Model-Based Optimization

Best Practices and Current Trends

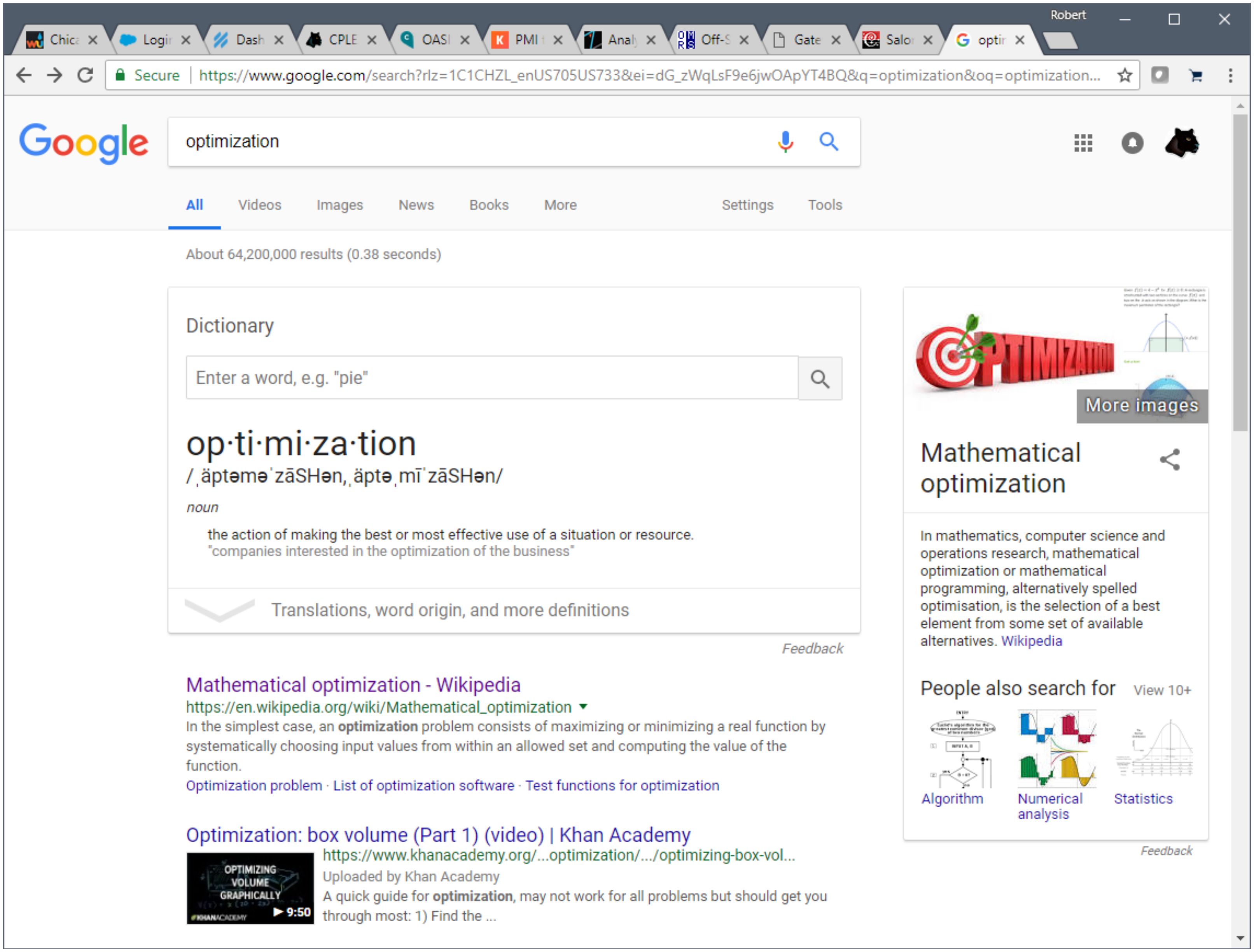
Robert Fourer

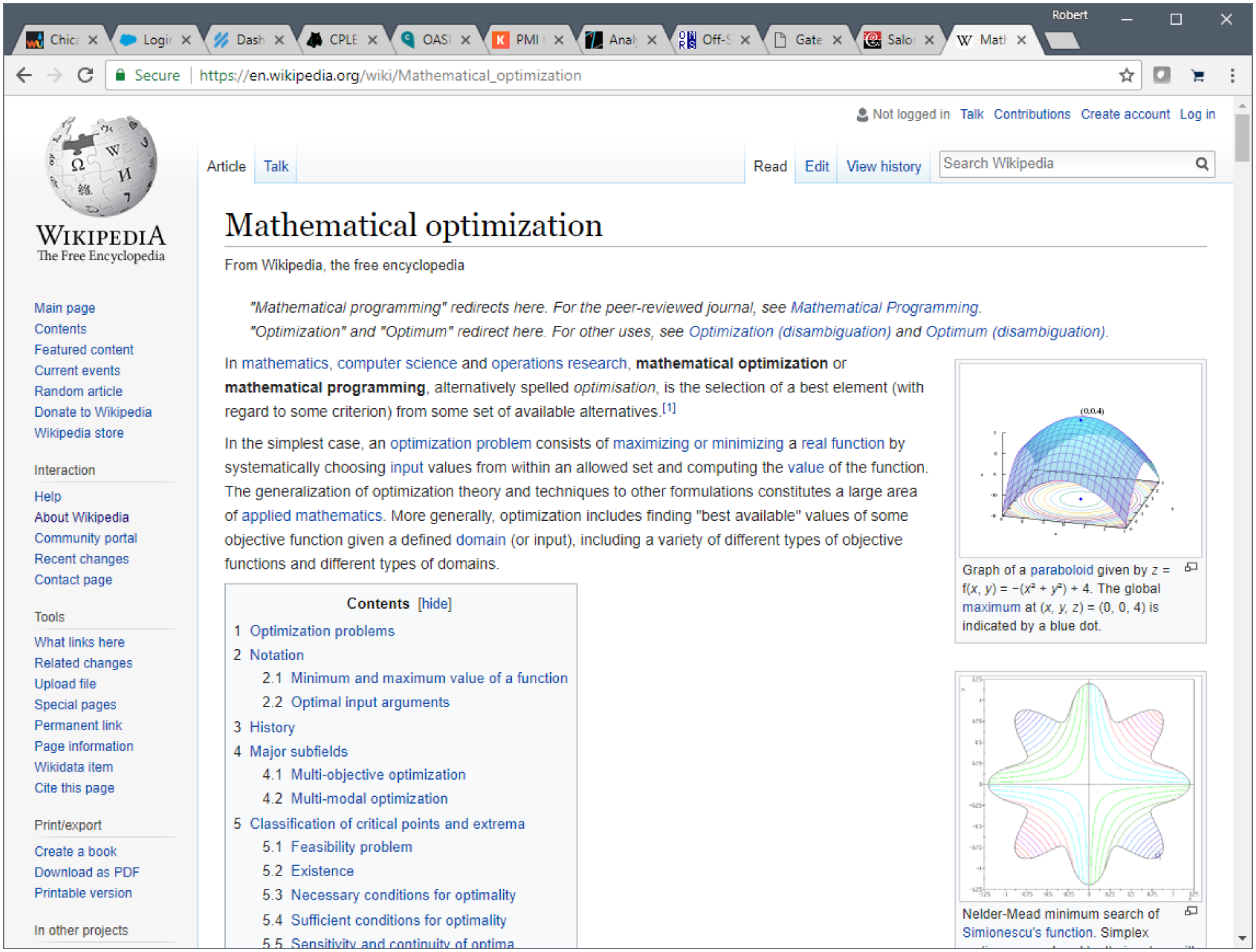
Northwestern University, AMPL Optimization Inc.

4er@northwestern.edu, 4er@ampl.com

INFORMS International Conference, Taipei, Taiwan

Tutorial Session MB06, 18 June 2018, 11:00-12:30


Robert
Secure
https://www.google.com/search?rlz=1C1CHZL_enUS705US733&ei=dG_zWqLsF9e6jwOApYT4BQ&q=optimization&oq=optimization...
optimization
All
Videos
Images
News
Books
More
Settings
Tools
About 64,200,000 results (0.38 seconds)
Dictionary
Enter a word, e.g. "pie"
op·ti·mi·za·tion
/ˌäptəməˈzāSHən,ˌäptəˌmīˈzāSHən/
noun
the action of making the best or most effective use of a situation or resource.
"companies interested in the optimization of the business"
Translations, word origin, and more definitions
Feedback
Mathematical optimization - Wikipedia
https://en.wikipedia.org/wiki/Mathematical_optimization
In the simplest case, an optimization problem consists of maximizing or minimizing a real function by systematically choosing input values from within an allowed set and computing the value of the function.
Optimization problem · List of optimization software · Test functions for optimization
Optimization: box volume (Part 1) (video) | Khan Academy
https://www.khanacademy.org/...optimization/.../optimizing-box-vol...
Uploaded by Khan Academy
A quick guide for optimization, may not work for all problems but should get you through most: 1) Find the ...
OPTIMIZING VOLUME GRAPHICALLY
9:50
More images
Mathematical optimization
In mathematics, computer science and operations research, mathematical optimization or mathematical programming, alternatively spelled optimisation, is the selection of a best element from some set of available alternatives. Wikipedia
People also search for
View 10+
Algorithm
Numerical analysis
Statistics
Feedback



https://en.wikipedia.org/wiki/Mathematical_optimization

Not logged in Talk Contributions Create account Log in

Article Talk Read Edit View history Search Wikipedia

WIKIPEDIA
The Free Encyclopedia

Main page
Contents
Featured content
Current events
Random article
Donate to Wikipedia
Wikipedia store

Interaction
Help
About Wikipedia
Community portal
Recent changes
Contact page

Tools
What links here
Related changes
Upload file
Special pages
Permanent link
Page information
Wikidata item
Cite this page

Print/export
Create a book
Download as PDF
Printable version

In other projects

Mathematical optimization

From Wikipedia, the free encyclopedia

"Mathematical programming" redirects here. For the peer-reviewed journal, see Mathematical Programming.

"Optimization" and "Optimum" redirect here. For other uses, see Optimization (disambiguation) and Optimum (disambiguation).

In mathematics, computer science and operations research, **mathematical optimization** or **mathematical programming**, alternatively spelled *optimisation*, is the selection of a best element (with regard to some criterion) from some set of available alternatives.[1]

In the simplest case, an optimization problem consists of maximizing or minimizing a real function by systematically choosing input values from within an allowed set and computing the value of the function. The generalization of optimization theory and techniques to other formulations constitutes a large area of applied mathematics. More generally, optimization includes finding "best available" values of some objective function given a defined domain (or input), including a variety of different types of objective functions and different types of domains.

Graph of a paraboloid given by $z = f(x, y) = -(x^2 + y^2) + 4$. The global maximum at $(x, y, z) = (0, 0, 4)$ is indicated by a blue dot.

Nelder-Mead minimum search of Simionescu's function. Simplex

Contents [hide]

Mathematical Optimization

In general terms,

- Given a function of some decision variables
- Choose values of the variables to
  make the function as large or as small as possible
- Subject to restrictions on the values of the variables

In practice,

- A paradigm for a very broad variety of problems
- A successful approach for finding solutions

Outline: Model-Based Optimization

Method-based vs. model-based approaches

- Example 1: Unconstrained optimization
- Example 2: Balanced assignment
- Example 3: Linear regression

Modeling languages for model-based optimization

- Executable languages
- Declarative languages

Solvers for model-based optimization

- “Linear” solvers
- “Nonlinear” solvers
- Other solver types

Example #1: Unconstrained Optimization

An example from calculus

- Min/Max $f(x_1, \ldots, x_n)$

  . . . where f is a smooth (differentiable) function

Approach #1

- Form $\nabla f(x_1, \ldots, x_n) = 0$
- Find an expression for the solution to these equations

Approach #2

- Choose a starting point $\mathbf{x}^0 = (x_1^0, \ldots, x_n^0)$
- Iterate $\mathbf{x}^{k+1} = \mathbf{x}^k + \mathbf{d},$ where $\nabla^2 f(\mathbf{x}^k) \cdot \mathbf{d} = -\nabla f(\mathbf{x}^k)$

  . . . until the iterates converge

What makes these approaches different?

What makes these approaches different?

Where You Put the Most Effort

Approach #1: Method-oriented

- Finding an expression that solves $\nabla f(x_1, \ldots, x_n) = 0$

  . . . requires a *different* "method" for each new form of f

Approach #2: Model-oriented

- Choosing f to model your problem

  . . . the *same* iteration procedure applies to any f and $\mathbf{x}^0$

  . . . can be implemented by general, *off-the-shelf* software

Am I leaving something out?

