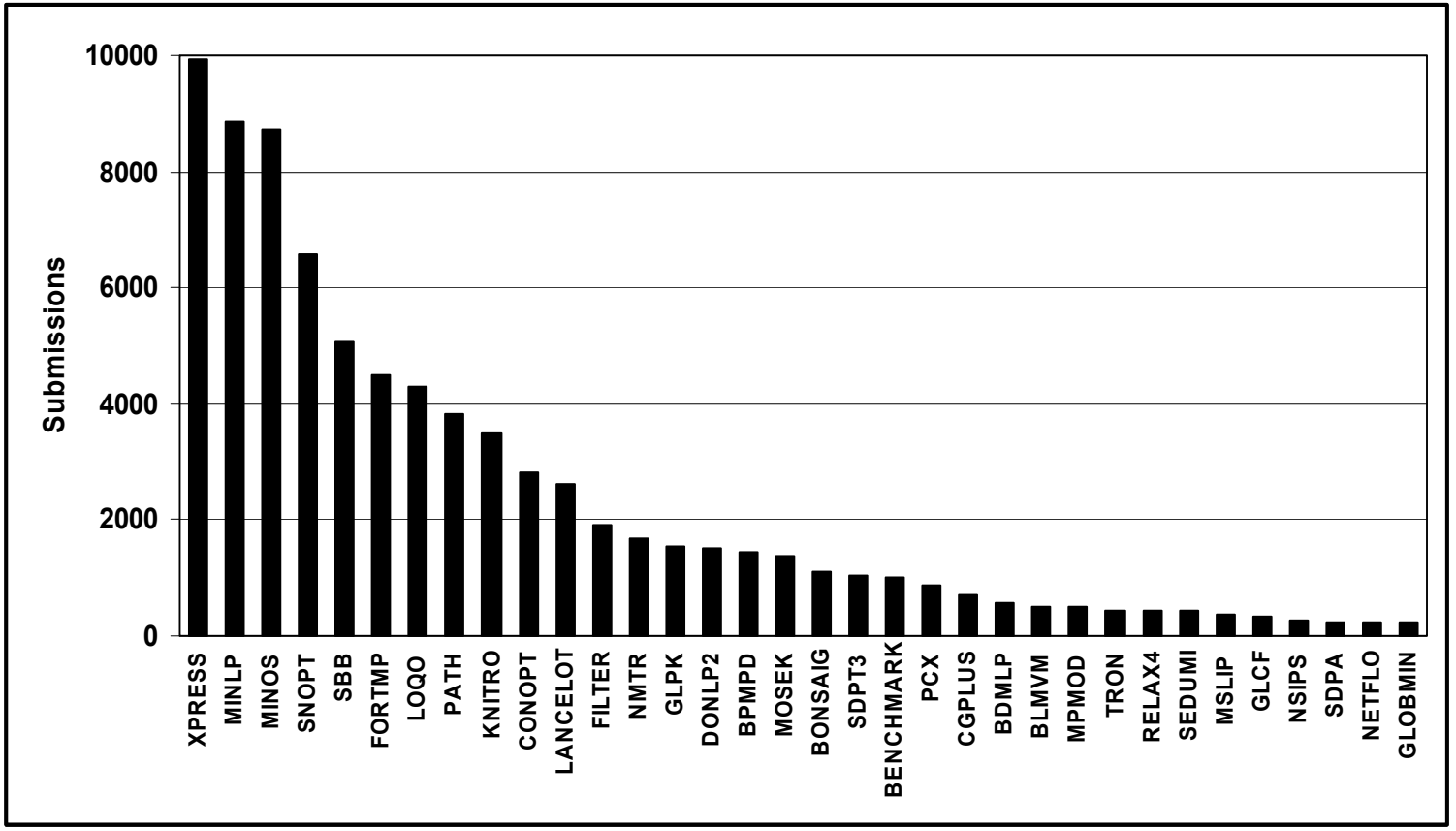
with thanks to . . .