- To solve $\nabla^2 f\left(\mathbf{x}^k\right) \cdot \mathbf{d} = -\nabla f(\mathbf{x}^k)$,

  you need to form ∇ and ∇^2 for the given f

Am I leaving something out?

No . . . Software Can Handle Everything

Modeling

- Describing of f as a function of variables

Evaluation

- Computing $f(\mathbf{x}^k)$ from the description
- Computing $\nabla f(\mathbf{x}^k)$, $\nabla^2 f(\mathbf{x}^k)$ by automatic differentiation

Solution

- Applying the iterative algorithm
  - Computing the iterates
  - Testing for convergence

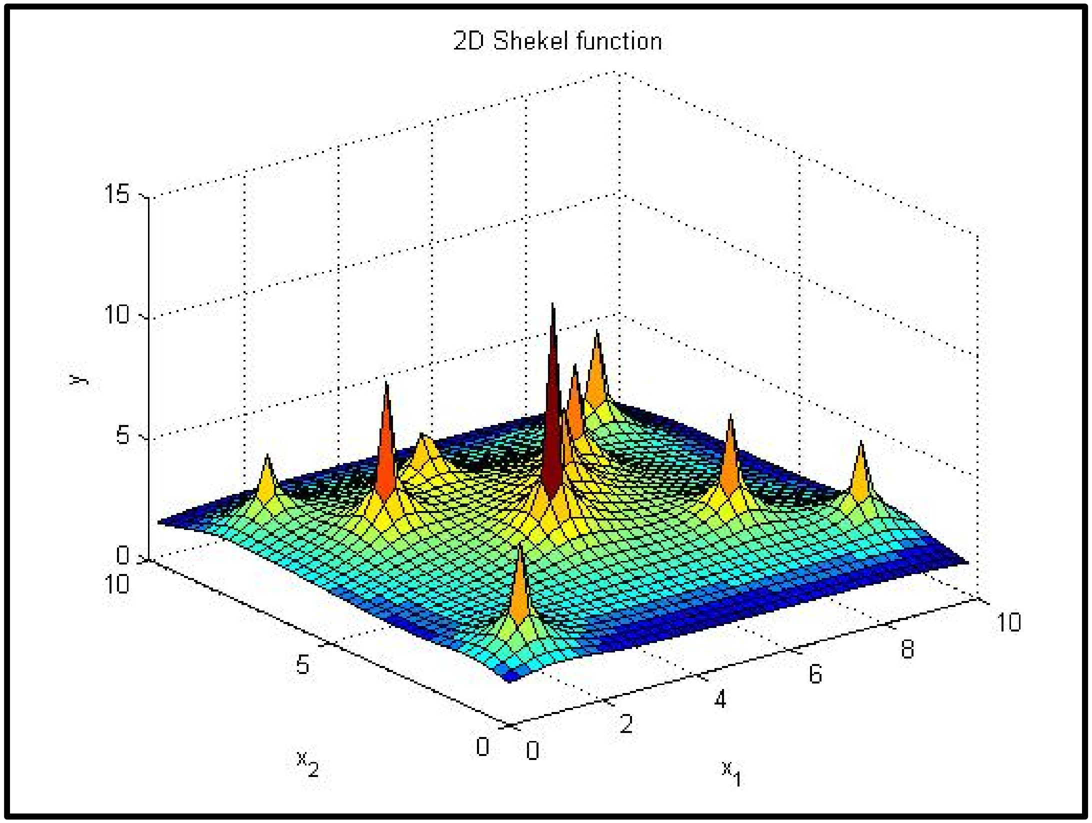
Send to an off-the-shelf solver?

- Choice of excellent smooth constrained nonlinear solvers
- Differentiable unconstrained optimization is a special case

Example 1a: Shekel Function

A small test case for solvers

- https://en.wikipedia.org/wiki/Shekel_function



Mathematical Formulation

Given

m number of locally optimal points

n number of variables

and

a_{ij} for each $i = 1, \ldots, m$ and $j = 1, \ldots, n$

c_i for each $i = 1, \ldots, m$

Determine

x_j for each $j = 1, \ldots, n$

to maximize

$$\sum_{i=1}^{m} 1/\left(c_i + \sum_{j=1}^{n} (x_j - a_{ij})^2\right)$$

Modeling Language Formulation

Symbolic model (AMPL)

```
param m integer > 0;
param n integer > 0;

param a {1..m, 1..n};
param c {1..m};

var x {1..n};

maximize objective:
   sum {i in 1..m} 1 / (c[i]  + sum {j in 1..n} (x[j] - a[i,j])^2);
```

$$\sum_{i=1}^{m} 1/\left(c_i + \sum_{j=1}^{n}(x_j - a_{ij})^2\right)$$

Modeling Language Data

Explicit data (independent of model)

```
param m := 5 ;
param n := 4 ;

param a:  1   2   3   4   :=
     1    4   4   4   4
     2    1   1   1   1
     3    8   8   8   8
     4    6   6   6   6
     5    3   7   3   7   ;

param c :=
     1   0.1
     2   0.2
     3   0.2
     4   0.4
     5   0.4 ;
```

Modeling Language Solution

Model + data = problem instance to be solved

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

ampl: option solver knitro;

ampl: solve;

Knitro 10.3.0: Locally optimal solution.
objective 5.055197729; feasibility error 0
6 iterations; 9 function evaluations

ampl: display x;

x [*] :=
1  1.00013
2  1.00016
3  1.00013
4  1.00016
;
```

Modeling Language Solution

. . . again with multistart option

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

ampl: option solver knitro;
ampl: option knitro_options 'ms_enable=1 ms_maxsolves=100';

ampl: solve;

Knitro 10.3.0: Locally optimal solution.
objective 10.15319968; feasibility error 0
43 iterations; 268 function evaluations

ampl: display x;

x [*] :=
1  4.00004
2  4.00013
3  4.00004
4  4.00013
;
```

Solution *(cont'd)*

. . . again with a "global" solver

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

ampl: option solver baron;

ampl: solve;

BARON 17.10.13 (2017.10.13):
43 iterations, optimal within tolerances.
Objective 10.15319968

ampl: display x;

x [*] :=
1  4.00004
2  4.00013
3  4.00004
4  4.00013
;
```

Example 1b: Protein Folding

Objective

- For a given protein molecule,
  find the configuration that has lowest energy
- Sum of 6 energy expressions defined in terms of variables

Variables

- Decision variables: *positions of atoms*
- Variables defined in terms of other variables: *22 kinds*

Data

- 8 index sets
- 30 collections of parameters, indexed over various sets

Modeling Language Formulation

Problem variables & some defined variables

```
var x {i in Atoms, j in D3} := x0[i,j];

var aax {i in Angles, j in D3} = x[it[i],j] - x[jt[i],j];
var abx {i in Angles, j in D3} = x[kt[i],j] - x[jt[i],j];

var a_axnorm {i in Angles} = sqrt(sum{j in D3} aax[i,j]^2);
var a_abdot {i in Angles} = sum {j in D3} aax[i,j]*abx[i,j];

.......

var cosphi {i in Torsions} = (sum{j in D3} ax[i,j]*bx[i,j])
                            / sqrt(sum{j in D3} ax[i,j]^2)
                            / sqrt(sum{j in D3} bx[i,j]^2);

var term {i in Torsions} = if np[i] == 1 then cosphi[i]
                            else if np[i] == 2 then 2*cosphi[i]^2 - 1
                            else 4*cosphi[i]^3 - 3*cosphi[i];
```

Modeling Language Formulation

Some more defined variables

```
var rinv14{i in Pairs14} =
   1. / sqrt( sum{j in D3} (x[i14[i],j] - x[j14[i],j])^2 );

var r614{i in Pairs14} =
   ((sigma[i14[i]] + sigma[j14[i]]) * rinv14[i]) ^ 6;

var rinv{i in Pairs} =
   1. / sqrt( sum{j in D3} (x[inb[i],j] - x[jnb[i],j])^2 );

var r6{i in Pairs} =
   ((sigma[inb[i]] + sigma[jnb[i]]) * rinv[i]) ^ 6;
```

Modeling Language Formulation

Components of total energy

```
var bond_energy = sum {i in Bonds} fcb[i] *
   (sqrt( sum {j in D3} (x[ib[i],j] - x[jb[i],j])^2 ) - b0[i] ) ^ 2

var angle_energy = sum {i in Angles} fct[i] *
   (atan2 (sqrt( sum{j in D3}
      (abx[i,j]*a_axnorm[i] - a_abdot[i]/a_axnorm[i]*aax[i,j])^2),
       a_abdot[i] ) - t0[i]) ^ 2

var torsion_energy =
   sum {i in Torsions} fcp[i]*(1 + cos(phase[i])*term[i])

var improper_energy =
   sum {i in Improper} (fcr[i] * idi[i]^2);
```

Modeling Language Formulation

Components of total energy (cont'd)

```
var pair14_energy =
   sum {i in Pairs14} ( 332.1667*q[i14[i]]*q[j14[i]]*rinv14[i]*0.5
      + sqrt(eps[i14[i]]*eps[j14[i]])*(r614[i]^2 - 2*r614[i]) );

var pair_energy =
   sum{i in Pairs} ( 332.1667*q[inb[i]]*q[jnb[i]]*rinv[i]
      + sqrt(eps[inb[i]]*eps[jnb[i]])*(r6[i]^2 - 2*r6[i]) );


minimize energy:
   bond_energy + angle_energy + torsion_energy +
   improper_energy + pair14_energy + pair_energy;
```

Modeling Language Data

Excerpts from parameter tables

```
param x0:  1                     2                     3           :=
  1   0                    0                     0
  2   1.0851518862529654   0                     0
  3  -0.35807838634224287  1.021365666308466     0
  4  -0.36428404194337122 -0.50505976829103794 -0.90115715381950734
  5  -0.52386736173121617 -0.69690490803763017  1.2465998798976687
  .......

param: ib jb  fcb         b0 :=
    1   1  2  340.000000  1.09000000
    2   1  3  340.000000  1.09000000
    3   1  4  340.000000  1.09000000

  .......

param : inb jnb :=
    1    1   10
    2    1   11
    3    1   12
  .......
```

Solution

Local optimum from Knitro run

```
ampl: model pfold.mod;
ampl: data pfold3.dat;

ampl: option solver knitro;
ampl: option show_stats 1;

ampl: solve;

Substitution eliminates 762 variables.
Adjusted problem:
66 variables, all nonlinear
0 constraints
1 nonlinear objective; 66 nonzeros.

Knitro 10.3.0: Locally optimal solution.
objective -32.38835099; feasibility error 0
13 iterations; 20 function evaluations
```

Solution

Local optimum from Knitro multistart run

```
ampl: model pfold.mod;
ampl: data pfold3.dat;

ampl: option solver knitro;
ampl: knitro_options 'ms_enable=1 ms_maxsolves=100';

ampl: solve;


Knitro 10.3.0: ms_enable=1
ms_maxsolves=100

Knitro 10.3.0: Locally optimal solution.
objective -32.39148536; feasibility error 0
3349 iterations; 4968 function evaluations
```

Solution

Details from Knitro run

```
Number of nonzeros in Hessian:  2211

    Iter       Objective   FeasError    OptError    ||Step||    CGits
       0  -2.777135e+001  0.000e+000
       1  -2.874955e+001  0.000e+000  1.034e+001  5.078e-001      142
       2  -2.890054e+001  0.000e+000  1.361e+001  1.441e+000        0
     ...
      11  -3.238833e+001  0.000e+000  6.225e-002  9.557e-002        0
      12  -3.238835e+001  0.000e+000  8.104e-004  3.341e-003        0
      13  -3.238835e+001  0.000e+000  8.645e-009  1.390e-005        0

# of function evaluations          =        20
# of gradient evaluations          =        14
# of Hessian evaluations           =        13

Total program time (secs)          =      0.022
Time spent in evaluations (secs)   =      0.009
```

Example #2: Balanced Assignment

Motivation

- meeting of employees from around the world

Given

- several employee categories
  (title, location, department, male/female)
- a specified number of project groups

Assign

- each employee to a project group

So that

- the groups have about the same size
- *the groups are as "diverse" as possible* with respect to all categories

Method-Based Approach

Define an algorithm to build a balanced assignment

- Start with all groups empty
- Make a list of people (employees)
- For each person in the list:
  - Add to the group whose resulting "sameness" will be least

```
Initialize all groups G = { }

Repeat for each person p
   sMin = Infinity

   Repeat for each group G
      s = total "sameness" in G ∪ {p}

      if s < sMin then
         sMin = s
         GMin = G

   Assign person p to group GMin
```

Method-Based Approach *(cont'd)*

Define a computable concept of "sameness"

- Sameness of a pair of people:
  - Number of categories in which they are the same
- Sameness in a group:
  - Sum of the sameness of all pairs of people in the group

Refine the algorithm to get better results

- Reorder the list of people
- Locally improve the initial "greedy" solution by swapping group members
- Seek further improvement through local search metaheuristics
  - What are the neighbors of an assignment?
  - How can two assignments combine to create a better one?

Model-Based Approach

Formulate a "minimal sameness" model

- Define decision variables for assignment of people to groups
  - $x_{ij} = 1$ if person 1 assigned to group j
  - $x_{ij} = 0$ otherwise
- Specify valid assignments through constraints on the variables
- Formulate sameness as an objective to be minimized
  - *Total sameness* = sum of the sameness of all groups

Send to an off-the-shelf solver

- Choice of excellent linear-quadratic mixed-integer solvers
- Zero-one optimization is a special case

Model-Based Formulation

Given

P set of people

C set of categories of people

t_{ik} type of person i within category k, for all $i \in P, k \in C$

and

G number of groups

$g^{\min}$ lower limit on people in a group

$g^{\max}$ upper limit on people in a group

Define

$s_{i_1 i_2} = |\{k \in C\colon\ t_{i_1 k} = t_{i_2 k}\}|$, for all $i_1 \in P, i_2 \in P$

sameness of persons i_1 and i_2

Model-Based Formulation *(cont'd)*

Determine

$x_{ij} \in \{0,1\}$ $= 1$ if person i is assigned to group j
$= 0$ otherwise, for all $i \in P,\ j = 1, \ldots, G$

To minimize

$\sum_{i_1 \in P} \sum_{i_2 \in P} s_{i_1 i_2} \sum_{j=1}^{G} x_{i_1 j}\, x_{i_2 j}$

total sameness of all pairs of people in all groups

Subject to

$\sum_{j=1}^{G} x_{ij} = 1,$ for each $i \in P$

each person must be assigned to one group

$g^{\min} \leq \sum_{i \in P} x_{ij} \leq g^{\max}$, for each $j = 1, \ldots, G$

each group must be assigned an acceptable number of people

Model-Based Solution

Optimize with an off-the-shelf solver

Choose among many alternatives

- Linearize and send to a mixed-integer linear solver
  - CPLEX, Gurobi, Xpress; CBC
- Send quadratic formulation to a mixed-integer solver that automatically linearizes products involving binary variables
  - CPLEX, Gurobi, Xpress
- Send quadratic formulation to a nonlinear solver
  - Mixed-integer nonlinear: Knitro, BARON
  - Continuous nonlinear (might come out integer): MINOS, Ipopt, . . .

Where Is the Work?

Method-based

- Programming an implementation of the method

Model-based

- Constructing a formulation of the model

Complications in Balanced Assignment

"Total Sameness" is problematical

- Hard for client to relate to goal of diversity
- *Minimize "total variation" instead*
  - Sum over all types: most minus least assigned to any group

Client has special requirements

- No employee should be "isolated" within their group
  - No group can have exactly one woman
  - Every person must have a group-mate from the same location and of equal or adjacent rank

Room capacities are variable

- Different groups have different size limits
- *Minimize "total deviation"*
  - Sum over all types: greatest violation of target range for any group

Method-Based *(cont'd)*

Revise or replace the original solution approach

- Total variation is less suitable to a greedy algorithm

Re-think improvement procedures

- Total variation is harder to locally improve
- Client constraints are challenging to enforce

Update or re-implement the method

- Even small changes to the problem can necessitate major changes to the method and its implementation

Model-Based *(cont'd)*

Add variables

$y_{kl}^{\min}$ fewest people of category k, type l in any group,

$y_{kl}^{\max}$ most people of category k, type l in any group,

for each $k \in C, l \in T_k = \bigcup_{i \in P} \{t_{ik}\}$

Add defining constraints

$y_{kl}^{\min} \leq \sum_{i \in P: t_{ik}=l} x_{ij}$, for each $j = 1, \ldots, G;\ k \in C, l \in T_k$

$y_{kl}^{\max} \geq \sum_{i \in P: t_{ik}=l} x_{ij}$, for each $j = 1, \ldots, G;\ k \in C, l \in T_k$

Minimize total variation

$\sum_{k \in C} \sum_{l \in T_k} (y_{kl}^{\max} - y_{kl}^{\min})$

. . . generalizes to handle varying group sizes

Model-Based *(cont'd)*

To express client requirement for women in a group, let

$$Q = \{i \in P: t_{i,\mathrm{m/f}} = \text{female}\}$$

Add constraints

$$\sum_{i \in Q} x_{ij} = 0 \text{ or } \sum_{i \in Q} x_{ij} \geq 2, \text{ for each } j = 1, \ldots, G$$

Model-Based *(cont'd)*

To express client requirement for women in a group, let

$$Q = \{i \in P: t_{i,\mathrm{m/f}} = \text{female}\}$$

Define logic variables

$$z_j \in \{0,1\} = 1 \text{ if any women assigned to group } j$$
$$= 0 \text{ otherwise, for all } j = 1, \ldots, G$$

Add constraints relating logic to assignment variables

$$2z_j \leq \sum_{i \in Q} x_{ij} \leq |Q|\, z_j, \text{ for each } j = 1, \ldots, G$$

Model-Based *(cont'd)*

To express client requirements for group-mates, let

$P_{l_1 l_2} = \{i \in P: t_{i,\text{loc}} = l_1, t_{i,\text{rank}} = l_2\}$, for all $l_1 \in T_{\text{loc}}, l_2 \in T_{\text{rank}}$

$A_l \subseteq T_{\text{rank}}$ ranks adjacent to rank l, for all $l \in T_{\text{rank}}$

Add constraints

$$\sum_{i \in P_{l_1 l_2}} x_{ij} = 0 \text{ or } \sum_{i \in P_{l_1 l_2}} x_{ij} + \sum_{l \in A_{l_2}} \sum_{i \in P_{l_1 l}} x_{ij} \geq 2,$$

$$\text{for each } l_1 \in T_{\text{loc}}, l_2 \in T_{\text{rank}}, \; j = 1, \ldots, G$$

Model-Based *(cont'd)*

To express client requirements for group-mates, let

$P_{l_1 l_2} = \{i \in P: t_{i,\mathrm{loc}} = l_1, t_{i,\mathrm{rank}} = l_2\}$, for all $l_1 \in T_{\mathrm{loc}}, l_2 \in T_{\mathrm{rank}}$

$A_l \subseteq T_{\mathrm{rank}}$ ranks adjacent to rank l, for all $l \in T_{\mathrm{rank}}$

Define logic variables

$w_{l_1 l_2 j} \in \{0,1\}$ $= 1$ if group j has anyone from location l_1 of rank l_2
$= 0$ otherwise, for all $l_1 \in T_{\mathrm{loc}}, l_2 \in T_{\mathrm{rank}}, \ j = 1, \ldots, G$

Add constraints relating logic to assignment variables

$w_{l_1 l_2 j} \le \sum_{i \in P_{l_1 l_2}} x_{ij} \le \left|P_{l_1 l_2}\right| w_{l_1 l_2 j}$,

$\sum_{i \in P_{l_1 l_2}} x_{ij} + \sum_{l \in A_{l_2}} \sum_{i \in P_{l_1 l}} x_{ij} \ge 2 w_{l_1 l_2 j}$,

for each $l_1 \in T_{\mathrm{loc}}, l_2 \in T_{\mathrm{rank}}, \ j = 1, \ldots, G$

Example #3: Linear Regression

Given

- Observed vector $\mathbf{y}$
- Regressor vectors $\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_p$

Choose multipliers $\beta_1, \beta_2, \ldots, \beta_p$ to . . .

- Approximate $\mathbf{y}$ by $\sum_{i=1}^{p} \mathbf{x}_i \beta_i$
- Explain $\mathbf{y}$ convincingly

Method-Based *(traditional)*

Iteratively improve choice of regressors

- Repeat
  - Solve a minimum-error problem using a subset of the regressors
  - Remove and re-add regressors as suggested by results
- Until remaining regressors are judged satisfactory

Results are varied

- Depends on interpretation and judgement of results
- "As much an art as a science"

Model-Based

Define a "best" choice of regressors

- ❖ Build a mixed-integer optimization model
  - ∗ Objective function minimizes error
  - ∗ Constraints specify desired properties of the regressor set

Optimize with an off-the-shelf solver

- ❖ Many excellent choices available
  - ∗ Linear-quadratic mixed-integer
  - ∗ General nonlinear mixed-integer

Dimitris Bertsimas and Angela King,
"An Algorithmic Approach to Linear Regression."
Operations Research **64** (2016) 2–16.

Algebraic Formulation

Given

m number of observations

n number of regressors

and

y_i observations, for each $i = 1, \ldots, m$

x_{ji} regressor values corresponding to observation i,
for each $j = 1, \ldots, n$ and $i = 1, \ldots, m$

Algebraic Formulation

Determine

β_j Multiplier for regressor j, for each $j = 1, \ldots, n$

z_j 1 if $\beta_j \neq 0$: regressor j is used,
0 if $\beta_j = 0$: regressor j is *not* used, for each $j = 1, \ldots, n$

to minimize

$$\sum_{i=1}^{m} \left(y_i - \sum_{j=1}^{n} x_{ji} \beta_j \right)^2 + \Gamma \sum_{j=1}^{n} \left| \beta_j \right|$$

Sum of squared errors
plus "lasso" term for regularization and robustness

Algebraic Formulation

Subject to

$-Mz_j \le \beta_j \le Mz_j$ for all $j = 1, \ldots, n$

If the j^{th} regressor is used then $z_j = 1$
(where M is a reasonable bound on $|\beta_j|$)

$\sum_{j=1}^{n} z_j \le k$

At most k regressors may be used

$z_{j_1} = \ldots = z_{j_{k(p)}}$ for $j_1, \ldots, j_{k(p)} \in \mathcal{GS}_p$, $p = 1, \ldots, n_{\mathcal{GS}}$

All regressors in each group sparsity set $\mathcal{GS}_p$
are either used or not used

$z_{j_1} + z_{j_2} \le 1$ for all $(j_1, j_2) \in \mathcal{HC}$

For any pair of highly collinear regressors,
only one may be used

Algebraic Formulation

Subject to

$\sum_{j \in \mathcal{T}_p} z_j \leq 1$ for all $p = 1, \ldots, n_{\mathcal{T}}$

For a regressor and any of its transformations, only one may be used

$z_j = 1$ for all $j \in \mathcal{I}$

Specified regressors must be used

$\sum_{j \in S_p} z_j \leq |\mathcal{S}_p| - 1$ for all $p = 1, \ldots, n_S$

Exclude previous solutions using β_j, $j \in S_p$

Method-Based Remains Popular for . . .

Problems hard to formulate for off-the-shelf solvers

- ❖ "Logic" constraints
  - ∗ sequencing, scheduling, cutting, packing
- ❖ "Black box" functions
  - ∗ simulations, approximations

Large, specialized applications

- ❖ Routing delivery trucks nationwide
- ❖ Finding shortest routes in mapping apps

Metaheuristic schemes

- ❖ Evolutionary methods, simulated annealing, . . .

Artificial intelligence and related computer science

- ❖ Constraint programming
- ❖ Training deep neural networks

Model-Based Has Become Standard for . . .

Diverse application areas

- Operations research & management science
- Business analytics
- Engineering & science
- Economics & finance

Diverse kinds of users

- Anyone who took an “optimization” class
- Anyone else with a technical background
- Newcomers to optimization

These have in common . . .

- Good algebraic formulations for off-the-shelf solvers
- Users focused on modeling

Trends Favor Model-Based Optimization

Off-the-shelf solvers keep improving

- Solve the same problems faster and faster
- Handle broader problem classes
- Recognize special cases automatically

Optimization has become more model-based

- Off-the-shelf solvers for constraint programming
- Model-based metaheuristics (“Matheuristics”)

Hybrid approaches have become easier to build

- Model-based APIs for solvers
- APIs for algebraic modeling systems

Modeling Languages for Model-Based Optimization

Background

- The modeling lifecycle
- Matrix generators
- Modeling languages

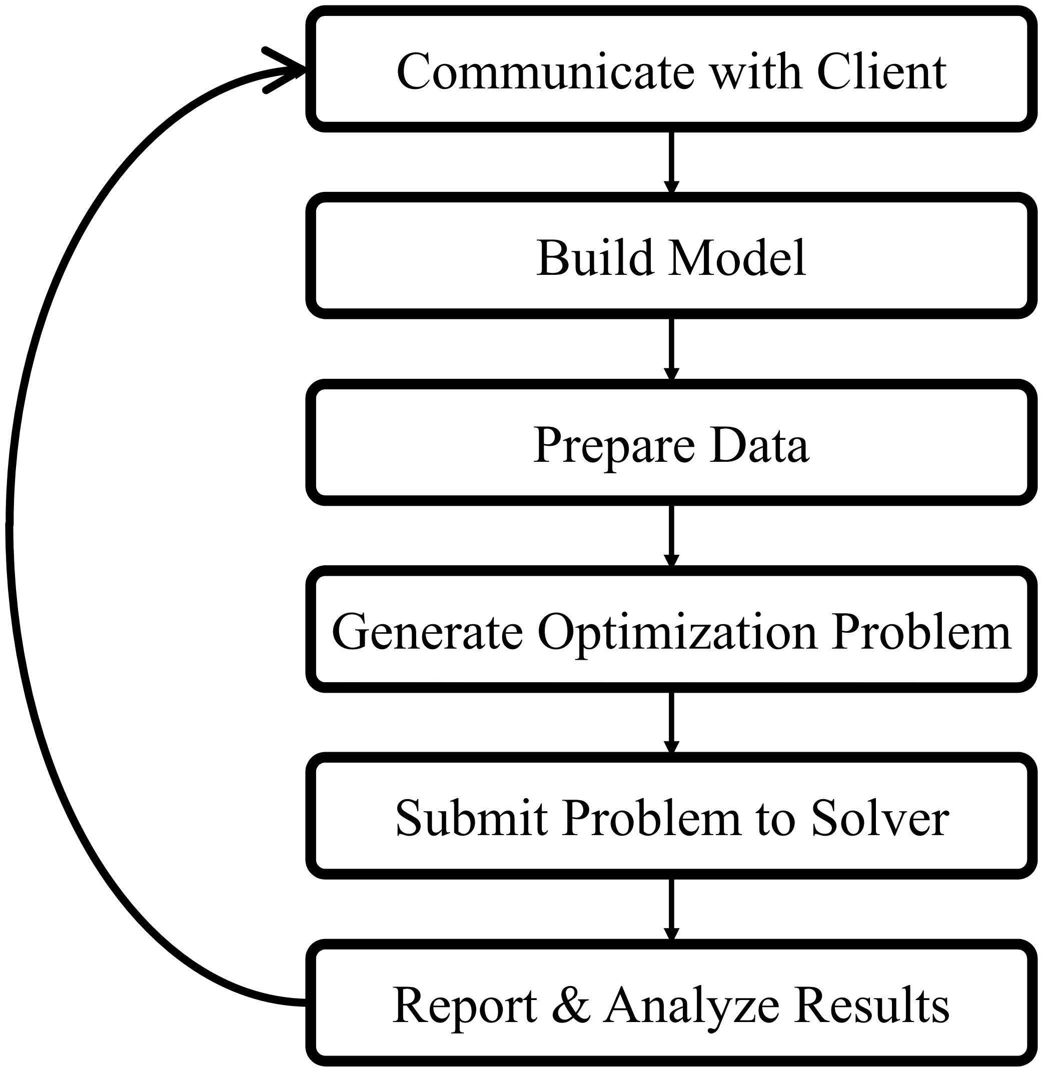
Algebraic modeling languages

- Design approaches: declarative, executable
- Example: AMPL vs. gurobipy
- Survey of available software

Balanced assignment model in AMPL

- Formulation
- Solution

The Optimization Modeling Lifecycle



Managing the Modeling Lifecycle

Goals for optimization software

- Repeat the cycle quickly and reliably
- Get results before client loses interest
- Deploy for application

Complication: two forms of an optimization problem

- Modeler's form
  - Mathematical description, easy for people to work with
- Solver's form
  - Explicit data structure, easy for solvers to compute with

Challenge: translate between these two forms

Matrix Generators

Write a program to generate the solver's form

- ❖ Read data and compute objective & constraint coefficients
- ❖ Communicate with the solver via its API
- ❖ Convert the solver's solution for viewing or processing

Some attractions

- ❖ Ease of embedding into larger systems
- ❖ Access to advanced solver features

Serious disadvantages

- ❖ Difficult environment for modeling
  - ∗ program does not resemble the modeler's form
  - ∗ model is not separate from data
- ❖ Very slow modeling cycle
  - ∗ hard to check the program for correctness
  - ∗ hard to distinguish modeling from programming errors

[1980] Over the past seven years we have perceived that the size distribution of general structure LP problems being run on commercial LP codes has remained about stable. . . . A 3000 constraint LP model is still considered large and very few LP problems larger than 6000 rows are being solved on a production basis. . . . That this distribution has not noticeably changed despite a massive change in solution economics is unexpected.

We do not feel that the linear programming user's most pressing need over the next few years is for a new optimizer that runs twice as fast on a machine that costs half as much (although this will probably happen). Cost of optimization is just not the dominant barrier to LP model implementation. The process required to manage the data, formulate and build the model, report on and analyze the results costs far more, and is much more of a barrier to effective use of LP, than the cost/performance of the optimizer.

Why aren't more larger models being run? It is not because they could not be useful; it is because we are not successful in using them. . . . They become unmanageable. LP technology has reached the point where anything that can be formulated and understood can be optimized at a relatively modest cost.

C.B. Krabek, R.J. Sjoquist and D.C. Sommer, The APEX Systems: Past and Future. *SIGMAP Bulletin* **29** (April ***1980***) 3–23.

Modeling Languages

Describe your model

- Write your symbolic model in a *computer-readable modeler's form*
- Prepare data for the model
- Let computer translate to & from the solver's form

Limited drawbacks

- Need to learn a new language
- Incur overhead in translation
- Make formulations clearer and hence easier to steal?

Great advantages

- Faster modeling cycles
- More reliable modeling
- More maintainable applications

[1982] The aim of this system is to provide one representation of a model which is easily understood by both humans and machines. . . . With such a notation, the information content of the model representation is such that a machine can not only check for algebraic correctness and completeness, but also interface automatically with solution algorithms and report writers.

. . . a significant portion of total resources in a modeling exercise . . . is spent on the generation, manipulation and reporting of models. It is evident that this must be reduced greatly if models are to become effective tools in planning and decision making.

The heart of it all is the fact that solution algorithms need a data structure which, for all practical purposes, is impossible to comprehend by humans, while, at the same time, meaningful problem representations for humans are not acceptable to machines. We feel that the two translation processes required (to and from the machine) can be identified as the main source of difficulties and errors. GAMS is a system that is designed to eliminate these two translation processes, thereby lifting a technical barrier to effective modeling . . .

J. Bisschop and A. Meeraus, On the Development of a General Algebraic Modeling System in a Strategic Planning Environment. *Mathematical Programming Study* **20** (***1982***) 1–29.

[1983] These two forms of a linear program — the modeler's form and the algorithm's form — are not much alike, and yet neither can be done without. Thus any application of linear optimization involves translating the one form to the other. This process of translation has long been recognized as a difficult and expensive task of practical linear programming.

In the traditional approach to translation, the work is divided between modeler and machine. . . .

There is also a quite different approach to translation, in which as much work as possible is left to the machine. The central feature of this alternative approach is a *modeling language* that is written by the modeler and translated by the computer. A modeling language is not a programming language; rather, it is a declarative language that expresses the modeler's form of a linear program in a notation that a computer system can interpret.

R. Fourer, Modeling Languages Versus Matrix Generators for Linear Programming.
ACM Transactions on Mathematical Software **9** (***1983***) 143–183.

Algebraic Modeling Languages

Designed for a model-based approach

- Define data in terms of sets & parameters
  - Analogous to database keys & records
- Define decision variables
- Minimize or maximize a function of decision variables
- Subject to equations or inequalities
  that constrain the values of the variables

Advantages

- Familiar
- Powerful
- Proven

Overview

Design alternatives

- *Executable:* object libraries for programming languages
- *Declarative:* specialized optimization languages

Design comparison

- Executable versus declarative using one simple example

Survey

- Solver-independent vs. solver-specific
- Proprietary vs. free
- Notable specific features

Executable

Concept

- Create an algebraic modeling language
  inside a general-purpose programming language
- Redefine operators like + and <=
  to return constraint objects rather than simple values

Advantages

- Ready integration with applications
- Good access to advanced solver features

Disadvantages

- Programming issues complicate description of the model
- Modeling and programming bugs are hard to separate
- Efficiency issues are more of a concern

Declarative

Concept

- Design a language specifically for optimization modeling
  - Resembles mathematical notation as much as possible
- Extend to command scripts and database links
- Connect to external applications via APIs

Disadvantages

- Adds a system between application and solver
- Does not have a full object-oriented programming framework

Advantages

- Streamlines model development
- Promotes validation and maintenance of models
- Works with many popular programming languages

Comparison: Executable *vs.* Declarative

Two representative widely used systems

- Executable: *gurobipy*
  - Python modeling interface for Gurobi solver
  - http://gurobi.com
- Declarative: *AMPL*
  - Specialized modeling language with multi-solver support
  - http://ampl.com

Comparison

Data

gurobipy

- Assign values to Python lists and dictionaries