- Modeling language vendors (AMPL, GAMS)
- ***Hans Mittelmann,*** Arizona State

NEOS Solvers

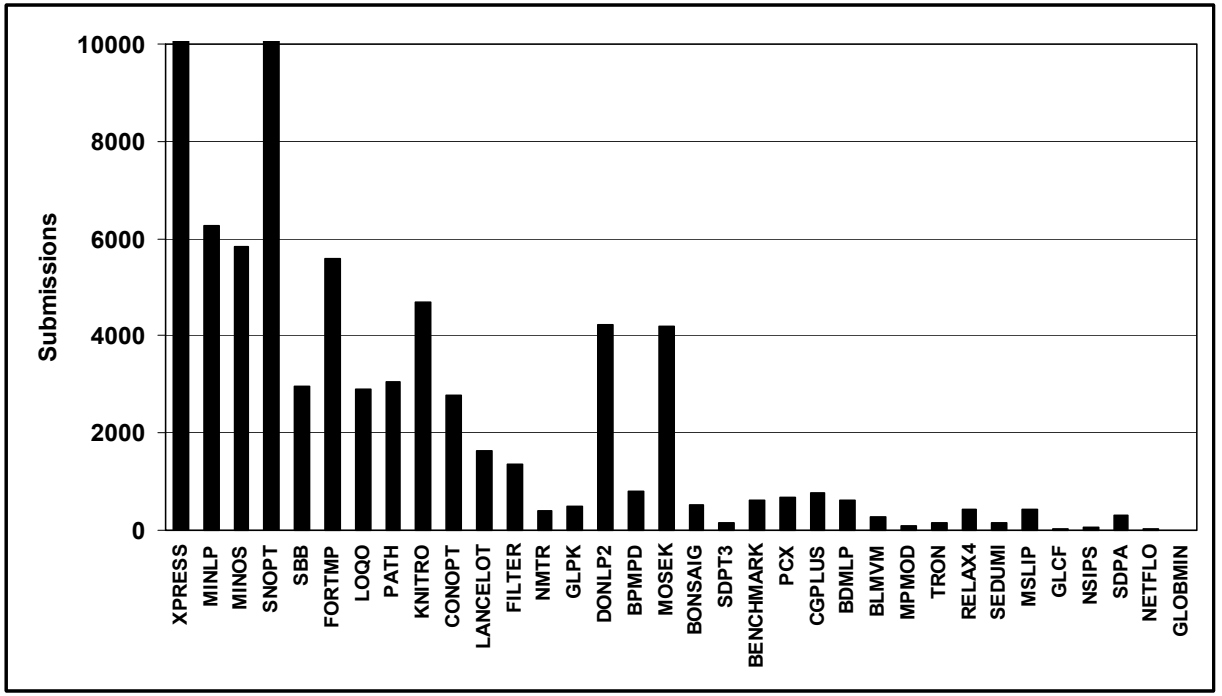
Which are Most Heavily Used?

Totals for 2002 . . .



Which are Most Heavily Used?

. . . totals since January 1st, 2003



NEOS Solvers

Where are They Hosted?

Varied workstations at

- Aachen University of Technology
- Argonne National Laboratory
- Arizona State University
- Lehigh University
- National Taiwan University
- Northwestern University *(with support from Sun Microsystems)*
- University of Wisconsin at Madison

. . . new hosts are readily added anywhere on the Internet

How is NEOS Supported?

Grants

- National Science Foundation, Operations Research Program, grant DMI-0322580
- National Science Foundation, Information Technology Research Program, grant CCR-0082807
- U.S. Department of Energy, Office of Advanced Scientific Computing, Mathematical, Information, and Computational Sciences Division subprogram, Contract W-31-109-Eng-38
- National Science Foundation, Challenges in Computational Science Program, grant CDA-9726385