```
commodities = ['Pencils', 'Pens']

nodes = ['Detroit', 'Denver',
 'Boston', 'New York', 'Seattle']

arcs, capacity = multidict({
 ('Detroit', 'Boston'):   100,
 ('Detroit', 'New York'):  80,
 ('Detroit', 'Seattle'):  120,
 ('Denver',  'Boston'):   120,
 ('Denver',  'New York'): 120,
 ('Denver',  'Seattle'):  120 })
```

- Provide data later in a separate file

AMPL

- Define symbolic model sets and parameters

```
set COMMODITIES;
set NODES;

set ARCS within {NODES,NODES};
param capacity {ARCS} >= 0;
```

```
set COMMODITIES := Pencils Pens ;

set NODES := Detroit Denver
 Boston 'New York' Seattle ;

param: ARCS: capacity:
        Boston 'New York' Seattle :=
 Detroit   100      80      120
 Denver    120     120      120  ;
```

Comparison

Data *(cont'd)*

gurobipy

```
inflow = {
 ('Pencils', 'Detroit'):   50,
 ('Pencils', 'Denver'):    60,
 ('Pencils', 'Boston'):   -50,
 ('Pencils', 'New York'): -50,
 ('Pencils', 'Seattle'):  -10,
 ('Pens',    'Detroit'):   60,
 ('Pens',    'Denver'):    40,
 ('Pens',    'Boston'):   -40,
 ('Pens',    'New York'): -30,
 ('Pens',    'Seattle'):  -30 }
```

AMPL

```
param inflow {COMMODITIES,NODES};
```

```
param inflow (tr):
              Pencils  Pens :=
  Detroit       50      60
  Denver        60      40
  Boston       -50     -40
 'New York'    -50     -30
  Seattle      -10     -30   ;
```

Comparison

Data *(cont'd)*

gurobipy

```
cost = {
  ('Pencils', 'Detroit', 'Boston'):   10,
  ('Pencils', 'Detroit', 'New York'): 20,
  ('Pencils', 'Detroit', 'Seattle'):  60,
  ('Pencils', 'Denver',  'Boston'):   40,
  ('Pencils', 'Denver',  'New York'): 40,
  ('Pencils', 'Denver',  'Seattle'):  30,
  ('Pens',    'Detroit', 'Boston'):   20,
  ('Pens',    'Detroit', 'New York'): 20,
  ('Pens',    'Detroit', 'Seattle'):  80,
  ('Pens',    'Denver',  'Boston'):   60,
  ('Pens',    'Denver',  'New York'): 70,
  ('Pens',    'Denver',  'Seattle'):  30 }
```

Comparison

Data *(cont'd)*

AMPL

```
param cost {COMMODITIES,ARCS} >= 0;
```

```
param cost

 [Pencils,*,*] (tr) Detroit  Denver :=
     Boston            10       40
    'New York'         20       40
     Seattle           60       30

 [Pens,*,*]    (tr) Detroit  Denver :=
     Boston            20       60
    'New York'         20       70
     Seattle           80       30   ;
```

Comparison

Model

gurobipy

```
m = Model('netflow')

flow = m.addVars(commodities, arcs, obj=cost, name="flow")

m.addConstrs(
  (flow.sum('*',i,j) <= capacity[i,j] for i,j in arcs), "cap")

m.addConstrs(
  (flow.sum(h,'*',j) + inflow[h,j] == flow.sum(h,j,'*')
     for h in commodities for j in nodes), "node")
```

alternatives

```
for i,j in arcs:
  m.addConstr(sum(flow[h,i,j] for h in commodities) <= capacity[i,j],
              "cap[%s,%s]" % (i,j))

m.addConstrs(
  (quicksum(flow[h,i,j] for i,j in arcs.select('*',j)) + inflow[h,j] ==
   quicksum(flow[h,j,k] for j,k in arcs.select(j,'*'))
     for h in commodities for j in nodes), "node")
```

Comparison

(Note on Summations)

gurobipy quicksum

```
m.addConstrs(
  (quicksum(flow[h,i,j] for i,j in arcs.select('*',j)) + inflow[h,j] ==
   quicksum(flow[h,j,k] for j,k in arcs.select(j,'*'))
      for h in commodities for j in nodes), "node")
```

quicksum (data)

A version of the Python `sum` function that is much more efficient for building large Gurobi expressions (LinExpr or QuadExpr objects). The function takes a list of terms as its argument.

Note that while `quicksum` is much faster than `sum`, it isn't the fastest approach for building a large expression. Use addTerms or the LinExpr() constructor if you want the quickest possible expression construction.

Comparison

Model *(cont'd)*

AMPL

```
var Flow {COMMODITIES,ARCS} >= 0;

minimize TotalCost:
   sum {h in COMMODITIES, (i,j) in ARCS} cost[h,i,j] * Flow[h,i,j];

subject to Capacity {(i,j) in ARCS}:
   sum {h in COMMODITIES} Flow[h,i,j] <= capacity[i,j];

subject to Conservation {h in COMMODITIES, j in NODES}:
   sum {(i,j) in ARCS} Flow[h,i,j] + inflow[h,j] =
   sum {(j,i) in ARCS} Flow[h,j,i];
```

Comparison

Solution

gurobipy

```
m.optimize()

if m.status == GRB.Status.OPTIMAL:
    solution = m.getAttr('x', flow)
        for h in commodities:
            print('\nOptimal flows for %s:' % h)
            for i,j in arcs:
                if solution[h,i,j] > 0:
                    print('%s -> %s: %g' % (i, j, solution[h,i,j]))
```

Solution *(cont'd)*

AMPL

```
ampl: solve;
Gurobi 8.0.0: optimal solution; objective 5500
2 simplex iterations

ampl: display Flow;

Flow [Pencils,*,*]
:         Boston 'New York' Seattle    :=
Denver      0        50        10
Detroit    50         0         0

 [Pens,*,*]
:         Boston 'New York' Seattle    :=
Denver     10         0        30
Detroit    30        30         0
;
```

Integration with Solvers

gurobipy

- Works closely with the Gurobi solver:
  callbacks during optimization, fast re-solves after problem changes
- Offers convenient extended expressions:
  min/max, and/or, if-then-else

AMPL

- Supports all popular solvers
- Extends to general nonlinear and logic expressions
  - Connects to nonlinear function libraries and user-defined functions
- Automatically computes nonlinear function derivatives

Integration with Applications

gurobipy

- Everything can be developed in Python
  - Extensive data, visualization, deployment tools available
- Limited modeling features also in C++, C#, Java

AMPL

- Modeling language extended with loops, tests, assignments
- Application programming interfaces (APIs) for calling AMPL from C++, C#, Java, MATLAB, Python, R
  - Efficient methods for data interchange
- Add-ons for streamlined deployment
  - QuanDec by Cassotis
  - Opalytics Cloud Platform

Software Survey

Solver-specific

- Associated with popular commercial solvers
- Executable and declarative alternatives

Solver-independent

- Support multiple solvers and solver types
- Mostly commercial/declarative and free/executable

Solver-Specific

Declarative, commercial

- OPL for CPLEX (IBM)
- MOSEL* for Xpress (FICO)
- OPTMODEL for SAS/OR (SAS)

Executable, commercial

- Concert Technology C++ for CPLEX
- gurobipy for Gurobi
- sasoptpy for SAS Optimization

Solver-Independent

Declarative, commercial

- AIMMS
- AMPL
- GAMS
- MPL

Declarative, free

- CMPL
- GMPL / MathProg

Executable, free

- PuLP; Pyomo / Python
- YALMIP; CVX / MATLAB
- JuMP / Julia
- FLOPC++ / C++

Trends

Commercial, declarative modeling systems

- Established lineup of solver-independent modeling systems that represent decades of development and support
- Continuing trend toward integration with popular programming languages and data science tools

Commercial, executable modeling systems

- Increasingly essential to commercial solver offerings
- Becoming the recommended APIs for solvers

Free, executable modeling systems

- A major current focus of free optimization software development
- Interesting new executable modeling languages have become easier to develop than interesting new solvers

Special Notes

Notable cases not detailed earlier . . .

- AIMMS *(solver-independent, commercial, declarative)* has extensive application development tools built in
- CMPL *(solver-independent, free, declarative)* has an IDE, Python and Java APIs, and remote server support
- GMPL/MathProg *(solver-independent, free, declarative)* is a free implementation of mainly a subset of AMPL
- JuMP *(solver-independent, free, executable)* claims greater efficiency through use of a new programming language, Julia
- MOSEL for Xpress *(solver-specific, commercial)* a hybrid of declarative and executable, has recently been made free and may accept other solvers

Balanced Assignment Revisited *in AMPL*

Sets, parameters, variables (for people)

```
set PEOPLE;   # individuals to be assigned

set CATEG;
param type {PEOPLE,CATEG} symbolic;

              # categories by which people are classified;
              # type of each person in each category

param numberGrps integer > 0;
param minInGrp integer > 0;
param maxInGrp integer >= minInGrp;

              # number of groups; bounds on size of groups

var Assign {i in PEOPLE, j in 1..numberGrps} binary;

              # Assign[i,j] is 1 if and only if
              # person i is assigned to group j
```

Modeling Language Formulation

Variables, constraints (for variation)

```
set TYPES {k in CATEG} := setof {i in PEOPLE} type[i,k];

              # all types found in each category

var MinType {k in CATEG, TYPES[k]};
var MaxType {k in CATEG, TYPES[k]};

              # fewest and most people of each type, over all groups

subj to MinTypeDefn {j in 1..numberGrps, k in CATEG, l in TYPES[k]}:
   MinType[k,l] <= sum {i in PEOPLE: type[i,k] = l} Assign[i,j];

subj to MaxTypeDefn {j in 1..numberGrps, k in CATEG, l in TYPES[k]}:
   MaxType[k,l] >= sum {i in PEOPLE: type[i,k] = l} Assign[i,j];

              # values of MinTypeDefn and MaxTypeDefn variables
              # must be consistent with values of Assign variables
```

$$y_{kl}^{\max} \geq \sum_{i \in P: t_{ik} = l} x_{ij}, \text{ for each } j = 1, \dots, G;\ k \in C, l \in T_k$$

Modeling Language Formulation

Objective, constraints (for assignment)

```
minimize TotalVariation:
   sum {k in CATEG, l in TYPES[k]} (MaxType[k,l] - MinType[k,l]);

              # Total variation over all types

subj to AssignAll {i in PEOPLE}:
   sum {j in 1..numberGrps} Assign[i,j] = 1;

              # Each person must be assigned to one group

subj to GroupSize {j in 1..numberGrps}:
   minInGrp <= sum {i in PEOPLE} Assign[i,j] <= maxInGrp;

              # Each group must have an acceptable size
```

$g^{\min} \leq \sum_{i \in P} x_{ij} \leq g^{\max}$, for each $j = 1, \ldots, G$

Balanced Assignment

Modeling Language Data

210 people

```
set PEOPLE :=
   BIW   AJH   FWI   IGN   KWR   KKI   HMN   SML   RSR   TBR
   KRS   CAE   MPO   CAR   PSL   BCG   DJA   AJT   JPY   HWG
   TLR   MRL   JDS   JAE   TEN   MKA   NMA   PAS   DLD   SCG
   VAA   FTR   GCY   OGZ   SME   KKA   MMY   API   ASA   JLN
   JRT   SJO   WMS   RLN   WLB   SGA   MRE   SDN   HAN   JSG
   AMR   DHY   JMS   AGI   RHE   BLE   SMA   BAN   JAP   HER
   MES   DHE   SWS   ACI   RJY   TWD   MMA   JJR   MFR   LHS
   JAD   CWU   PMY   CAH   SJH   EGR   JMQ   GGH   MMH   JWR
   MJR   EAZ   WAD   LVN   DHR   ABE   LSR   MBT   AJU   SAS
   JRS   RFS   TAR   DLT   HJO   SCR   CMY   GDE   MSL   CGS
   HCN   JWS   RPR   RCR   RLS   DSF   MNA   MSR   PSY   MET
   DAN   RVY   PWS   CTS   KLN   RDN   ANV   LMN   FSM   KWN
   CWT   PMO   EJD   AJS   SBK   JWB   SNN   PST   PSZ   AWN
   DCN   RGR   CPR   NHI   HKA   VMA   DMN   KRA   CSN   HRR
   SWR   LLR   AVI   RHA   KWY   MLE   FJL   ESO   TJY   WHF
   TBG   FEE   MTH   RMN   WFS   CEH   SOL   ASO   MDI   RGE
   LVO   ADS   CGH   RHD   MBM   MRH   RGF   PSA   TTI   HMG
   ECA   CFS   MKN   SBM   RCG   JMA   EGL   UJT   ETN   GWZ
   MAI   DBN   HFE   PSO   APT   JMT   RJE   MRZ   MRK   XYF
   JCO   PSN   SCS   RDL   TMN   CGY   GMR   SER   RMS   JEN
   DWO   REN   DGR   DET   FJT   RJZ   MBY   RSN   REZ   BLW ;
```

Modeling Language Data

4 categories, 18 types

```
set CATEG := dept loc rate title ;

param type:

     dept        loc      rate    title   :=

BIW   NNE   Peoria         A   Assistant
KRS   WSW   Springfield    B   Assistant
TLR   NNW   Peoria         B   Adjunct
VAA   NNW   Peoria         A   Deputy
JRT   NNE   Springfield    A   Deputy
AMR   SSE   Peoria         A   Deputy
MES   NNE   Peoria         A   Consultant
JAD   NNE   Peoria         A   Adjunct
MJR   NNE   Springfield    A   Assistant
JRS   NNE   Springfield    A   Assistant
HCN   SSE   Peoria         A   Deputy
DAN   NNE   Springfield    A   Adjunct

.......

param numberGrps := 12 ;
param minInGrp := 16 ;
param maxInGrp := 19 ;
```

Modeling Language Solution

Model + data = problem instance to be solved (CPLEX)

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

ampl: option solver cplex;
ampl: option show_stats 1;
ampl: solve;

2556 variables:
        2520 binary variables
        36 linear variables
654 constraints, all linear; 25632 nonzeros
        210 equality constraints
        432 inequality constraints
        12 range constraints
1 linear objective; 36 nonzeros.

CPLEX 12.8.0.0: optimal integer solution; objective 16
59597 MIP simplex iterations
387 branch-and-bound nodes
```

8.063 sec

Modeling Language Solution

Model + data = problem instance to be solved (Gurobi)

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

ampl: option solver gurobi;
ampl: option show_stats 1;
ampl: solve;

2556 variables:
        2520 binary variables
        36 linear variables
654 constraints, all linear; 25632 nonzeros
        210 equality constraints
        432 inequality constraints
        12 range constraints
1 linear objective; 36 nonzeros.