Donations

- Processor cycles
- Many people's time

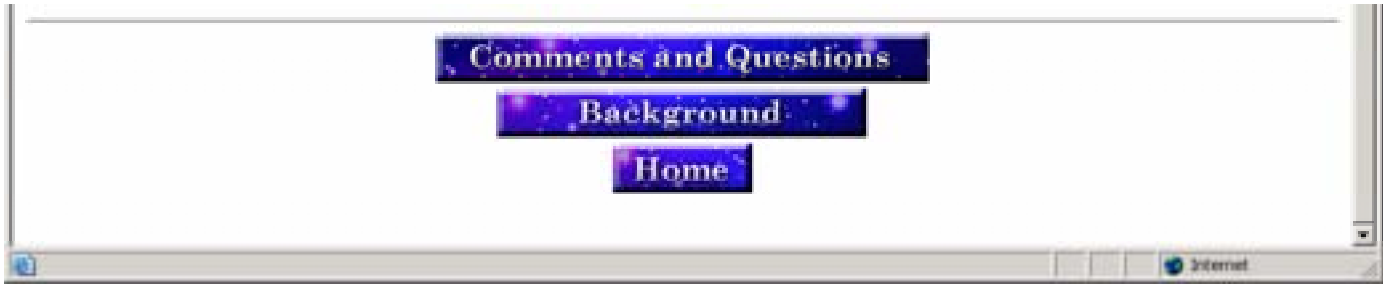
. . . no user charges as yet

Who Answers Users' Questions?

Large mailing list for support questions

- NEOS developers
- Solver developers

Support request buttons on every page



Recent & Forthcoming Developments

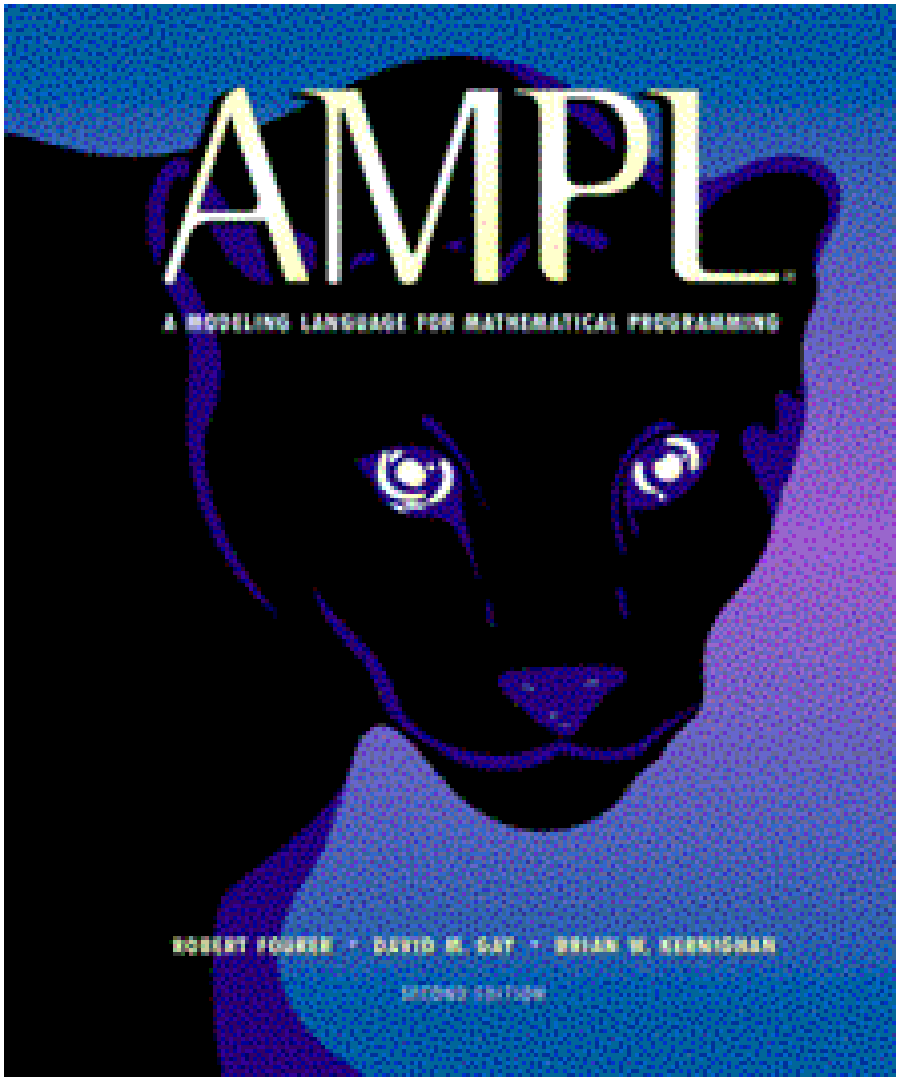
XML-standard formats

Problem analysis and solver choice

Web services

Benchmarking and verification

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.ampl.com/BOOK/