Gurobi 7.5.0: optimal solution; objective 16
338028 simplex iterations
1751 branch-and-cut nodes
```

66.344 sec

Modeling Language Solution

Model + data = problem instance to be solved (Xpress)

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

ampl: option solver xpress;
ampl: option show_stats 1;
ampl: solve;

2556 variables:
        2520 binary variables
        36 linear variables
654 constraints, all linear; 25632 nonzeros
        210 equality constraints
        432 inequality constraints
        12 range constraints
1 linear objective; 36 nonzeros.

XPRESS 8.4(32.01.08): Global search complete
Best integer solution found 16
6447 branch and bound nodes
```

61.125 sec

Modeling Language Formulation *(revised)*

Add bounds on variables

```
var MinType {k in CATEG, t in TYPES[k]}
   <= floor (card {i in PEOPLE: type[i,k] = t} / numberGrps);

var MaxType {k in CATEG, t in TYPES[k]
   >= ceil (card {i in PEOPLE: type[i,k] = t} / numberGrps);
```

```
ampl: include BalAssign+.run

Presolve eliminates 72 constraints.
...

Gurobi 7.5.0: optimal solution; objective 16
2203 simplex iterations
```

0.203 sec

Solvers for Model-Based Optimization

Off-the-shelf solvers for broad problem classes

Two main problem classes

- “Linear” solvers
- “Nonlinear” solvers

Other useful classes

- “Constraint” programming solvers
- “Global” optimization solvers

Typical Enhancements

Algorithms

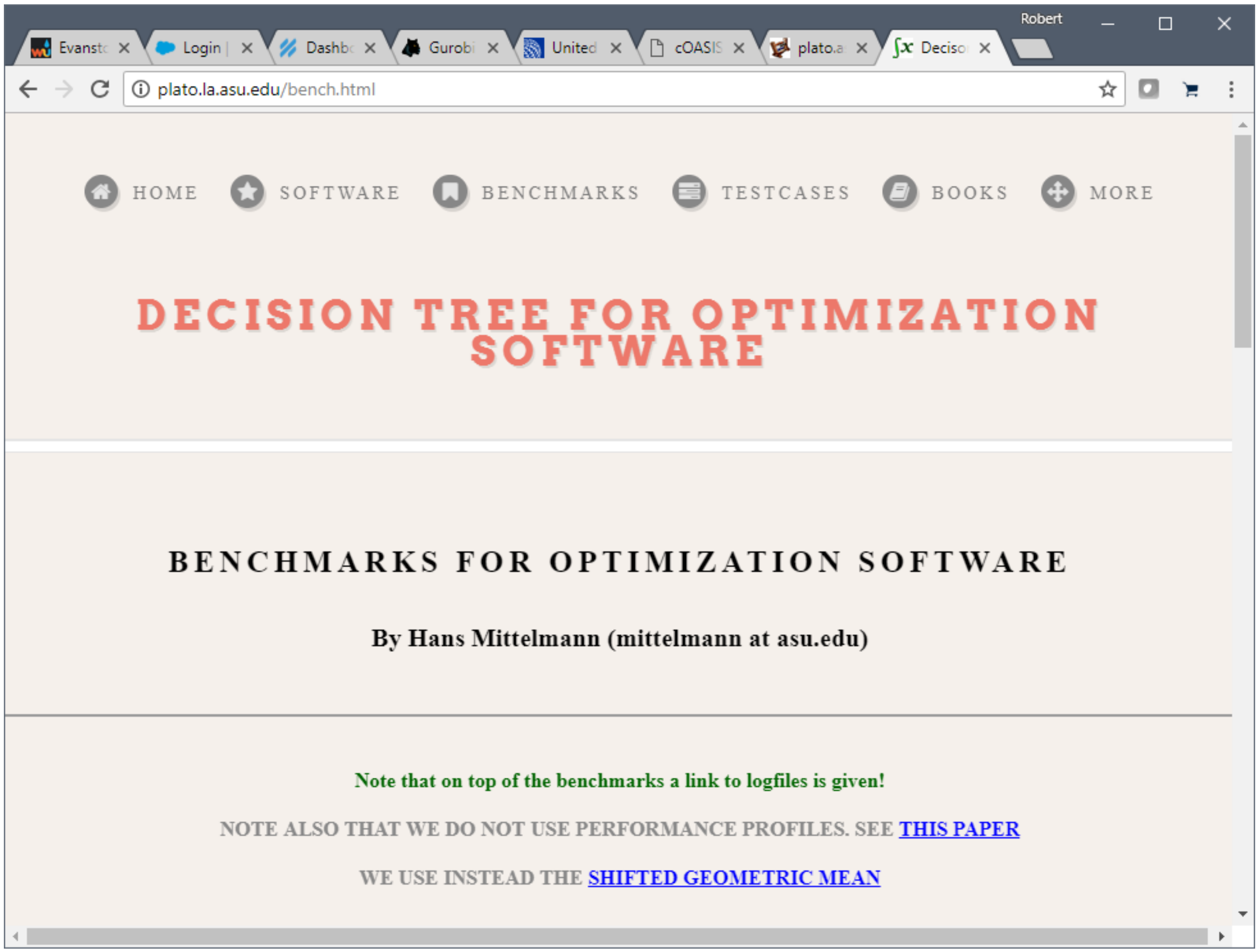
- Provisions for integer-valued variables
- Extensions of the technology to related problem classes
- Parallel implementation on multiple processor cores

Support for . . .

- Model-based optimization
- Application deployment
- Cloud-based services
  - Optimization on demand
  - Server clusters

Benchmarks

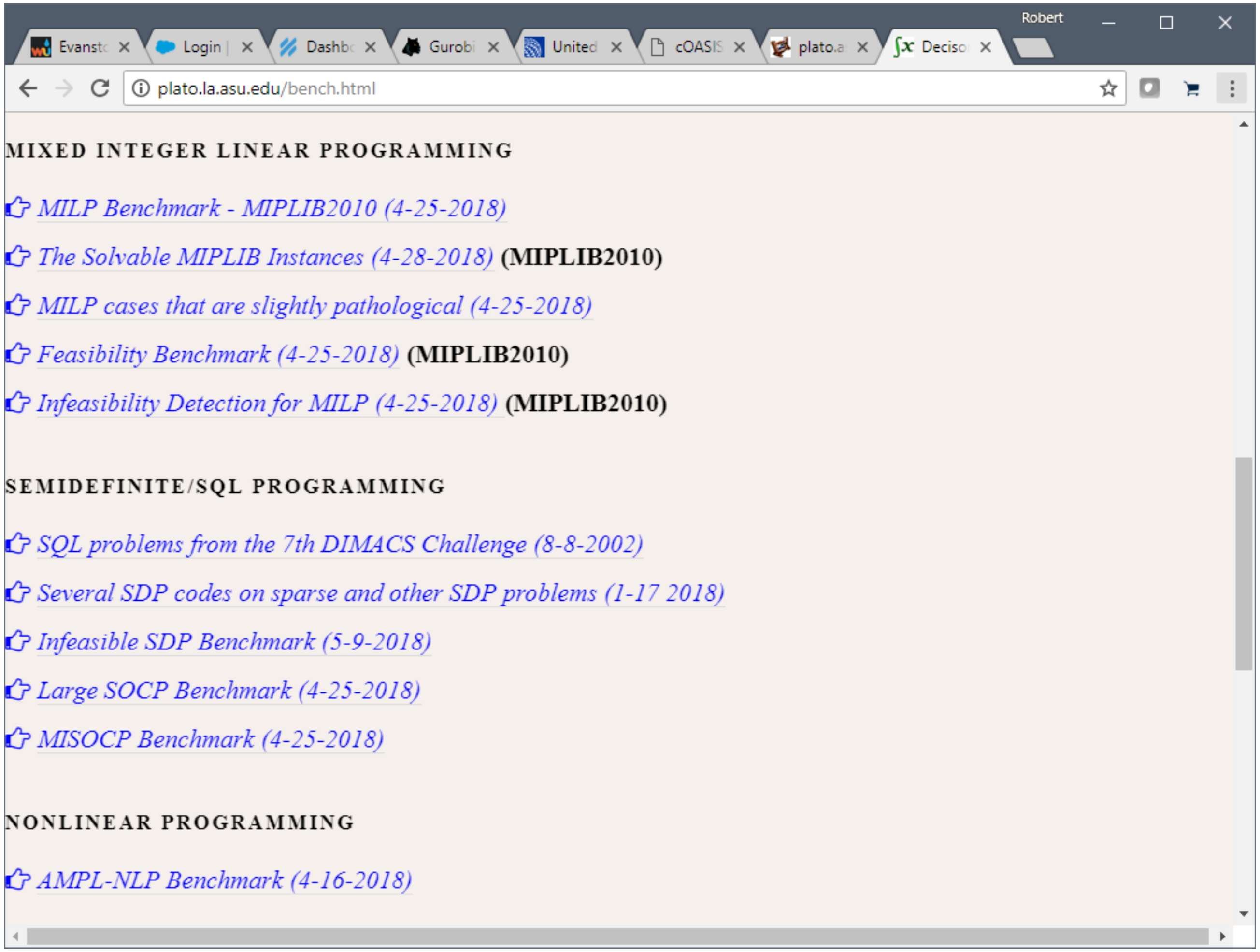
Prof. Hans Mittelmann's benchmark website



Off-the-Shelf Solvers

Benchmarks

By problem type and test set



Off-the-Shelf Solvers

Benchmarks

Documentation, summaries, links to detailed results

plato.asu.edu/ftp/milpc.html

```
25 Apr 2018 =======================================================
          Mixed Integer Linear Programming Benchmark (MIPLIB2010)
          =======================================================
                   H. Mittelmann (mittelmann@asu.edu)
```

The following codes were run with a limit of 2 hours on the MIPLIB2010 benchmark set with the MIPLIB2010 scripts (exc Matlab) on two platforms. 1/4 threads: Intel i7-4790K, 4 cores, 32GB, 4GHz, available memory 24GB; 12 threads: Intel Xeon X5680, 2x6 cores, 32GB, 3.33Ghz, available memory 24GB; These are updated and extended versions of the results produced for the MIPLIB2010 paper.

CPLEX-12.8.0: CPLEX
GUROBI-8.0.0 GUROBI
ug[SCIP/cpx/spx]-5.0.0: Parallel development version of SCIP (SCIP+CPLEX/SOPLEX on 1 thread)
CBC-2.9.8: CBC
XPRESS-8.4.0: XPRESS
MATLAB-2018a: MATLAB (intlinprog)
MIPCL-1.5.1: MIPCL
SAS-OR-14.3: SAS

Table for single thread, Result files per solver, Log files per solver

Table for 4 threads, Result files per solver, Log files per solver

Table for 12 threads, Result files per solver, Log files per solver

Statistics of the problems can be obtained from the MIPLIB2010 webpage.

+++

Unscaled and scaled shifted geometric means of run times

```
All non-successes are counted as max-time.
The third line lists the number of problems (87 total) solved.
```

1 thr	CBC	CPLEX	GUROBI	SCIPC	SCIPS	XPRESS	MATLAB	SAS
unscal	1639	72.2	47.8	281	329	76.7	2667	120
scaled	34	1.51	1	5.88	6.87	1.61	55.8	2.52
solved	53	87	87	83	76	86	39	84

“Linear” Solvers

Linear objective and constraints

- Continuous variables
  - Primal simplex, dual simplex, interior-point
- Integer (including zero-one) variables
  - Branch-and-bound + feasibility heuristics + cut generation
  - Automatic transformations to integer:
    piecewise-linear, discrete variable domains, indicator constraints
  - “Callbacks” to permit problem-specific algorithmic extensions

Quadratic extensions

- Convex elliptic objectives and constraints
- Convex conic constraints
- Variable × binary in objective
  - Transformed to linear (or to convex if binary × binary)

“Linear” Solvers *(cont'd)*

CPLEX, Gurobi, Xpress

- Dominant commercial solvers
- Similar features
- Supported by many modeling systems

SAS Optimization, MATLAB intlinprog

- Components of widely used commercial analytics packages
- SAS performance within 2x of the “big three”

MOSEK

- Commercial solver strongest for conic problems

CBC, MIPCL, SCIP

- Fastest noncommercial solvers
- Effective alternatives for easy to moderately difficult problems
- MIPCL within 7x on some benchmarks

Special Notes

Special abilities of certain solvers . . .

- CPLEX has an option to handle nonconvex quadratic objectives
- MOSEK extends to general semidefinite optimization problems
- SCIP extends to certain logical constraints

“Nonlinear” Solvers

Continuous variables

- Smooth objective and constraint functions
- Locally optimal solutions
- Variety of methods
  - Interior-point, sequential quadratic, reduced gradient

Extension to integer variables

“Nonlinear” Solvers

Knitro

- Most extensive commercial nonlinear solver
- Choice of methods; automatic choice of multiple starting points
- Parallel runs and parallel computations within methods
- Continuous and integer variables

CONOPT, LOQO, MINOS, SNOPT

- Highly regarded commercial solvers for continuous variables
- Implement a variety of methods

Bonmin, Ipopt

- Highly regarded free solvers
  - Ipopt for continuous problems via interior-point methods
  - Bonmin extends to integer variables

“Global” Solvers

Nonlinear + global optimality

- Substantially harder than local optimality
- Smooth nonlinear objective and constraint functions
- Continuous and integer variables

BARON

- Dominant commercial global solver

Couenne

- Highly regarded noncommercial global solver

LGO

- High-quality solutions, may be global
- Objective and constraint functions may be nonsmooth

“Constraint” Solvers

Motivated by “constraint programming”

- Logic directly in constraints using *and, or, not* operators
- Range of other nonsmooth and logic operators
- “All different” and other special constraints
- Variables in subscripts to other variables and parameter
- *Encoding of logic in binary variables not necessary*

Alternative solvers employed

- Globally optimal solutions
  - Branching search like other “integer” solvers
- Not originally model-based,
  but trend has been toward model-based implementations
- More general modeling languages needed

"Constraint" Solvers

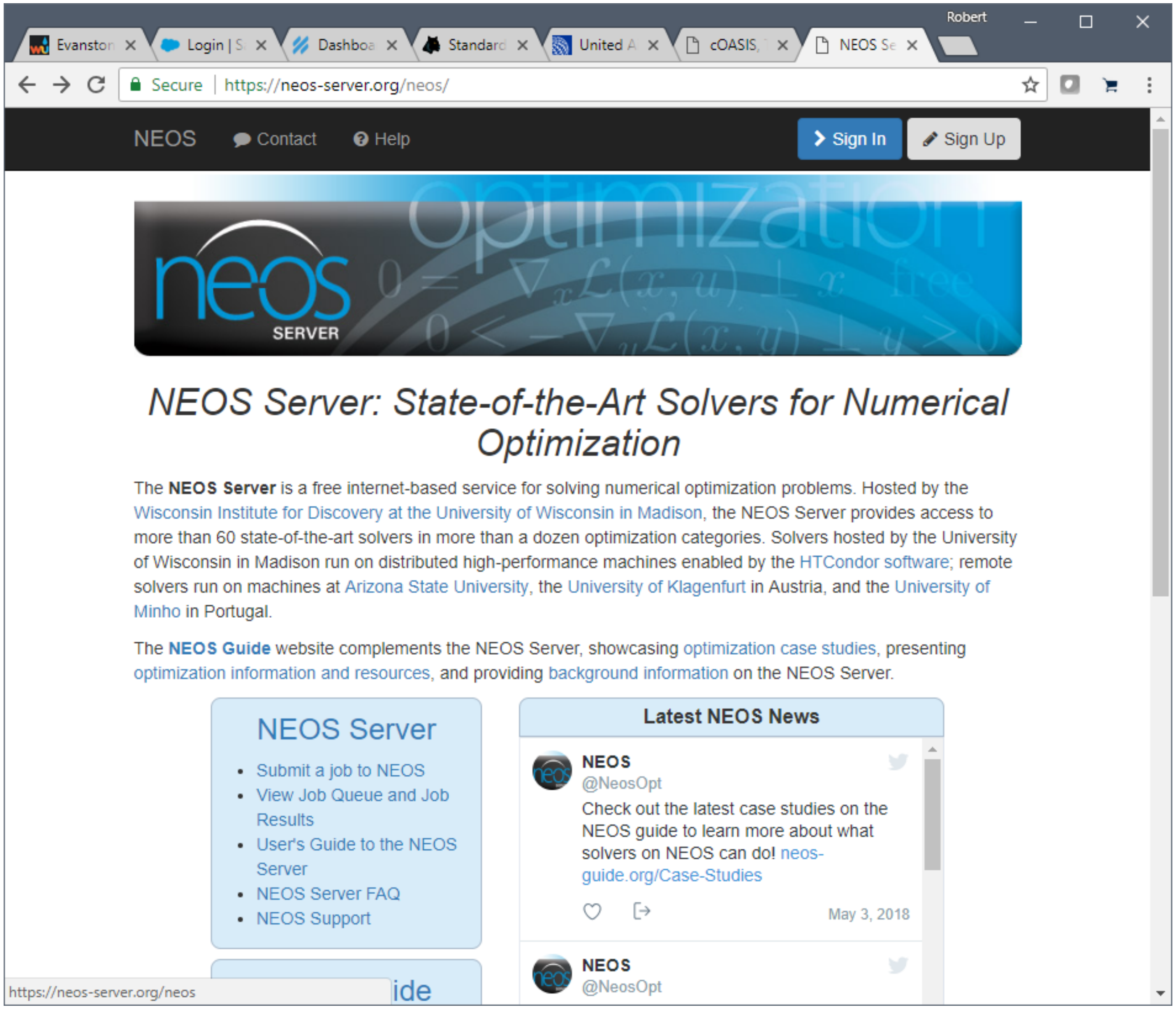
IBM ILOG CP

- Established commercial constraint programming solver
- Solver-specific modeling language support
  - C++ API combining model-based and method-based approaches
  - C++ "Concert Technology" executable modeling language
  - OPL declarative modeling language
- Some support from solver-independent languages

Gecode, JaCoP

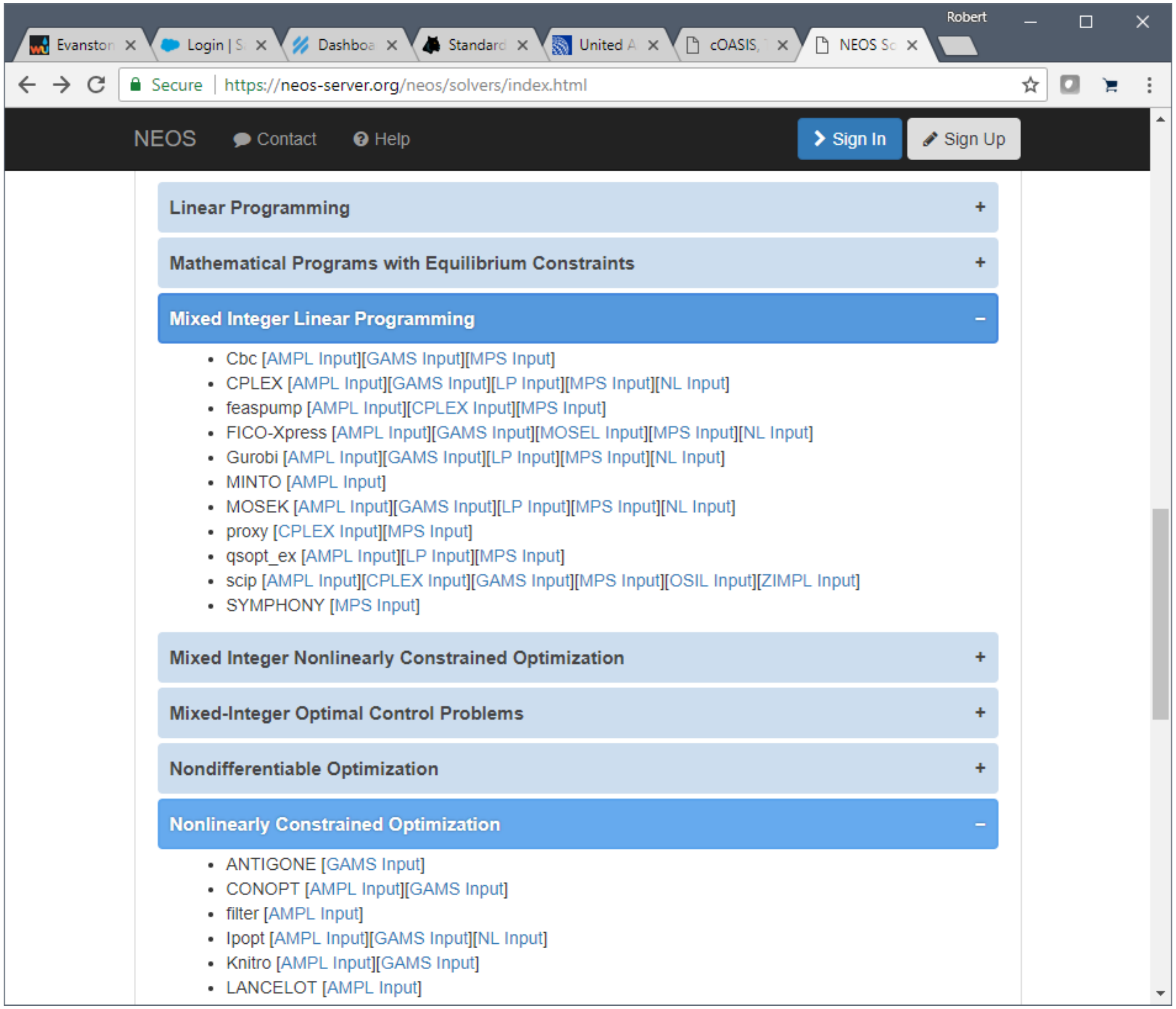
- Notable noncommercial solvers
- Limited modeling language support

Curious? Try Them Out on NEOS!



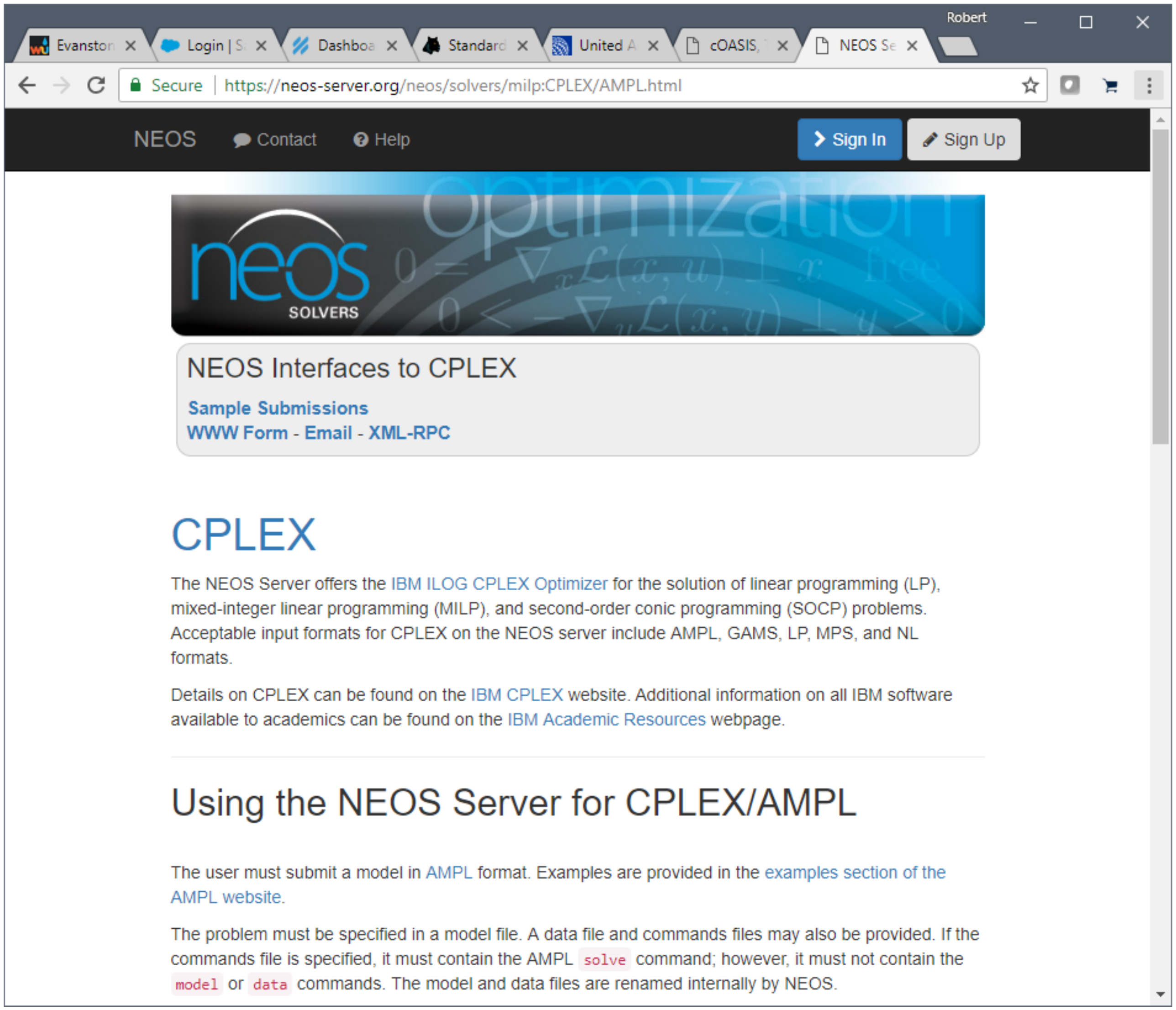
NEOS Server

Solver & Language Listing



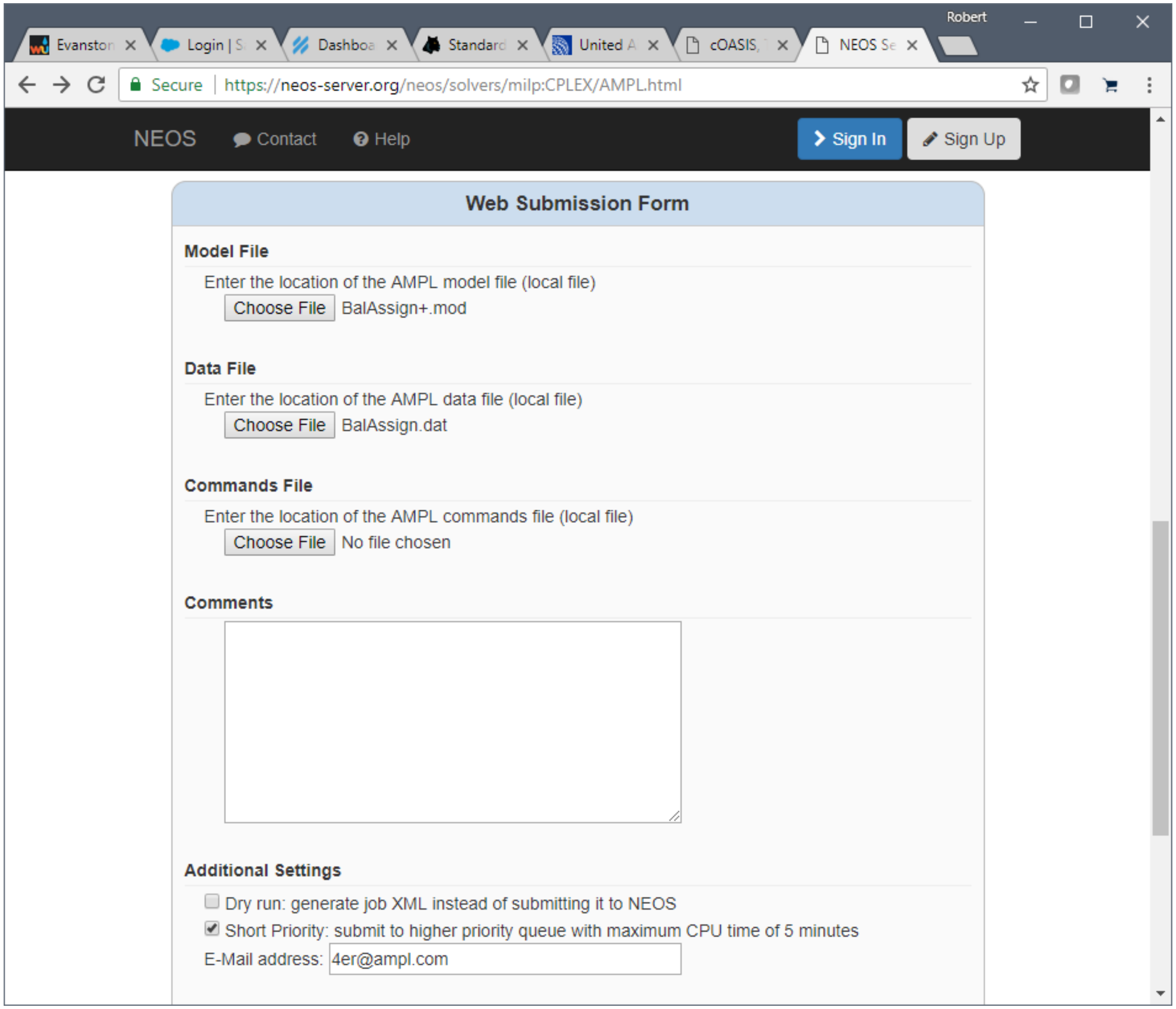
NEOS Server

Input Page for CPLEX using AMPL



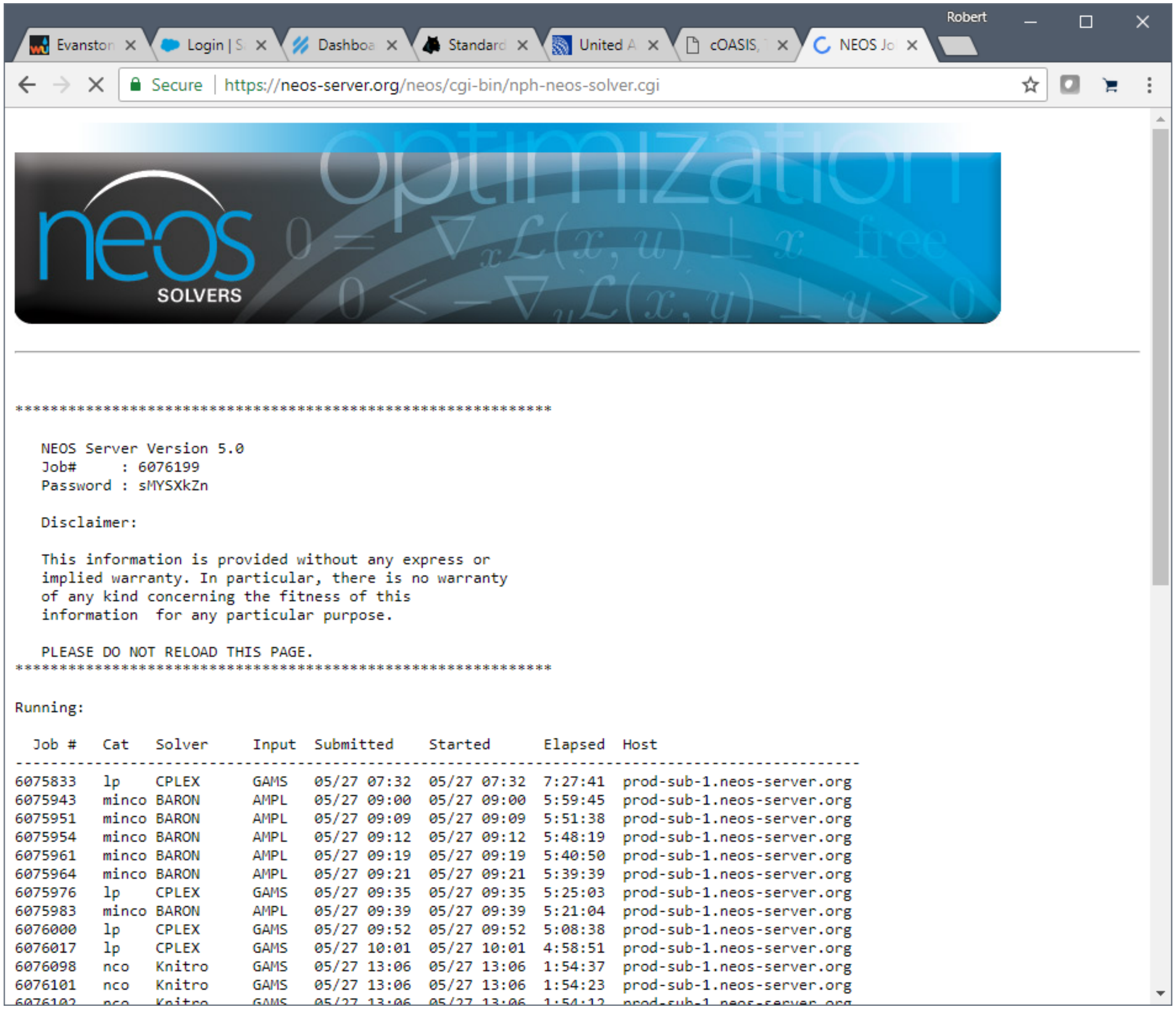
NEOS Server

Input Page *(cont'd)*



NEOS Server

Queue Page



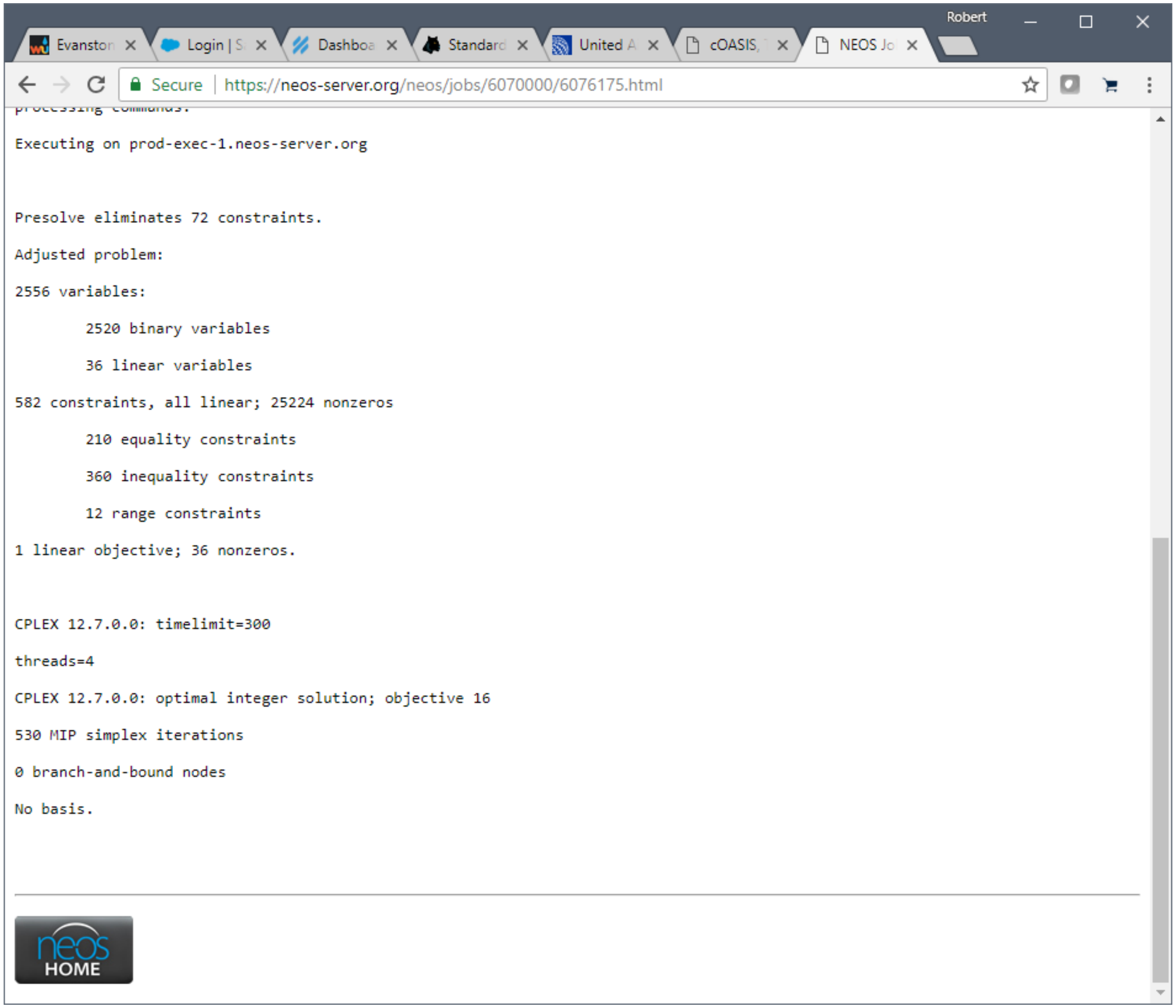
NEOS Server

Output Page


```
*************************************************************

   NEOS Server Version 5.0
   Job#     : 6076175
   Password : MxudUoip
   User     : None
   Solver   : milp:CPLEX:AMPL
   Start    : 2018-05-27 14:33:37
   End      : 2018-05-27 14:33:42
   Host     : NEOS HTCondor Pool

   Disclaimer:

   This information is provided without any express or
   implied warranty. In particular, there is no warranty
   of any kind concerning the fitness of this
   information  for any particular purpose.
*************************************************************
File exists

You are using the solver cplexamp.

Checking ampl.mod for cplex_options...

Executing AMPL.

processing data.

processing commands.

Executing on prod-exec-1.neos-server.org
```

NEOS Server

Output Page *(cont'd)*



About the NEOS Server

Solvers

- 18 categories, 60+ solvers
- Commercial and noncommercial choices
- Almost all of the most popular ones

Inputs

- AMPL, GAMS, and others
- MPS, LP, and other lower-level problem formats

Interfaces

- Web browser
- Special solver (“Kestrel”) for AMPL and GAMS
- Python API

About the NEOS Server *(cont'd)*

Limits

- 8 hours
- 3 GBytes

Operation

- Requests queued centrally,
  distributed to various servers for solving
- 650,000+ requests served in the past year,
  about 1800 per day or 75 per hour
- 17,296 requests on peak day (15 March 2018)